Universität Stuttgart

Shipping von Funktionen und Daten in heterogenen IT-Umgebungen

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

Michael Zimmermann

aus Stuttgart

Hauptberichter:	Prof. Dr. Dr. h. c. Frank Leymann
Mitberichter:	Prof. Dr. Guido Wirtz
Tag der mündlichen Prüfung:	2. Juni 2023

Institut für Architektur von Anwendungssystemen

der Universität Stuttgart

2023

Inhaltsverzeichnis

Zusammenfassung

Die Analyse von generierten und gesammelten Daten ist ein wichtiger Erfolgsfaktor für moderne Unternehmen. Mittels Datenanalysen können beispielsweise Muster identifiziert sowie allgemein neue Erkenntnisse gewonnen werden. Daraus folgend können unter anderem Geschäftsprozesse optimiert, Kosten reduziert oder die Produktivität erhöht werden. Hierzu müssen zunächst geeignete Anwendungen zur Analyse der Daten entwickelt werden und weiterhin diese Anwendungen mit den zu analysierenden Daten zusammengebracht werden. Aufgrund der fortschreitenden Entwicklungen und dem Wachstum im Bereich Internet of Things (IoT) sowie der damit zunehmenden Vernetzung physischer Objekte steigt die Menge an generierten und verfügbaren Daten allerdings kontinuierlich an. Das Bedürfnis einer hohen Datenlokalität rückt dadurch in den Fokus und das Finden von geeigneten Umgebungen zur Ausführung der Analyselogik wird zu einer wichtigen und komplexen Aufgabe. Nicht nur stehen zur Datenverarbeitung verschiedene Computing-Paradigmen wie Cloud, Fog und Edge zur Verfügung, sondern es müssen zudem verschiedene Aspekte hierbei betrachtet werden. Unter anderem müssen Anforderungen hinsichtlich der benötigten Leistung in der Ausführungsumgebung sowie Sicherheitsanforderungen, zum Beispiel bezüglich des Datenschutzes, erfüllt werden. Zum anderen müssen Faktoren wie die Datenmengen der einzelnen zu analysierenden Datensätze, die aktuellen Lokalitäten dieser Datensätze, die komponentenbasierte Struktur der Anwendung sowie der Datenfluss der Anwendung berücksichtigt werden.

Diese Arbeit stellt daher eine Methode sowie verschiedene Konzepte vor, um den Nutzer systematisch über den Lebenszyklus einer komponentenbasierten, datenverarbeitenden Anwendung hinweg, bei der Modellierung, Implementierung, Gruppierung, Platzierung und Bereitstellung der Anwendung zu unterstützen. Als Grundlage hierfür wird die sogenannte Shipping-Methode in dieser Arbeit eingeführt. Diese ermöglicht die Kollaboration verschiedener Experten und das Wiederverwenden von Expertenwissen. Weiterhin können Schritte der Methode sowohl manuell, semi-automatisiert als auch vollständig automatisiert durchgeführt werden. Die Methode basiert auf der deklarativen Modellierungssprache D^3M, welche ebenfalls vorgestellt wird. D^3M ermöglicht neben der reinen Modellierung einer klassischen Anwendungstopologie, auch die Modellierung von zusätzlichen Informationen, die für die Gruppierung und Platzierung der Anwendungen benötigt werden. Dazu gehören beispielsweise Informationen über die einzelnen zu verarbeitenden Datensätze und den Datenfluss innerhalb der Anwendung. Zur Vereinfachung kann ein solches D^3M-Modell weiterhin auch aus einem Datenflussmodell generiert und die benötigten Informationen automatisiert daraus abgeleitet werden. Um die Implementierung von verteilbaren und interagierenden Komponenten eines D^3M-Modells zu unterstützen, wird ein Programmiermodell vorgestellt, das mittels der Generierung von verschiedenen Code-Fragmenten den Endpunktaustausch und die Kommunikation zwischen diesen Komponenten abstrahiert. Um eine sichere und datenschutzkonforme Platzierung der Anwendungskomponenten sowie Verarbeitung der Daten gewährleisten zu können, wird ein Konzept zur Modellierung von wiederverwendbaren Sicherheitsrichtlinien sowie ein automatisiertes Verfahren zur Kontrolle und Einhaltung dieser Richtlinien eingeführt.

Zur Validierung der praktischen Umsetzbarkeit der im Rahmen dieser Arbeit vorgestellten Konzepte wird weiterhin die Gesamtarchitektur eines entsprechenden Frameworks zur Realisierung der Shipping-Methode präsentiert. Darüber hinaus werden die hierzu angefertigten prototypischen Implementierungen vorgestellt und die Wiederverwendung der Konzepte und Prototypen in anderen Arbeiten und Projekten diskutiert.

Abstract

Analyzing generated and collected data is an important success factor for modern companies. By analyzing data, for example, patterns can be identified, and new insights can be gained in general. As a result, business processes can be optimized, costs can be reduced and productivity can be increased. In order to do this, first of all suitable applications must be developed for analyzing the data and subsequently these applications must be brought together with the data to be analyzed. However, as a result of the ongoing developments and growth in the Internet of Things (IoT) area as well as the corresponding increase of connected physical objects, the amount of generated and available data is continuously increasing. As a consequence, the requirement for high data locality is becoming more and more important, and finding suitable environments for executing the analysis logic is becoming an important and complex task. Not only do various computing paradigms such as cloud, fog and edge exist for data processing, but there are also various additional aspects that need to be taken into account. Among other things, requirements regarding the required performance in the execution environment as well as security requirements, for example regarding data protection, must be fulfilled. Furthermore, factors such as the data volumes of the individual data sets to be analyzed, the current localities of these data sets, the component-based structure of the application, as well as the data flow of the application must be taken into account.

Therefore, this thesis introduces a method as well as different concepts to support the user systematically over the lifecycle of a component-based, data-processing application, during the modeling, implementation, distribution, placement and deployment of the application. As a basis for this, the so-called Shipping-Method is introduced in this thesis. The method enables the collaboration of different experts and the reuse of expert knowledge. Furthermore, steps of the method can be performed manually, semi-automated as well as fully automated. The method is based on the declarative modeling language D^3M, which is also introduced in this thesis. In addition to the plain modeling of a classical application topology, D^3M also allows the modeling of additional information that is needed for the distribution and placement of the applications. This includes, for example, information about the individual data sets to be processed and the data flow within the application. Furthermore, for simplification purposes, such a D^3M model can also be generated from a data flow model and the required information can be derived from it in an automated manner. To support the implementation of distributable and interacting components modeled with D^3M, a programming model is presented that uses the generation of different types of code fragments to abstract the endpoint exchange and the communication between these components. In order to ensure a secure and privacy-compliant placement of application components as well as processing of data, a concept for modeling reusable security policies as well as an automated mechanism for controlling and enforcing these policies is introduced.

To validate the practical feasibility of the presented concepts, the overall architecture of a corresponding framework supporting the Shipping-Method is presented. Furthermore, the prototypical implementations developed for this thesis will be presented and the possibilities of reusing the concepts and prototypes in other works and projects will be discussed.

Kapitel 1

Einleitung und Motivation

Fortschreitende Entwicklungen in Cloud Computing [AFG+10], Internet of Things (IoT) [AIM10] und Big Data [GH15] sowie der zunehmende Einsatz von Cyber-physischen Systemen (CPS) [GPGV14] führen zu einer wachsenden Menge an generierten, gespeicherten und zu verarbeitenden Daten [Mar18; SR20]. Viele Arbeitsfelder können von den sich daraus ergebenden Möglichkeiten profitieren, beispielsweise durch die Anwendung von Big Data Analysen [Rus11] oder durch Kosteneinsparungen durch verbrauchsbasierter Abrechnungsmodelle (Pay-per-Use) in der Cloud [Ley09], zum Beispiel in den Bereichen Mobilität [GHH+17], Gesundheitswesen [HAR14], Energiemanagement [SM15] und Simulationswissenschaften [Sza11].

In der Industrie 4.0 [SGG+13] sind beispielsweise die Produktionsprozessoptimierung von Fertigungsstrecken sowie die vorausschauende Wartung (Predictive Maintenance) zwei entscheidende Faktoren um den ungewollten Stillstand von Maschinen zu reduzieren und somit eine höhere Produktivität zu erreichen [KWXD17; Mob02]. Um die hierfür nötigen Datenanalysen zu ermöglichen, müssen typischerweise Daten aus verschiedenen heterogenen Quellen sowie Standorten kombiniert, analysiert und verglichen werden.

Aufgrund der zunehmenden Datenmenge wird die Cloud und das Internet jedoch immer mehr zum Engpass und der Transfer von Daten zwischen unterschiedlichen Lokalitäten zum Problem, insbesondere wenn zeitkritische Verarbeitungen von Daten durchgeführt werden sollen [CZ14; DWX+20; ZCZ+18]. Je nach Anwendungsfall sind jedoch nicht nur die zu verarbeitenden Daten selbst, sondern auch die datenverarbeitenden Komponenten über verschiedene Umgebungen hinweg verteilt, beispielsweise öffentliche und private Clouds sowie Fog und Edge Clouds [SDL20]. Um die Daten analysieren zu können, beispielsweise um neue Erkenntnisse und Optimierungspotenziale zu erhalten, müssen die zu verarbeitenden Daten entsprechend jeweils mit der datenverarbeitenden Funktionalität zusammengebracht werden.

Um dies zu erreichen, können die beiden Paradigmen *Function Shipping* sowie *Data Shipping* angewandt werden [FBC+16; ZBF+17b]. Die Bezeichnungen und Ansätze stammen ursprünglich aus dem Bereich der Datenbanken[1] [BFG+95; FJK96] und werden im Rahmen dieser Arbeit entsprechend auf den Kontext von komponentenbasierten Anwendungen [HC01] übertragen verwendet. Function Shipping beschreibt dabei, dass die Funktionalität, beispielsweise die datenverarbeitende Komponente einer Anwendung, in der Nähe der zu verarbeitenden Daten bereitgestellt und betrieben wird. Data Shipping hingegen beschreibt, dass die Daten zu einer entfernt betriebenen Anwendungskomponente übertragen und entsprechend dort verarbeitet werden. Weiterhin ist auch eine Kombination aus beiden Paradigmen möglich.

Die Verteilung von Anwendungen führt jedoch grundsätzlich zu verschiedenen Herausforderungen: Infrastrukturkomponenten zum Betrieb der Anwendungskomponenten, wie beispielsweise physische Hardware oder virtuelle Maschinen, müssen vorbereitet und gestartet werden. Benötigte Middleware- und Anwendungskomponenten müssen bereitgestellt und konfiguriert werden. Bei komplexen Anwendungen und Bereitstellungsprozessen ist die manuelle Durchführung zudem fehleranfällig, zeitaufwändig und daher nicht effizient [BBK+13a; EKK+06]. Die Bereitstellung und Verwaltung

[1] Neben *Function Shipping* werden hier unter anderem auch die Bezeichnungen *Query Shipping* sowie *Function Request Shipping* genutzt und bezeichnen die Ausführung der Funktionalität in unmittelbarer Nähe zu den Daten [CDP86; VÖU04].

von Anwendungen kann mithilfe verschiedener Bereitstellungstechnologien automatisiert werden, zum Beispiel (i) anbieterspezifischen Technologien, (ii) anbieterunabhängigen aber plattformspezifischen Technologien, (iii) universell einsetzbaren Technologien oder (iv) anbieterunabhängigen sowie technologieunabhängigen Standards [WBF+19]. Allerdings ist die sinnvolle Gruppierung und Platzierung von Komponenten einer datenverarbeitenden Anwendung von verschiedensten Faktoren abhängig: Zum Beispiel den Anforderungen der Anwendungskomponenten sowie der Daten, sowohl technischer als auch regulatorischer Art, beispielsweise um gesetzliche oder unternehmensinterne Datenschutzrichtlinien einzuhalten. Weiterhin sind die Größen der einzelnen zu verarbeitenden Datensätze, deren jeweilige Lokalität sowie der grundsätzliche Datenfluss der Anwendung wichtige Merkmale, die bei der Entscheidung, wie die Anwendung gruppiert und wo welche Komponenten bereitgestellt werden sollen berücksichtigt werden müssen.

In der vorliegenden Arbeit wird daher die automatisierte Bereitstellung verteilter und komponentenbasierter Anwendungen unter Berücksichtigung der oben genannten Faktoren sowie insbesondere mit Hinblick auf eine intelligente und datenflussoptimierte Gruppierung und Platzierung der datenverarbeitenden Anwendungskomponenten sowie Datensätze betrachtet.

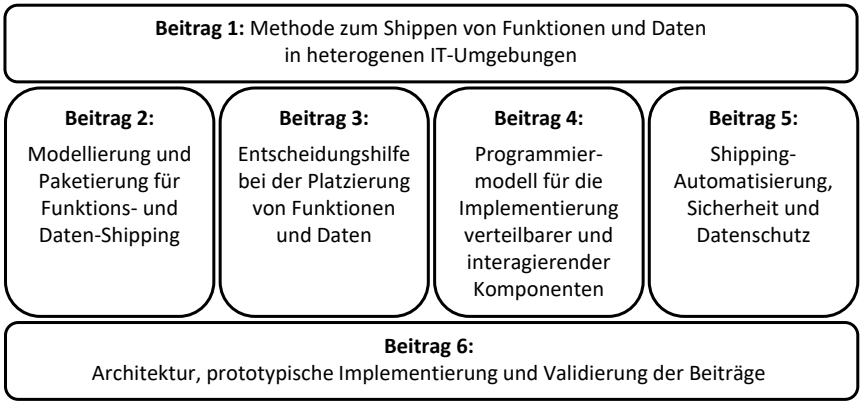


Abbildung 1.1: Übersicht der Forschungsbeiträge

1.1 Problemstellung und Forschungsbeiträge

In diesem Abschnitt werden die Problemstellungen beschrieben, die sich von den zuvor diskutierten Herausforderungen und Zielen ableiten. Weiterhin wird vorgestellt, wie die einzelnen Forschungsbeiträge dieser Arbeit diese Problemstellungen adressieren. Abbildung 1.1 stellt hierfür eine Übersicht der verschiedenen Forschungsbeiträge dar und veranschaulicht deren Zusammenhang: In Forschungsbeitrag 1 wird eine Methode zur systematischen Entwicklung und Verteilung von Anwendungen vorgestellt, welche unter anderem durch die Forschungsbeiträge 2-5 realisiert wird. In Forschungsbeitrag 6 wird die Architektur der Prototypen, die zur Umsetzung und Validierung der einzelnen Beiträge entwickelt wurden, vorgestellt und diskutiert. Die konzeptionellen Forschungsbeiträge 2-5 sowie deren Prototypen (Forschungsbeitrag 6) können weiterhin auch jeweils unabhängig der Shipping-Methode aus Forschungsbeitrag 1 zur Modellierung von Anwendungen, der vereinfachten Implementierung und datenflussoptimierten Verteilung von Komponenten, der Erstellung eigenständiger Deployment-Pakete sowie dem sicheren Deployment von Anwendungen verwendet werden.

1.1.1 Methode zum Shippen von Funktionen und Daten

Es existieren verschieden Ansätze und Konzepte um Komponenten und Daten einer Anwendung zu verteilen und an bestimmten Lokalitäten zu platzieren, wovon einige im Rahmen von Kapitel 2 vorgestellt und diskutiert werden. Viele der Ansätze beschränken sich jedoch auf eine rein konzeptionelle Lösung und bieten dem Anwender keine technischen Möglichkeiten zur Ausführung. Andere Arbeiten, die auch eine technische Umsetzung bieten, sind weiterhin häufig an bestimmte Technologien oder Infrastrukturen gebunden oder ermöglichen nur das Shipping einzelner Komponenten ohne die Gesamtstruktur einer Anwendung zu betrachten. Ebenfalls liegen den unterschiedlichen Lösungsansätzen unterschiedliche Motivationen zugrunde und haben dementsprechend verschiedene Optimierungsziele im Fokus, wie zum Beispiel die Reduzierung von Kosten. Eine integrierte Vorgehensweise

für die Verteilung von datenverarbeitenden Anwendungen mit dem Fokus auf einer datenflussoptimierten und sicheren Platzierung der Komponenten und Daten sowie einer automatisierten und technologieunabhängigen Bereitstellung der Anwendung ist jedoch nicht vorhanden. Im Rahmen dieser Arbeit wird daher eine entsprechende Methode sowie prototypische Implementierungen vorgestellt, welche den Anwender bei der Gruppierung und Platzierung von Komponenten und Daten unter anderem mit Hinblick auf den Datenfluss der Anwendung sowie weiteren Anforderungen unterstützen.

> ***Forschungsbeitrag 1:*** *Methode zum Shippen von Funktionen und Daten in heterogenen IT-Umgebungen.*
> Die in der vorliegenden Arbeit eingeführte Shipping-Methode ermöglicht unter anderem das technologieunabhängige sowie anbieterunabhängige Shipping sowohl einzelner Anwendungskomponenten als auch ganzer Softwarestacks sowie Daten, mit dem Fokus auf einer datenflussoptimierten Gruppierung und Platzierung der einzelnen Komponenten und Daten einer Anwendung. Aufgrund des modellbasierten Ansatzes unterstützt die vorgestellte Methode dieser Arbeit den Anwender zudem weiterhin systematisch über den gesamten Lebenszyklus einer Anwendung, von der Modellierung der Anwendung, über die Implementierung der einzelnen Komponenten, das automatisierte Deployment sowie das Management der entsprechenden Anwendung. Die Mehrzahl der einzelnen Schritte der Methode können dabei vollständig automatisiert oder semi-automatisiert durchgeführt werden. Um jedoch eine manuelle Kontrolle sowie Beeinflussung durch den Anwender zu ermöglichen, ist ein manueller Kontrollschritt explizit innerhalb der Methode vorgesehen. Aufgrund des modellgetriebenen Ansatzes kann zudem jederzeit der aktuelle Stand der Modellierung eingesehen und gegebenenfalls bei Bedarf modifiziert werden.

1.1.2 Modellierung und Paketierung für Funktions- und Daten-Shipping

Um die Shipping-Methode durchführen zu können und die genannten Ziele zu erreichen, wird eine entsprechende Modellierungssprache zur Modellierung der Anwendungen benötigt. Die bestehenden Ansätze zur Modellierung von Anwendungen, um diese anschließend automatisiert bereitstellen und verwalten zu können, lassen sich in imperative sowie deklarative Ansätze einteilen. Bei imperativen Ansätzen muss der Nutzer, um einen gewünschten Zustand zu erreichen, sowohl die einzelnen auszuführenden Operationen sowie deren genaue Reihenfolge im Detail beschreiben. Ein typisches Beispiel hierfür ist ein Skript, das alle benötigten Befehle enthält. Bei deklarativen Ansätzen dagegen, kann der Nutzer den gewünschten Zielzustand direkt modellieren. Die zum Erreichen des Zustands benötigten Schritte werden dann von der Ausführungsumgebung ermittelt und durchgeführt. Da der Fokus dieser Arbeit aber vor allem darauf liegt, den Nutzer bei der Verteilung seiner Anwendung und der Platzierung der Komponenten und Daten zu unterstützen sowie weiterhin ihn bei der konkreten Durchführung dessen zu begleiten (von der Modellierung, über die Implementierung bis zur Bereitstellung und Verwaltung der Anwendung), werden hierfür zusätzliche Informationen benötigt, die über die reine Bereitstellung und Verwaltung einer Anwendung hinausgehen. Beispielweise werden die jeweiligen Mengen der zu verarbeitenden Daten sowie weitere Informationen bezüglich des Datenflusses innerhalb der Anwendung benötigt. Darüber hinaus muss ein portables und zur Modellierungssprache kompatibles Format zur Paketierung der modellierten Anwendung sowie aller benötigten Artefakte gefunden werden, das die automatisierte Bereitstellung der Anwendung ermöglicht.

Forschungsbeitrag 2: *Modellierung und Paketierung für Funktions- und Daten-Shipping.*
In der vorliegenden Arbeit wird mit dem D^{3}M-Metamodell (Declarative Deployment and Component Distribution Model) eine deklarative Modellierungssprache eingeführt, die es neben der reinen Modellierung von Komponenten und deren Relationen zueinander auch ermöglicht, die zur Durchführung der Shipping-Methode zusätzlich benötigten Datenflussinformationen sowie Schnittstellen der datenverarbeitenden Anwendungskomponenten zu modellieren. Weiterhin wird ein systematisches Vorgehen vorgestellt, um eigenständige Deployment-Pakete zu erstellen und damit die Verteilung, Bereitstellung und Verwaltung der modellierten Anwendungen auch in IT-Umgebungen mit beschränktem Internetzugriff zu ermöglichen.

1.1.3 Entscheidungshilfe bei der Platzierung von Funktionen und Daten

Über die Verlagerung von lokalen Anwendungen in die Cloud sowie der grundsätzlichen Platzierung von Komponenten in Fog und Edge Clouds existieren viele wissenschaftliche Arbeiten. Die unterschiedlichen Möglichkeiten zur Verteilung von Komponenten bieten verschiedene Vor- sowie Nachteile, die je nach Anwendungsfall und Anforderungen unterschiedlich gewichtet werden müssen. So kann das Fog und Edge Computing zum Beispiel eine höhere Zeiteffizienz ermöglichen, falls die Berechnungen dadurch in unmittelbarer Nähe zu den Daten durchgeführt werden und nicht zunächst in die Cloud gesendet werden müssen. Auf der anderen Seite bietet die Cloud typischerweise mehr Leistung und ist ausfallsicherer. Daher werden geeignete Konzepte zur automatisierten Gruppierung und Platzierung von Komponenten und Daten benötigt, die zum einen die Struktur der jeweiligen Anwendung berücksichtigen und zum anderen die Einhaltung von definierten Anforderungen und Richtlinien gewährleisten. Im Rahmen dieser Arbeit wird dabei insbesondere die Optimierung des Datenflusses einer Anwendung durch die sinnvolle Platzierung von Komponenten und Daten betrachtet.

Forschungsbeitrag 3: *Entscheidungshilfe bei der Platzierung von Funktionen und Daten.*
Im dritten Forschungsbeitrag dieser Arbeit wird ein Konzept sowie Algorithmen zur automatisierten Gruppierung und Platzierung von Komponenten und Daten, mit dem Fokus auf die Optimierung des Datenflusses, vorgestellt und diskutiert. Als Grundlage hierfür dienen die Anwendungsstruktur, Informationen über den Datenfluss innerhalb der Anwendung, zum Beispiel die Menge an zu verarbeitenden Daten und deren jeweilige Lokalität, sowie zusätzlich definierte Anforderungen und Richtlinien, zum Beispiel bezüglich benötigter Leistung oder datenschutzrechtlichen Aspekten. Außerdem wird die Integration bestehender Konzepte und Algorithmen, beispielsweise zur Vervollständigung unvollständiger Anwendungstopologien oder zum Erreichen weiterer Optimierungsziele, vorgestellt und diskutiert.

1.1.4 Programmiermodell für die Implementierung verteilbarer und interagierender Komponenten

Durch Forschungsbeitrag 3 wird unter anderem das Verteilen der Anwendungskomponenten einer Anwendung auf Basis ihres Datenflusses ermöglicht. Um die eingeführten Konzepte und Algorithmen anwenden zu können, müssen die Anwendungen dementsprechend komponentenbasiert strukturiert sein. Damit die Komponenten im Anschluss an die Bereitstellung miteinander kommunizieren können, müssen diese zudem entsprechende Schnittstellen anbieten und konfiguriert werden, beispielsweise indem ihnen die Endpunktinformationen der anderen Komponenten, wie zum Beispiel Servicenamen, Adressen von Diensten oder Zugangsdaten, mitgeteilt werden. Die Implementierung der Komponenten und deren Schnittstellen kann unter anderem manuell oder durch bestehende Technologien und Frameworks unterstützt durchgeführt werden. Zur nötigen Konfiguration und Orchestrierung der einzelnen heterogenen Komponenten wird jedoch typischerweise individuell geschriebener Code verwendet, beispielsweise Konfigurations-

skripte, was jedoch zum einen die Portabilität der Anwendung einschränkt und zum anderen den Aufwand sowie die Komplexität der Implementierung erhöht. In diesem Forschungsbeitrag wird daher ein auf D^{3}M-basierendes Programmiermodell vorgestellt, das die modellgetriebene Entwicklung von verteilbaren und interagierenden Anwendungskomponenten unterstützt.

> ***Forschungsbeitrag 4:*** *Programmiermodell für die Implementierung verteilbarer und interagierender Komponenten.*
> Im vierten Forschungsbeitrag dieser Arbeit wird ein Programmiermodell sowie eine darauf aufbauende Entwicklungsmethode vorgestellt, die den Entwickler bei der Implementierung verteilbarer und interagierender Komponenten zur Realisierung von automatisch verteilbaren und bereitstellbaren Anwendungen unterstützen. Auf Basis der in einem D^{3}M definierten Informationen, zum Beispiel Identifikatoren und Schnittstellenbeschreibungen der Anwendungskomponenten, können hierfür verschiedene Code-Fragmente generiert werden, die sowohl den Endpunktaustausch als auch die Kommunikation zwischen den interagierenden Komponenten abstrahieren.

1.1.5 Shipping-Automatisierung, Sicherheit und Datenschutz

Abhängig von verschiedenen Faktoren, beispielsweise der Art des Unternehmens oder dem Land, in dem das Unternehmen Dienstleistung anbietet, müssen bestimmte Verordnungen und Gesetze beim Betrieb von Anwendungen sowie der Verarbeitung von Daten eingehalten werden. Unternehmen können zudem interne Richtlinien vorgeben, beispielsweise um den Einsatz von veralteter und unsicherer Software zu unterbinden und damit die eigene IT-Infrastruktur vor potenziellen Sicherheitsbedrohungen und dem Diebstahl von Betriebsgeheimnissen abzusichern. Bei einer automatisierten Gruppierung und Platzierung von heterogenen Anwendungskomponenten, so wie sie zum Beispiel in Forschungsbeitrag 3 vorgestellt wird, ist die manuelle Überprüfung aller geltenden Gesetze und Vorschriften allerdings zeitaufwendig

und fehleranfällig, insbesondere bei komplexen und heterogenen Anwendungen mit vielen Komponenten. Dementsprechend wird eine Möglichkeit zur einheitlichen Definition von wiederverwendbaren Anforderungen und Richtlinien sowie ein automatisierter Kontrollmechanismus hierfür benötigt.

> ***Forschungsbeitrag 5:*** *Shipping-Automatisierung, Sicherheit und Datenschutz.*
> Im fünften Forschungsbeitrag dieser Arbeit wird ein Konzept zur Modellierung von wiederverwendbaren Regeln, im Kontext von automatisiert bereitstellbaren Anwendungen vorgestellt. Dadurch wird die automatisierte Kontrolle und Einhaltung von vorgegebenen Anforderungen und Richtlinien bei der Verteilung und Bereitstellung einer Anwendung ermöglicht. Um die Erstellung der Regeln zu vereinfachen und zudem die Wiederverwendbarkeit der Regeln zu gewährleisten, können diese unabhängig von konkreten Anwendungstopologien definiert und bearbeitet werden. Zudem müssen keine vollständigen Anwendungsstacks, sondern nur die für eine Regel relevanten Informationen modelliert werden, beispielsweise bestimmte Eigenschaften einer Komponente oder unzulässige Komponententypen. Die Modellierung einer Anwendung sowie die Modellierung der einzuhaltenden Regeln eines Unternehmens können somit unabhängig voneinander und jeweils durch entsprechende Experten durchgeführt werden.

1.1.6 Architektur, Prototypen und Validierung der Beiträge

Zur Validierung der praktischen Umsetzbarkeit müssen die Konzepte der einzelnen Forschungsbeiträge prototypisch implementiert werden. Hierzu muss zunächst eine Architektur des Gesamtsystems zur Modellierung, Gruppierung und Platzierung von heterogenen Anwendungen und deren Komponenten und Daten erarbeitet werden. Darauf aufbauend werden die Prototypen zur Realisierung der Forschungsbeiträge, die zur Umsetzung der Shipping-Methode aus Forschungsbeitrag 1 erforderlich sind, implementiert.

Forschungsbeitrag 6: *Architektur, prototypische Implementierung und Validierung der Beiträge.*
Im sechsten Forschungsbeitrag dieser Arbeit wird eine Gesamtarchitektur vorgestellt, welche die einzelnen Komponenten zur Umsetzung der Forschungsbeiträge integriert. Zur praktischen Validierung der im Rahmen dieser Arbeit erarbeiteten Konzepte werden weiterhin die entwickelten prototypischen Implementierungen vorgestellt und diskutiert: (i) Zur Modellierung von datenverarbeitenden Anwendungen, (ii) zur Vereinfachung der Implementierung von interagierenden Anwendungskomponenten, (iii) zur Erstellung und Kontrolle von wiederverwendbaren Deployment-Regeln, (iv) zur Erstellung von eigenständigen Deployment-Paketen sowie (v) zur Gruppierung, Platzierung und Bereitstellung der Komponenten und Daten einer Anwendung. Mit diesen kann die in Forschungsbeitrag 1 eingeführte Shipping-Methode realisiert sowie größtenteils automatisiert werden. Außerdem wird die Wiederverwendbarkeit der vorgestellten Konzepte sowie Prototypen in anderen Arbeiten und Projekten diskutiert.

1.2 Veröffentlichungen im Rahmen der Arbeit

Die folgenden begutachteten Veröffentlichungen sind im Rahmen der Forschung dieses Promotionsvorhabens entstanden und decken insbesondere die zuvor beschriebenen Forschungsbeiträge dieser Dissertation ab. Die Publikationen sind dabei in chronologischer Reihenfolge aufgelistet.

1. [ZBF+17b] M. Zimmermann, U. Breitenbücher, M. Falkenthal, F. Leymann und K. Saatkamp. „Standards-based Function Shipping - How to use TOSCA for Shipping and Executing Data Analytics Software in Remote Manufacturing Environments“. In: *Proceedings of the 21st IEEE International Enterprise Distributed Object Computing Conference (EDOC 2017)*. IEEE, 2017, S. 50–60

2. [ZBL17] M. Zimmermann, U. Breitenbücher und F. Leymann. „A TOSCA-based Programming Model for Interacting Components of Automatically Deployed Cloud and IoT Applications“. In: *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017)*. SciTePress, 2017, S. 121–131

3. [ZBF+17a] M. Zimmermann, F. W. Baumann, M. Falkenthal, F. Leymann und U. Odefey. „Automating the Provisioning and Integration of Analytics Tools with Data Resources in Industrial Environments using OpenTOSCA“. In: *Proceedings of the 21st IEEE International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW 2017)*. IEEE, 2017, S. 3–7

4. [ZBL18] M. Zimmermann, U. Breitenbücher und F. Leymann. „A Method and Programming Model for Developing Interacting Cloud Applications Based on the TOSCA Standard“. In: *Enterprise Information Systems*. Springer, 2018, S. 265–290

5. [ZBKL18] M. Zimmermann, U. Breitenbücher, C. Krieger und F. Leymann. „Deployment Enforcement Rules for TOSCA-based Applications“. In: *Proceedings of The Twelfth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2018)*. Xpert Publishing Services, 2018, S. 114–121

6. [ZBG+18] M. Zimmermann, U. Breitenbücher, J. Guth, S. Hermann, F. Leymann und K. Saatkamp. „Towards Deployable Research Object Archives Based on TOSCA“. In: *Papers from the 12th Advanced Summer School on Service-Oriented Computing (SummerSoC 2018)*. IBM Research Division, 2018, S. 31–42

7. [ZBH+20] M. Zimmermann, U. Breitenbücher, L. Harzenetter, F. Leymann und V. Yussupov. „Self-Contained Service Deployment Packages“. In: *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*. SciTePress, 2020, S. 371–381

8. [ZBK+20] M. Zimmermann, U. Breitenbücher, K. Képes, F. Leymann und B. Weder. „Data Flow Dependent Component Placement of Data Processing Cloud Applications“. In: *Proceedings of the 2020 IEEE International Conference on Cloud Engineering (IC2E 2020)*. IEEE Computer Society, 2020, S. 83–94

Neben den zuvor ausgelisteten Veröffentlichungen haben zusätzlich die folgenden ebenfalls begutachteten Publikationen zu den in dieser Arbeit vorgestellten Forschungsergebnissen beigetragen.

1. [WBB+14b] J. Wettinger, T. Binz, U. Breitenbücher, O. Kopp, F. Leymann und M. Zimmermann. „Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA“. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. SciTePress, 2014, S. 559–568
2. [FBC+16] M. Falkenthal, U. Breitenbücher, M. Christ, C. Endres, A. W. Kempa-Liehr, F. Leymann und M. Zimmermann. „Towards Function and Data Shipping in Manufacturing Environments: How Cloud Technologies leverage the 4th Industrial Revolution“. In: *Proceedings of the 10th Advanced Summer School on Service Oriented Computing*. IBM Research Report, 2016, S. 16–25
3. [BEK+16] U. Breitenbücher, C. Endres, K. Képes, O. Kopp, F. Leymann, S. Wagner, J. Wettinger und M. Zimmermann. „The OpenTOSCA Ecosystem - Concepts & Tools“. In: *European Space project on Smart Systems, Big Data, Future Internet - Towards Serving the Grand Societal Challenges - Volume 1* (2016), S. 112–130
4. [FBK+16] M. Falkenthal, U. Breitenbücher, K. Képes, F. Leymann, M. Zimmermann, M. Christ, J. Neuffer, N. Braun und A. W. Kempa-Liehr. „OpenTOSCA for the 4th Industrial Revolution: Automating the Provisioning of Analytics Tools Based on Apache Flink“. In: *Proceedings of the 6th International Conference on the Internet of Things*. ACM, 2016, S. 179–180

5. [PBL+17] A. Panarello, U. Breitenbücher, F. Leymann, A. Puliafito und M. Zimmermann. „Automating the Deployment of Multi-Cloud Applications in Federated Cloud Environments“. In: *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2017)*. ACM, 2017, S. 1–8

6. [KBF+17] K. Képes, U. Breitenbücher, M. P. Fischer, F. Leymann und M. Zimmermann. „Policy-Aware Provisioning Plan Generation for TOSCA-based Applications“. In: *Proceedings of The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)*. Xpert Publishing Services, 2017, S. 142–149

7. [BOH+17] F. W. Baumann, U. Odefey, S. Hudert, M. Falkenthal und M. Zimmermann. „Cyber-physical System Control via Industrial Protocol OPC UA“. In: *Proceedings of the Eleventh International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2017)*. Xpert Publishing Services, 2017, S. 45–49

8. [FBG+17] M. Falkenthal, F. W. Baumann, G. Grünert, S. Hudert, F. Leymann und M. Zimmermann. „Requirements and Enforcement Points for Policies in Industrial Data Sharing Scenarios“. In: *Proceedings of the 11th Advanced Summer School on Service Oriented Computing*. IBM Research Division, 2017, S. 28–40

9. [SBK+18] K. Saatkamp, U. Breitenbücher, K. Képes, F. Leymann und M. Zimmermann. „OpenTOSCA Injector: Vertical and Horizontal Topology Model Injection“. In: *Proceedings of the Service-Oriented Computing – ICSOC 2017 Workshops*. Springer, 2018, S. 379–383

10. [YFK+18] V. Yussupov, M. Falkenthal, O. Kopp, F. Leymann und M. Zimmermann. „Secure Collaborative Development of Cloud Application Deployment Models“. In: *Proceedings of The Twelfth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2018)*. Hrsg. von G. Yee, S. Rass, S. Schauer und M. Latzenhofer. Xpert Publishing Services, 2018, S. 48–57

11. [OBG+18] U. Odefey, F. Baumann, G. Grünert, S. Hudert, M. Zimmermann, M. Falkenthal und F. Leymann. „Manufacturing Smart Services for automotive production lines“. In: *18. Internationales Stuttgarter Symposium*. Springer, 2018, S. 813–825

12. [WBK+20] B. Weder, U. Breitenbücher, K. Képes, F. Leymann und M. Zimmermann. „Deployable Self-Contained Workflow Models“. In: *Proceedings of the 8th European Conference on Service-Oriented and Cloud Computing (ESOCC 2020)*. Springer International Publishing, 2020, S. 85–96

13. [WBH+20] K. Wild, U. Breitenbücher, L. Harzenetter, F. Leymann, D. Vietz und M. Zimmermann. „TOSCA4QC: Two Modeling Styles for TOSCA to Automate the Deployment and Orchestration of Quantum Applications“. In: *Proceedings of the 24th International Enterprise Distributed Object Computing Conference (EDOC 2020)*. IEEE, 2020, S. 125–134

14. [KLZ20] K. Képes, F. Leymann und M. Zimmermann. „Situation-Aware Updates for Cyber-Physical Systems“. In: *Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020)*. Springer International Publishing, 2020, S. 12–32

15. [WBLZ21] B. Weder, J. Barzen, F. Leymann und M. Zimmermann. „Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective“. In: *Proceedings of the 18th IEEE International Conference on Web Services (ICWS 2021)*. IEEE, 2021, S. 1–13

16. [HILZ22] M. Hirsch, D. Iglezakis, F. Leymann und M. Zimmermann. „The ReSUS Project - Infrastructure for Sharing Research Software“. In: *E-Science-Tage 2021: Share Your Research Data*. heiBOOKS, 2022, S. 267–276

1.3 Aufbau der Arbeit

In Kapitel 2 werden zunächst die benötigten Grundlagen zum Verständnis dieser Arbeit sowie relevante verwandte Arbeiten vorgestellt und diskutiert. Im Anschluss daran wird in Kapitel 3 die Shipping-Methode zum automatisierten Platzieren und Shippen von Funktionen und Daten verteilter Anwendungen in heterogenen IT-Umgebungen vorgestellt. Da die nachfolgenden Forschungsbeiträge auf einer entsprechend modellierten Anwendungstopologie basieren, wird in Kapitel 4 zunächst das D^3M-Metamodell eingeführt und verschiedene Modellierungskonzepte vorgestellt. Weiterhin wird die Paketierung von eigenständigen Deployment-Paketen für D^3M-basierte Anwendungen vorgestellt um das Deployment von enthaltenen Anwendungskomponenten auch in IT-Umgebungen ohne Internetverbindung durchführen zu können. Kapitel 5 stellt anschließend die automatisierte Gruppierung und Platzierung der Anwendungskomponenten vor und zeigt wie die ebenfalls automatisierte Vervollständigung von Anwendungstopologien hierfür genutzt werden kann. Um eine Anwendung entsprechend verteilen zu können, muss diese komponentenbasiert konzipiert sein. Daher wird in Kapitel 6 ein D^3M-basiertes Programmiermodell vorgestellt, das den Entwickler bei der Implementierung verteilbarer und interagierender Komponenten unterstützt. Um das sichere und datenschutzkonforme Shippen von Anwendungskomponenten und Daten automatisiert zu ermöglichen, wird in Kapitel 7 ein Konzept zur Definition und Kontrolle von wiederverwendbaren Anforderungen und Richtlinien vorgestellt. Anschließend wird in Kapitel 8 die Architektur der prototypischen Implementierungen zur Validierung der einzelnen Forschungsbeiträge vorgestellt und diskutiert. In Kapitel 9 werden abschließend die Forschungsbeiträge dieser Arbeit zusammengefasst und einen Ausblick auf mögliche weiterführende Forschungsarbeiten gegeben.

Kapitel 2

Grundlagen und verwandte Arbeiten

In diesem Kapitel werden wichtige Grundlagen beschrieben, um die in der vorliegenden Arbeit vorgestellten Konzepte und Ansätze zu verstehen. Weiterhin werden verwandte Arbeiten diskutiert, um darzulegen, wie sie sich von dieser Arbeit unterscheiden und inwieweit sie als Grundlage für die vorgestellten Konzepte und Ansätze dienen. In Abschnitt 2.1 werden zunächst die Paradigmen Cloud Computing, Fog Computing, Edge Computing und Osmotic Computing vorgestellt und voneinander abgegrenzt. In Abschnitt 2.2 werden anschließend verschiedene Ansätze zum Shippen von Funktionen und Daten vorgestellt sowie allgemein Datenlokalität diskutiert. In Abschnitt 2.3 wird ein Überblick über verschiedene Ansätze zur Platzierung von Komponenten in verteilten Systemen gegeben. In Abschnitt 2.4 wird der aktuelle Stand der Wissenschaft und Technik zur Beschreibung von Anwendungstopologien und dem Deployment von Anwendungen betrachtet. Abschließend fasst Abschnitt 2.5 die zuvor vorgestellten Themen zusammen und diskutiert deren konkreten Bezug zur vorliegenden Arbeit.

2.1 Cloud Computing, IoT, Fog Computing, Edge Computing und Osmotic Computing

In diesem Abschnitt werden die verschiedenen Computing Paradigmen Cloud Computing, Fog Computing, Edge Computing, Osmotic Computing sowie Internet of Things (IoT) vorgestellt und voneinander abgegrenzt.

2.1.1 Cloud Computing

Cloud Computing ist in der heutigen IT-Landschaft von entscheidender Bedeutung für die Realisierung moderner IT-Systeme mit Fokus auf automatisierte Bereitstellung und Verwaltung [Ley09]. Eine Vielzahl von Anwendungsbereichen kann von diesen dadurch entstandenen Möglichkeiten profitieren, zum Beispiel Mobilität [GHH+17], Gesundheitswesen [HAR14], Energiemanagement [SM15] und Simulationswissenschaften [IOY+11]. Für den Begriff Cloud Computing gibt es eine Vielzahl an, sich meist nur im Detail leicht unterscheidenden, Definitionen und Beschreibungen [AFG+09; Ley09; MG11; VRCL08; WVY+10]. Eine in Publikationen häufig genutzte Definition für Cloud Computing stammt von dem National Institute of Standards and Technology (NIST) [MG11], einer US-amerikanischen Standardisierungsstelle. Diese Definition wird ebenfalls von der European Network and Information Security Agency (ENISA) [CH09], einer europäischen Agentur für Cybersicherheit, sowie dem Bundesamt für Sicherheit in der Informationstechnik (BSI) [BSI] verwendet.

Definition 2.1 (Cloud Computing)
Cloud Computing ist ein Modell, das erlaubt, jederzeit und von überall aus, bei Bedarf, über ein Netzwerk auf einen gemeinsam genutzten Pool von konfigurierbaren Rechnerressourcen (zum Beispiel Netzwerke, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die mit minimalem Verwaltungsaufwand oder Interaktion mit dem Dienstanbieter schnell bereitgestellt und freigegeben werden können [MG11]. ■

Die NIST-Definition von Cloud Computing nennt dabei fünf wesentliche Merkmale: (i) bedarfsgerechte Selbstbedienung (*on-demand self-service*), (ii) breiter Netzwerkzugang (*broad network access*), (iii) Ressourcenpooling (*resource pooling*), (iv) Elastizität der Ressourcen (*rapid elasticity*) und (v) Messbarkeit der Dienste (*measured service*).

- **Bedarfsgerechte Selbstbedienung.** Ein Nutzer kann Rechenkapazitäten, wie zum Beispiel Rechenzeit und Netzwerkspeicher, selbstständig und automatisch nach Bedarf bereitstellen, ohne dass eine menschliche Interaktion mit dem jeweiligen Dienstanbieter erforderlich ist.
- **Breiter Netzwerkzugang.** Die Funktionen sind per Netzwerkzugriff verfügbar und der Zugriff erfolgt über Standardmechanismen. Somit ist die Nutzung durch unterschiedlichste Clients, wie beispielsweise Mobiltelefone, Tablets, Laptops und PCs möglich.
- **Ressourcenpooling.** Die Ressourcen des Anbieters werden in einem Pool gesammelt, aus dem mehrere Nutzer bedient werden können (Multi-Tenant-Modell). Der Nutzer hat in der Regel keine Kontrolle oder Kenntnis über den genauen Standort der bereitgestellten Ressourcen, kann aber in der Lage sein kann, den Standort auf einer höheren Abstraktionsebene (z.B. Land oder Rechenzentrum) anzugeben.
- **Elastizität der Ressourcen.** Die Ressourcen können elastisch bereitgestellt und freigegeben werden. In einigen Fällen auch automatisch, um bei Bedarf schnell skalieren zu können. Für den Nutzer erscheinen die für die Bereitstellung verfügbaren Kapazitäten unbegrenzt und können in beliebiger Menge und zu jeder Zeit angefordert werden.
- **Messbarkeit der Dienste.** Cloud-Systeme können die Ressourcennutzung automatisch steuern und optimieren, indem sie eine Messfunktion auf einer für die Art des Dienstes geeigneten Abstraktionsebene nutzen (zum Beispiel Speichermenge, Rechenzeit, Bandbreite und aktive Benutzerkonten). Die Ressourcennutzung kann gemessen, kontrolliert und abgefragt werden, wodurch sowohl für den Anbieter als auch für den Nutzer des genutzten Dienstes Transparenz geschaffen wird.

Weiterhin werden in der NIST-Definition drei Service-Modelle für Cloud Computing eingeführt: (i) *Software as a Service (SaaS)*, (ii) *Platform as a Service (PaaS)* sowie (iii) *Infrastructure as a Service (IaaS)*. In Abbildung 2.1 werden diese drei Service-Modelle sowie zum Vergleich eine traditionelle On-Premise Lösung zur Veranschaulichung zusätzlich grafisch dargestellt.

- **SaaS.** Dem Nutzer werden Anwendungen zur Verfügung gestellt, die auf einer Cloud-Infrastruktur laufen. Der Zugriff auf die Anwendungen kann von verschiedenen Clients erfolgen, wie zum Beispiel über einen Webbrowser oder über eine Programmschnittstelle. Der Nutzer muss die zugrunde liegende Cloud-Infrastruktur, einschließlich Netzwerk, Server, Betriebssysteme, Speicher und auch einzelne Anwendungsfunktionen nicht selbst verwalten. Dies ist Aufgabe des Anbieters.
- **PaaS.** Dem Nutzer wird eine Plattform, bestehend aus diversen Programmiersprachen, Bibliotheken, Diensten und Tools, zur Erstellung von Anwendungen zur Verfügung gestellt. Der Nutzer muss die zugrunde liegende Cloud-Infrastruktur, einschließlich Netzwerk, Server, Betriebssysteme, Speicher nicht selbst verwalten. Allerdings hat er die Kontrolle über seine bereitgestellten Anwendungen.
- **IaaS.** Dem Nutzer wird eine Infrastruktur, bestehend aus Speicher-, Netzwerk- und Rechenressourcen zu Verfügung gestellt, auf denen der Nutzer beliebige Software, einschließlich Betriebssystemen und Anwendungen, einsetzen und ausführen kann. Der Nutzer verwaltet oder kontrolliert die zugrunde liegende Cloud-Infrastruktur nicht, hat aber die Kontrolle über Betriebssysteme, Speicher und bereitgestellte Anwendungen sowie möglicherweise eine begrenzte Kontrolle über bestimmte Netzwerkkomponenten, wie zum Beispiel einer Firewall.

Neben den drei ursprünglichen Service-Modellen werden in der Literatur weitere *aaS Service-Modellarten benannt. Dazu gehören beispielsweise *Network as a Service (NaaS)* [CMPW12], *Security as a Service (SecaaS)* [Get12] und *Database as a Service (DBaaS)* [CJP+11]. Als Sammelbegriff für einen über das Internet angebotenen Dienst wird typischerweise auch der Begriff *Everything as a Service (XaaS)* genutzt [BFB+11; DFZ+15].

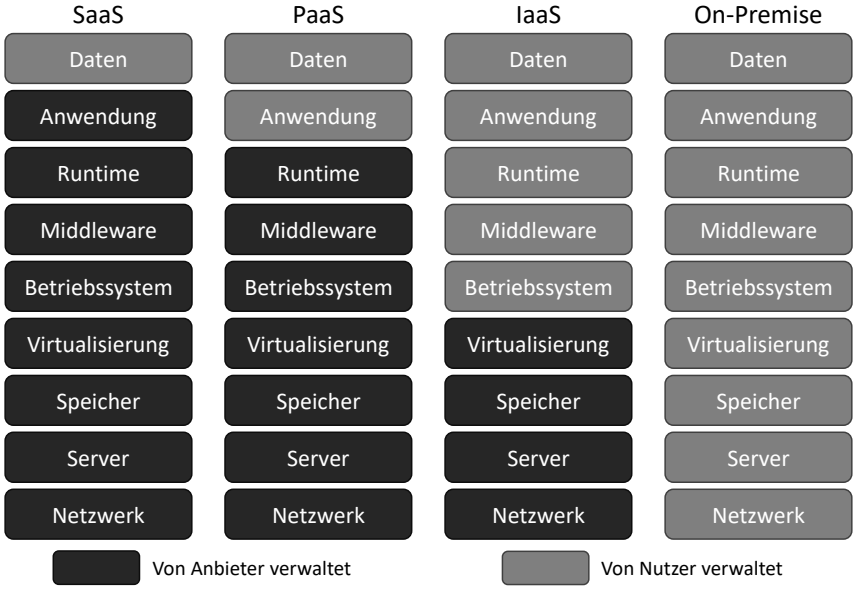


Abbildung 2.1: Vergleich der drei traditionellen Cloud Computing Service-Modellen und On-Premise Lösung

Schließlich werden in der NIST-Definition für Cloud Computing vier Bereitstellungsmodelle aufgelistet und spezifiziert:

- **Private Cloud.** Die Cloud-Infrastruktur wird für die ausschließliche Nutzung durch eine einzelne Organisation mit mehreren Verbrauchern (zum Beispiel Abteilungen) bereitgestellt. Sie kann von der Organisation selbst oder eines Dritten verwaltet und betrieben werden. Weiterhin kann sie entweder auf dem eigenen Gelände oder außerhalb stehen.
- **Community Cloud.** Die Cloud-Infrastruktur wird für die ausschließliche Nutzung durch eine bestimmte Gemeinschaft von Kunden aus Organisationen bereitgestellt, die gemeinsame oder ähnliche Anliegen haben. Sie kann im Besitz einer oder mehrerer dieser Organisationen oder eines Dritten sein und wird von diesen verwaltet und betrieben.

- **Öffentliche Cloud.** Die Cloud-Infrastruktur wird für die offene Nutzung durch die Allgemeinheit bereitgestellt. Sie kann sich im Besitz einer geschäftlichen, akademischen oder staatlichen Organisation befinden und wird von dieser verwaltet und betrieben. Die Infrastruktur befindet sich auf dem Gelände des Cloud-Anbieters.
- **Hybride Cloud.** Die Cloud-Infrastruktur ist eine Zusammensetzung aus zwei oder mehreren zuvor genannten Cloud-Infrastrukturen, die jedoch eigenständige Einheiten bleiben. Durch standardisierte oder proprietäre Technologien sind diese miteinander verbunden, wodurch die Portabilität von Daten und Anwendungen ermöglicht wird.

Die Vorteile des Cloud Computings liegen vor allem in der Möglichkeit der Kostenreduktion. Durch die gemeinsame Nutzung einer Infrastruktur, der nutzungsabhängigen Bezahlung der Dienste sowie einer dynamischen Skalierbarkeit der Ressourcen können einmalige Investitionen für IT-Ressourcen in laufende Betriebsausgaben umgewandelt werden. Weiterhin können die eigenen angebotenen Dienstleistungen einfach und dynamisch über mehrere Standorte global verteilt werden, da die Steuerung sowie der Zugriff auf genutzte Cloud-Dienste typischerweise per standardisierten und einheitlichen Schnittstellen erfolgt und dementsprechend automatisiert werden kann.

2.1.2 Internet of Things

Internet of Things (IoT) bezeichnet die Vernetzung verteilter und heterogener Geräte, die sowohl mit Menschen als auch mit anderen Geräten kommunizieren [XYWV12]. Ein solches Gerät ermöglicht dabei neben der Kommunikation, je nach Anwendung, auch weitere optionale Fähigkeiten, wie zum Beispiel Sensorik, Aktorik, Datenerfassung, Datenspeicherung und Datenverarbeitung [REC15]. Die Haupteigenschaften von IoT können wie folgt zusammengefasst werden [AIM17; REC15; XYWV12]:

- **Heterogenität.** Die Geräte im IoT können auf unterschiedlicher Hardware basieren. Außerdem können sie über verschiedene Netzwerke mit unterschiedlichen Service-Plattformen und Geräten interagieren.

- **Interkonnektivität.** Alle Geräte in der IoT können mit der globalen Kommunikations- und Informationsinfrastruktur verbunden werden. Dies ermöglicht die Interoperabilität der Geräte sowie ihre nahtlose Integration. Die Geräte sind dabei zudem eindeutig adressierbar.
- **Dynamische Änderungen und unkontrollierte Umgebung.** Im IoT können sich der Zustand der Geräte (zum Beispiel aktiv/inaktiv, verbunden/getrennt) und der Kontext (zum Beispiel Standort, Ausrichtung oder Geschwindigkeit) dynamisch ändern. Dementsprechend sind IoT-Geräte als Teil einer instabilen und unkontrollierten Umgebung zu sehen, in der die Interaktionen zwischen den Geräten aufgrund der ebenfalls möglichen instabilen Netzwerkkonnektivität und der dynamischen Änderungen des Gerätezustands unzuverlässig sind. Weiterhin kann sich auch die Anzahl der Geräte dynamisch ändern.
- **Beschränktheit der Ressourcen.** IoT umfasst in der Regel Geräte, die unter anderem in ihrer Speicherkapazität und Rechenleistung beschränkt sind.
- **Menge an Geräten.** Die Anzahl der zu verwaltenden Geräte, die miteinander kommunizieren, ist groß und wächst zunehmend. Weiterhin nimmt dementsprechend ebenfalls die Kommunikationslast zwischen den Geräten weiter zu. Daher wächst auch die Menge an Daten, die von solchen Geräten erzeugt wird, um Informationen miteinander zu teilen, stetig weiter.

Da sowohl die Hardware mit fortlaufender Weiterentwicklung kleiner als auch günstiger wird und zudem auch im Bereich der Anwendungen kontinuierlich neue Produkte entwickelt werden, nimmt die Anzahl an solchen IoT-Geräten zunehmend zu [Mic18]. Während es im Jahr 2018 noch etwa 18,4 Milliarden vernetzter Geräte weltweit gab, soll es bis Ende 2023 bereits 29,3 Milliarden vernetzte Geräte geben [Cis20]. Aufgrund der großen und noch weiter zunehmenden Anzahl an IoT-Geräten sowie der damit verbundenen Menge an erzeugten sowie zu versendenden Daten, ist IoT als eine der Hauptquellen von Big Data zu sehen [DX14]. Mittels Big Data Analytics

kann neues Wissen und hilfreiche Erkenntnisse erlangt werden, zum Beispiel zur Optimierung von Produktionsstraßen oder für Predictive Maintenance [Rus11]. Um die Analyse und Auswertung der hohen Datenvolumen zu ermöglichen, wird unter anderem eine hohe Rechenleistung benötigt. Daher werden hierfür Cloud Computing Technologien (siehe Abschnitt 2.1.1) genutzt. Der in Tabelle 2.1 dargestellte Vergleich zeigt, wie grundsätzlich unterschiedlich die Grundmerkmale von Cloud Computing und IoT sind und warum sich dadurch die beiden Paradigmen sehr gut gegenseitig ergänzen.

Unter anderem aufgrund der zuvor genannten Heterogenität von Geräten und Technologien, fehlen dem IoT im Vergleich zu Cloud Computing einige Eigenschaften wie zum Beispiel Verfügbarkeit, Zuverlässigkeit, Interoperabilität und Skalierbarkeit. Demgegenüber lassen sich durch IoT jedoch sensorische Daten, zum Beispiel von Maschinen, sowohl ermitteln, aggregieren, als auch zur weiteren Verarbeitung weiterleiten. Zusammen ermöglichen die beiden Paradigmen dadurch eine einfache und effektive Möglichkeit zur Datensammlung und Datenverarbeitung und daraus resultierend die Möglichkeit für komplexe Analysen und Vorhersagen. Beispiele hierfür sind unter anderem in den Bereichen Automotive [HYX14; KKW+17], Gesundheitswesen [DM12; GBAP12] und Smart Home [RT17; SAH+13] zu finden.

	Cloud Computing	**Internet of Things**
Big Data	Verwaltung & Verarbeitung	Datenquelle
Ressourcen/Komponenten	Virtuell	Physisch
Rechenleistung	Unbegrenzt*	Begrenzt
Speicherkapazität	Unbegrenzt*	Begrenzt
Erreichbarkeit	Durchgehend gut	Unterschiedlich
Standort	Zentralisiert*	Verteilt

* Aus Nutzersicht

Tabelle 2.1: Vergleich von Cloud Computing und IoT (nach [AAA+17])

Die starke Verzahnung von IoT und Cloud Computing sowie die stetige Zunahme an zu verarbeitenden Daten bei jedoch gleichzeitigem Bedarf nach

bestenfalls Echtzeitanalysen bringen allerdings auch einige neue Herausforderungen mit sich. Das Senden der Daten in die Cloud erfordert beispielsweise eine hohe Netzwerkbandbreite und ist teilweise auch aus Datenschutzgründen nicht erlaubt [SWW15; ZBKL18; ZMK+15]. Daher müssen hierfür die am Rand des Netzwerks erzeugten Daten lokal gespeichert und verarbeitet werden, ohne die Cloud miteinzubeziehen. Bei der Anforderung nach möglichst niedriger Latenz, zum Beispiel für industrielle Steuerungssysteme, ist die Nutzung der Cloud aufgrund der Entfernung und damit verbundenen Dauer der Bearbeitung ebenfalls zum Teil ungeeignet [ZMK+15]. Weiterhin sind IoT-Geräte typischerweise nur mit sehr begrenzten Ressourcen ausgestattet und teilweise selbst eingeschränkt um direkt und effizient mit der Cloud zu interagieren [BIS+18]. Um diese Herausforderungen und Probleme zu lösen benötigt es daher neue und darauf angepasste Paradigmen.

Fog Computing sowie Edge Computing sollen diese offene Lücke zwischen IoT und Cloud Computing schließen und die bestehenden Ansätze mit der Möglichkeit für eine effizientere Datenverarbeitung am Rande des Netzwerks erweitern. Die beiden Paradigmen Fog Computing und Edge Computing werden daher in den beiden folgenden Kapiteln vorgestellt.

2.1.3 Fog Computing

Die Idee und das Ziel hinter Fog Computing ist es, Rechen- und Speicherleistung sowie Verarbeitungs-Intelligenz dezentral am Rand der Cloud, möglichst in der Nähe der jeweiligen Datenquellen zur Verfügung zu stellen [DB16]. Hierzu werden Rechen- und Vermittlungsknoten, sogenannte Fog-Nodes, als Verbindungskomponenten zwischen den datengenerierenden IoT-Geräten sowie der Cloud genutzt. Diese sollen sowohl genug Leistung bieten, um bestimmte Daten bereits lokal zu verarbeiten, als auch die nötige Intelligenz und Funktionalität haben, um weitere Daten bei Bedarf an einen zentralen Cloud-Dienst zur weiteren Verarbeitung senden zu können [BMZA12]. Aufgrund der mittels der Fog-Nodes gebildeten Zwischenschicht im Netzwerk, muss nicht mehr zwingend die komplette zu verarbeitende

Datenmenge in die Cloud geschickt werden. Durch die dadurch verringerte Entfernung von der Datenquelle bis zu ihren datenverarbeitenden Knoten sowie die Reduzierung des zu übertragenen Datenvolumens in die Cloud, können sich die Kommunikationslatenzen und dementsprechend die Gesamtbearbeitungszeiten einer Anwendung verringern lassen, wodurch somit die Informationsverarbeitung in nahezu Echtzeit ermöglicht wird [BMZA12; ZBK+20]. Weiterhin können je nach Platzierung der Fog-Nodes auch mögliche Datenschutzprobleme gelöst werden [MMP+22; VR14]. Ein Vergleich der beiden Paradigmen aus Nutzersicht wird in Tabelle 2.2 gegeben.

	Cloud Computing	**Fog Computing**
Verteilung	Zentralisiert	Verteilt
Anzahl an Knoten	Wenige	Viele
Entfernung von Datenquelle	Eher groß	Eher gering
Latenz	Eher hoch	Eher niedrig
Leistungsfähigkeit	Sehr Hoch	Limitiert
Standortbewusstsein	Nein	Ja

Tabelle 2.2: Vergleich von Cloud und Fog Computing (nach [AWW18])

Das *OpenFog Consortium*[1], ein Zusammenschluss aus Industrie und Wissenschaft, wurde mit dem Ziel gegründet, die Einführung von Fog Computing zu fördern und zu beschleunigen. Im Rahmen ihrer Arbeit wurde eine Definition für Fog Computing veröffentlicht [OFC17], die im IEEE-Standard 1934-2018 [IEE18] wiederverwendet wird und auch für die vorliegende Arbeit verwendet wird:

Definition 2.2 (Fog Computing)
Fog Computing ist eine horizontale Architektur auf Systemebene, die Ressourcen und Dienste für Datenverarbeitung, Speicherung, Steuerung sowie Netzwerkfunktionen näher an den Benutzern und zwischen der Cloud und dem Internet of Things verteilt [IEE18; OFC17]. ■

[1] Das OpenFog Consortium fusionierte 2019 mit dem Industrial Internet Consortium: https://www.iiconsortium.org/press-room/01-31-19.htm

2.1.4 Edge Computing

Das Ziel von Edge Computing ist, wie auch bei Fog Computing, das Verlagern von Diensten und Rechenleistung weg von zentralisierten Cloud-Knoten hin zu dem Rande eines Netzwerks [SD16]. In einigen Publikationen werden Fog Computing und Edge Computing als Synonyme genutzt [GD18; HNYL17; VWB+16; YQL15]. Nach Shi et al. [SCZ+16] sowie Mukherjee et al. [MMS+17] konzentriert sich das Edge Computing allerdings mehr in Richtung der Geräte des IoT, während das Fog Computing eher als eine infrastrukturelle Zwischenschicht im Netzwerk zu sehen ist. Beim Edge Computing werden die anfallenden Daten möglichst direkt in der Netzwerkperipherie, wo die Daten generiert werden, verarbeitet, wodurch das zu versendende Datenvolumen weiter gesenkt sowie die Verarbeitungszeit weiter beschleunigt werden kann [SCZ+16]. Bei einem vollständig autonomen Fahrzeug werden zum Beispiel während einer 24-stündigen Testfahrt etwa 40 Terabyte an Daten produziert [BMW21]. Das Versenden, Verarbeiten in der Cloud und Rücksenden eines Ergebnisses bei dieser Menge an Daten würde zu lange dauern, um beispielsweise unmittelbar auf auftretende Hindernisse reagieren zu können. Dementsprechend bietet Edge Computing die Möglichkeit, Daten nah am Ort ihrer Entstehung und somit in Echtzeit zu verarbeiten, und ermöglicht es Geräten und intelligenten Anwendungen damit unmittelbar auf ein auftretendes Problem oder Ereignis zu reagieren. Dieser Ansatz, die Analyse genau dort durchzuführen, wo die Daten generiert werden, wird auch Edge Analytics genannt [Sat17; SSX+15; VWB+16].

Eine Übersicht über das Zusammenspiel und die Unterschiede der drei Computing Paradigmen Cloud Computing, Fog Computing und Edge Computing wird in Abbildung 2.2 gegeben. Zusammenfassend kann grundsätzlich festgehalten werden, dass je näher an den Datenquellen die Daten verarbeitet werden können, die Bandbreite für die Datenübertragung sowie die Reaktionsfähigkeit auf auftretende Ereignisse steigt. Wohingegen bei zunehmender Entfernung zwar auch die Speicherkapazitäten, Rechenleistungen und damit auch die potenzielle Intelligenz prinzipiell zunehmen, jedoch dabei die, für bestimmte Anwendungen benötigte, Reaktionsfähigkeit verloren geht.

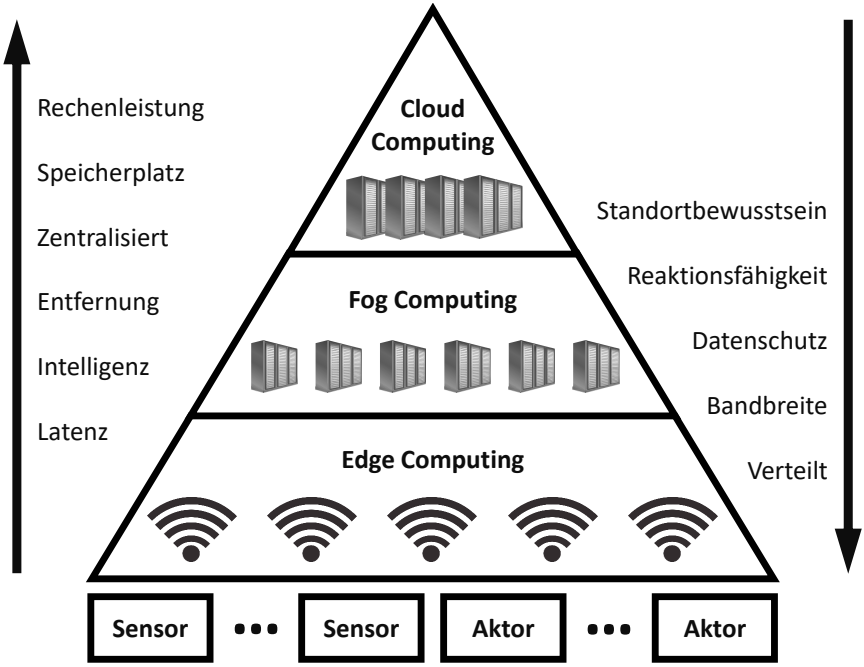


Abbildung 2.2: Zusammenspiel der verschiedenen Paradigmen: Cloud Computing, Fog Computing und Edge Computing

2.1.5 Osmotic Computing

Osmotic Computing ist das neuste der vorgestellten Paradigmen. Ziel ist der Aufbau einer dynamischen virtuellen Infrastruktur, welche die zuvor vorgestellten Paradigmen verbindet, um dabei die jeweiligen Vorteile, wie zum Beispiel hohe Rechenleistung und Speicherkapazität in der Cloud sowie die Möglichkeit von Echtzeitanalysen in der Edge, zu vereinen [VFD+16]. Begrifflich leitet sich Osmotic Computing von „Osmose“ ab, also dem Konzentrationsausgleich per Diffusion von Molekülen von einer höheren zu einer niedrigeren Konzentration [VCF18]. Übertragen auf IT-Infrastrukturen bedeutet das, die Last sowie die Ressourcennutzung ohne Redundanz, über Edge-, Fog- und Cloud-Infrastrukturen dynamisch auszugleichen.

Typischerweise werden für die Realisierung des Osmotic Computings Anwendungen mithilfe von containerbasierten Virtualisierungstechnologien, wie beispielsweise Docker[1] und Kubernetes[2], in einzelne Microservices [New15; STV18; Zim17] zerlegt, welche dann auf Basis von vorgegebenen Anforderungen dynamisch in den verschiedenen Umgebungen verteilt werden [VFD+16]. Anforderungen können sich dabei sowohl auf die Infrastruktur, zum Beispiel Zuverlässigkeit und Verfügbarkeit, als auch auf die Anwendung, zum Beispiel Kontextbewusstsein und Entfernung, beziehen. Weiterhin erlaubt Osmotic Computing auch das nachträgliche Anpassen dieser Anforderungen, was jedoch eine Umverteilung der Microservices zur Folge haben kann. Dementsprechend wird mit Osmotic Computing auch die dynamische Migration sowie die Verwaltung von Diensten über die verschiedenen Computing-Plattformen hinweg beschrieben [VFD+16].

Eine Herausforderung im Osmotic Computing ist unter anderem die hohe Heterogenität der unterschiedlichen beteiligten physischen Ressourcen, weshalb das Deployment der Microservices jeweils an die vorliegenden Bedingungen angepasst werden muss. Ebenfalls ist ein dynamisches sowie effizientes Migrationsmanagement nötig, um Ausfälle oder Verschlechterungen bezüglich der Dienstgüte (Quality of Service - QoS) bei auftretenden Netzwerkproblemen vermeiden und beheben zu können.

2.2 Shipping von Funktionen und Daten & Datenlokalität

Die immer weiter fortschreitende Entwicklung in Bereichen wie Internet of Things (IoT) [AIM10] und Cloud Computing [Ley09], sowie allgemein die verstärkte Nutzung von mit Sensoren ausgestatteten Geräten und Cyber-physischen Systemen (CPS) [GPGV14] führen fortlaufend zu einer deutlich erhöhten Menge an in der Cloud gespeicherten und zu verarbeitenden Daten [Mar18; SR20; VOE11]. Eine Vielzahl an Domänen kann von Möglichkeiten, wie zum Beispiel Big Data Analytics [Rus11] oder Kosteneinspa-

[1] https://www.docker.com/
[2] https://kubernetes.io/

rungen durch Pay-per-Use-Modelle profitieren. Beispielsweise seien hier die Bereiche Mobilität [GHH+17], Gesundheitswesen [HAR14], Energiemanagement [SM15] und Simulationswissenschaften [Sza11] genannt. Auch im Bereich der Industrie 4.0 [SGG+13] lassen sich neue Anwendungsfälle realisieren, indem beispielsweise die Verarbeitungslogik möglichst nah an den Datenquellen bereitgestellt und betrieben wird [FBC+16; ZBF+17b]. Um eine Datenanalyse zu ermöglichen, die neue Erkenntnisse und Optimierungspotenziale, beispielsweise in Fertigungsprozessen bietet, müssen hierzu Daten aus unterschiedlichen Quellen integriert, analysiert und verglichen werden. Diese Quellen können sich dabei unter anderem sowohl technologisch als auch hinsichtlich ihrer Lokalität unterscheiden.

Mit der zunehmenden Menge an gespeicherten und in der Cloud verteilten Daten wird das Netzwerk jedoch immer mehr zum Flaschenhals und der Transfer der Daten von einem Ort zum anderen wird zum Problem [AFG+10; CZ14; TAG22]. Datenverarbeitende Komponenten sowie die Daten selbst sind dementsprechend, je nach den spezifischen Anforderungen sowie der Lokalität der zu verarbeitenden Daten, oft über verschiedene Standorte und heterogene Umgebungen verteilt, zum Beispiel in privaten Clouds, öffentlichen Clouds und Edge Clouds [Sat17]. Zur Datenverarbeitung müssen jedoch die verarbeitende Funktionalität mit den entsprechenden Daten zusammengebracht werden. Dies kann durch die beiden Paradigmen *Function Shipping* und *Data Shipping* erreicht werden [FBC+16; FJK96].

Function Shipping wird angewendet, wenn die Funktionalität in der Nähe der zu verarbeitenden Daten ausgeliefert oder bereitgestellt werden soll. Data Shipping hingegen wird angewendet, wenn die Daten zur Funktionalität gesendet werden sollen, die die Daten verarbeitet. Gerade bei großen Datenmengen ist das Versenden der Funktionalität einfacher und schneller als das Versenden der Daten. Auch hohe Leistungsanforderungen können den Einsatz des Function Shippings erfordern. Der Datenschutz ist ein weiterer möglicher Grund für die Verwendung von Function Shipping, beispielsweise wenn Daten nur in einer bestimmten Umgebung verarbeitet werden dürfen und nicht über beliebige Lokalitäten verteilt werden dürfen [ZBKL18].

Demgegenüber gibt es jedoch auch Szenarien in denen das Data Shipping genutzt wird. Im Bereich der automatischen Produktion wird dies zum Beispiel verwendet, wenn viele Daten aus verschiedenen Quellen, zum Beispiel von einzelnen Maschinen, zuerst gesammelt werden müssen und die lokale IT-Infrastruktur in der Nähe der Daten nicht über genügend Rechenleistung zur Verarbeitung der Daten verfügt. In diesem Fall müssen die Daten zunächst in eine leistungsfähigere Ausführungsumgebung übertragen werden, zum Beispiel in eine Public Cloud. Wann also welches Shipping Paradigma am besten geeignet ist, hängt vom konkreten Szenario ab.

Gemein haben beide Ansätze, dass damit Datenlokalität erreicht werden kann. Die Grundidee der Datenlokalität besteht darin, die Daten verarbeitenden Funktionen sowie die zu verarbeitenden Daten so nah wie möglich beieinander zu haben, zum Beispiel auf demselben physischen Knoten [GFZ12; WL91]. Das grundsätzliche Ziel ist es, die Leistungsfähigkeit dadurch zu erhöhen, dass Daten und die verarbeitenden Funktionen nahe beisammen sind und somit der Transport der Daten möglichst keinen oder nur einen geringen Einfluss auf die Leistung hat. Da sowohl Datenlokalität als auch die Idee hinter Function Shipping und Data Shipping keine neuen Ansätze sind, wird im Folgenden ein Überblick über Bereiche gegeben, in denen diese Konzepte bereits erforscht oder angewendet wurden.

Im Bereich von Mikroprozessoren und Mikrocomputer wird zum Beispiel an der Zusammenlegung von Verarbeitungs- und Speicher-Komponenten auf einem einzigen Chip geforscht, um die Gesamtdatenbewegung zu reduzieren und somit die Gesamtleistung zu erhöhen [ESS+99; MGGA19; PAC+97]. Bei diesem sogenannte *Near-Data-Processing (NDP)* [BCM+14; GAK15] handelt es sich um eine hardwareabhängige Lösung und nicht um einen allgemeinen Ansatz zur Ermöglichung von Function Shipping aus Sicht der Software

Im Bereich von Datenbanken gibt es zum Beispiel das Konzept der *Stored Procedures* [Dat83]. Diese Prozeduren werden auf dem Server gespeichert und können somit mithilfe eines einfachen Befehls in direkter Nähe der Daten ausgeführt werden. Eigene entwickelte Stored Procedures können auch weitergegeben werden, um sie in Datenbanken zu verwenden, die

diesen Ansatz unterstützen. Cornell et al. [CDP86] hat die Leistung von gekoppelten Multisystem-Datenbankensystemen mit sogenanntem *function request shipping*-Ansatz sowie *data sharing*-Ansatz untersucht. Beim ersten Ansatz wird die Datenbank auf die Multisysteme aufgeteilt und nur die Funktionsanforderungen werden zwischen ihnen verschickt. Bei letzterem Ansatz besteht die Idee darin, die Daten zwischen den Multisystemen zu teilen, indem eine gemeinsame Datenbank auf Festplattenebene verwendet wird. Das Hauptziel beider Ansätze ist es, die unnötige Übertragung einer großen Datenmenge zwischen den einzelnen Systemen zu vermeiden.

Mit MOCHA (**M**iddleware Based **O**n a Code S**H**ipping **A**rchitecture) [RR00] haben Rodríguez-Martínez und Roussopoulos ein erweiterbares Datenbank-Middleware-System vorgestellt, das für die Verbindung von über das Netzwerk verteilten Datenquellen konzipiert ist. MOCHA ermöglicht den automatischen Versand von in Java implementierter Funktionalität an die Datenstelle, an der die Funktionalität ausgeführt werden soll. Das Ziel von MOCHA ist es, die Leistung dadurch zu verbessern, indem die Menge der zu übertragenden Daten reduziert wird. Daher werden datenreduzierende Abfragen an den Ort verlagert, an dem sich die Daten befinden, während bei datenvergrößernden Abfragen die zu verarbeitenden Daten zur Abfrage verlagert werden. Der Ansatz ermöglicht jedoch nur den Versand von in Java implementierten Funktionen und ist dabei weiterhin auf die Funktionen beschränkt, die in ihrem Katalog, einem zentralen Code-Repository, gespeichert sind. Beliebige Software-Komponente können dementsprechend damit nicht ausgeliefert und bereitgestellt werden.

In Bereich von Hochleistungsrechner ist der Versand der Daten oft die bevorzugte Option, da in solchen Systemen, die typischerweise CPU-intensive Berechnungen durchführen, der Aufwand für die Übertragung der Daten zu den Recheneinheiten im Vergleich zur Rechenzeit eher gering ist [ABB+01]. Im Kontext von Big Data, IoT, Cloud und Edge Computing wäre jedoch dieser Ansatz, große Datenmengen zu den verarbeitenden Funktionen zu übertragen, in der Regel deutlich langsamer, als die verarbeitenden Funktionen möglichst nahe an die zu verarbeitenden Daten zu bringen [AFG+10].

Darüber hinaus sind im Bereich der wissenschaftlichen Workflows, insbesondere wenn diese sehr datenintensiv sind, sowohl die konkrete Platzierung als auch die Verteilung in Abhängigkeit von der Größe der zu verarbeitenden Daten wichtige Forschungsfragen [DC08; DPA+18].

Im Bereich der verteilten Datenverarbeitung machen sich auch Cluster-Computing-Frameworks wie Apache Hadoop [TASFb] oder Apache Storm [TASFa] das Prinzip des Function Shipping zunutze. In Apache Hadoop werden beispielsweise die Funktionen, die MapReduce [DG08]-Funktionalität implementieren, an andere Knoten innerhalb des Clusters verschickt. Bei Apache Storm wird die Topologie, die die Verarbeitungslogik beschreibt, vom Master-Knoten analysiert und anschließend die Aufgaben auf Worker-Knoten verteilt, wo sie ausgeführt werden. MapReduce [DG08] selbst stellt dabei ein Programmiermodell vor, das die Datenlokalitätsprobleme adressiert, um die Leistung von datenintensiven Anwendungen im Bereich Cloud Computing und Big Data zu verbessern [GFZ12]. MapReduce vereinfacht die Entwicklung und Ausführung großer Datenverarbeitungsjobs, indem es die Aufgaben zerlegt und auf verteilte Dateisysteme verteilt. Nach der Bearbeitung werden die Ergebnisse anschließend wieder gesammelt sowie integriert. Verteilte Dateisysteme, wie zum Beispiel Google File System [GGL03] und Hadoop Distributed File System [SKRC10], können dadurch die Verarbeitung von großen Datenmengen bewältigen, indem sie die Daten auf mehreren Knoten verteilt speichern, verarbeiten und replizieren.

In verteilten System können *Remote Procedure Calls (RPC)* [BN84] verwendet werden, um eine Funktion in einem anderen Adressraum, zum Beispiel auf einem anderen Server, aufzurufen und auszuführen. Hierfür gibt es verschiedene Implementierungen wie beispielsweise Java RMI [Ora10] sowie Spezifikationen wie CORBA [OMG12], die eine objektorientierte Middleware beschreibt. Allerdings wird dabei nur der Aufruf einer Operation und gegebenenfalls noch für den Aufruf benötigte Daten oder Parameter übertragen, nicht aber die Funktion an sich. Mobile Agenten [PK98] sind ein weiteres verwandtes Paradigma auf dem Gebiet des verteilten Rechnens. Sie ermöglichen die autonome Migration von Software von einem Knoten zu

einem anderen Knoten innerhalb eines Netzwerks und reduzieren so unter anderem den Bandbreitenverbrauch innerhalb des Netzwerks.

Während in der Big-Data-Analytik die Datenverarbeitung typischerweise eher zentralisiert erfolgt, zum Beispiel in einem Big-Data-Warehouse, wird in der Edge-Analytik [SSX+15] die Datenverarbeitung in der Nähe des aktiven Geräts oder sogar direkt auf dem Gerät selbst durchgeführt. Dadurch wird eine hohe Datenlokalität erreicht. Gerade im Kontext von Internet of Things mit vielen angeschlossenen und verbundenen Geräten wird dieser Ansatz oft genutzt, um den Aufwand und vor allem die benötigte Zeit für das Senden der Gerätedaten in die Cloud oder einen On-Premise-Server zu reduzieren.

Im Bereich des Cloud Computings gibt es das Konzept der *Serverless Architectures* [AWS15; Rob16]. Die Idee hinter diesem Ansatz ist, dass der Entwickler einer Funktion das System oder die Laufzeitumgebung, auf der die Funktion ausgeführt wird, nicht einrichten muss. Stattdessen ist der Cloud-Anbieter dafür verantwortlich, eine Laufzeitumgebung bereitzustellen, die alle erforderlichen Fähigkeiten für die Ausführung der Funktion gewährleistet, und diese Ressourcen auch zu verwalten. Der Nutzer, zum Beispiel der Entwickler einer Funktion, muss diese zum Betrieb nur in die entsprechende Laufzeitumgebung schieben. Allerdings können bei diesem Konzept die Funktionen nur an einen Cloud-Anbieter ausgeliefert werden, der einen solchen Ansatz unterstützt. Um Funktionen nahe an Daten zu liefern, zum Beispiel in eine Fertigungsumgebung, ist ein Anbieter-unabhängiger Ansatz erforderlich.

Alle diese unterschiedlichen Ansätze haben gemeinsam, dass nur die Funktionalität selbst und nicht ein ganzer Software-Stack in die Zielumgebung ausgeliefert werden kann. Typischerweise besteht eine komplexe Anwendung jedoch aus mehreren Komponenten. Weiterhin sind diese Konzepte abhängig von einer bestimmten Zielumgebung, zum Beispiel einer bestimmten Datenbank, einer bestimmten Middleware oder bestimmten Cloud-Anbietern, in die die Funktion ausgeliefert und dort betrieben werden kann.

2.3 Platzierung von Komponenten in verteilten Systemen

Die Verlagerung von Arbeitslasten zwischen der Cloud und Fog sowie Edge durch die entsprechende Platzierungen der Komponenten und Dienste wurde unter anderem in den Studien von Kumar und Lu [KL10] (*cloud offloading*), Mach und Becvar [MB17] (*edge offloading*) sowie Villari et al. [VCF18] (*osmotic computing*) erforscht. Mit der teilweisen Verlagerung von Rechenlasten von der Cloud hin zu Fog und Edge entstehen jedoch neue Probleme bei der konkreten Verteilung der einzelnen Komponenten, Dienste sowie Daten einer Anwendung. So ermöglichen das Fog und Edge Computing prinzipiell zwar eine höhere Zeiteffizienz und weniger Netzwerkverkehr, sind aber typischerweise auch instabiler und weniger performant als die Cloud [AGT20; KMH+21]. Dementsprechend werden geeignete Platzierungs-Konzepte benötigt, um eine jeweils sinnvolle Verteilung für die entsprechenden Anwendungen zu gewährleisten [Man22; MSKV19; SDL20].

In der Literatur lassen sich viele verschiedene Ansätze, Studien sowie konkrete Algorithmen bezüglich der Platzierung von Komponenten oder Operatoren sowie hinsichtlich der Übertragung von Daten in verteilten Systemen finden [AAA21; BFGL19a; HM21; HYZ22; LT19; SM22]. Sie beruhen weitestgehend jedoch auf sehr unterschiedlichen Optimierungszielen und Annahmen und sind daher schwer miteinander vergleichbar. In diesem Abschnitt werden daher nachfolgend verschiedene Ansätze zur Platzierung von Komponenten sowie Daten in verteilten Systemen vorgestellt und voneinander abgegrenzt.

Die existierenden Platzierungskonzepte zielen auf die Optimierung verschiedener Ziele ab, zum Beispiel dem Ressourcenverbrauch [GGQ+13; LQLW13; ZLL+16], dem Energieverbrauch [AAR22; CD14; FZ15; WTTL12], der Verfügbarkeit sowie Zuverlässigkeit [BBB+11; MMM10; ZWC+17], den verursachenden Kosten [AMA+18; CCLS11; VTM09] sowie der zu erwartenden Antwortzeit [AL13; ARJE22; KYT14; PY10]. Weiterhin gibt es dabei sowohl Ansätze, die nur auf eins dieser Ziele abzielen [BBB+11; KYT14], als auch Ansätze, zur Optimierung mehrerer dieser Ziele [GGQ+13; VTM09]. Auch hinsichtlich der genutzten Methoden zur Platzierungsentscheidung können

die vorhandenen Arbeiten voneinander abgegrenzt werden. So beruhen die Arbeiten unter anderem auf graphentheoretischen Ansätze [FSB13; LDK09; ZA08], Greedy-Algorithmen [AL06; GÖÖ16; LTX15], mathematischen Optimierungsansätzen [ADP17; CGLN16; RDR10], sowie verschiedenen Arten von heuristischen Ansätzen [CV14; MPZ10; SMN17]. Im Folgenden werden einige dieser Forschungsarbeiten zur Gruppierung und Platzierung von Komponenten sowie dem Fokus auf die Übertragung von Daten näher vorgestellt.

Eine Forschungsarbeit mit speziellem Fokus auf Fog Computing gibt es mit SpanEdge [SDAV16]. Sajjad et al. stellen dabei einen Ansatz vor, der es Programmierern ermöglicht, die Teile ihrer Anwendung zu spezifizieren, die später nahe an den Datenquellen platziert werden sollen. Das Ziel ist es, den Bandbreitenverbrauch und die Antwortlatenz zu reduzieren. Hierzu erweiterten sie für ihre prototypische Implementierung Apache Storm [TAS-Fa], ein Framework zur Verarbeitung von Datenströmen. Um die Kosten für die Bereitstellung von Ressourcen zu optimieren, schlagen Arkian et al. mit MIST [ADP17] ebenfalls ein Konzept für die Platzierung von IoT-Anwendungen auf Fog-Ressourcen vor. Hierfür formulieren sie zunächst ein nichtlineares gemischt-ganzzahliges Optimierungsproblem (MINLP), das anschließend in ein lineares gemischt-ganzzahliges Optimierungsproblem (MILP) linearisiert wird. Goethals et al. [GDV20] schlagen für die Optimierung der Platzierung von Fog-Diensten eine Strategie namens Swirly vor. Dafür wird der Einfluss von allen angeschlossenen Geräten und Netzwerkverbindungen in der Computing-Umgebung analysiert und ein Verhaltensmodell konstruiert. Basierend auf diesem Verhaltensmodell sowie vorher definierbaren Schwellenwerten werden die Services dann platziert.

Sahni et al. [SCY19] schlagen einen mehrstufigen Greedy-Anpassungsalgorithmus vor, um datengesteuertes Task-Scheduling in einer ressourcenbeschränkten Computing-Umgebung, wie der Edge-Cloud, hinsichtlich der Reduzierung der Latenzzeit zu optimieren. Der vorgestellte Algorithmus berücksichtigt dabei sowohl die Platzierung der Dienste als auch die Netzwerkflüsse, mit dem Ziel Engpässe und Überlastungen bei der Datenübertragung zu vermeiden. Bourhim et al. [BED19] präsentieren eine Me-

thode zur Platzierung von Container-basierten Diensten in der Fog-Cloud, die die Kommunikation zwischen den einzelnen Containern berücksichtigt. Ihr Ziel ist es, die Kommunikationslatenzen zwischen den eingesetzten Containern zu optimieren. Hierfür schlagen sie einen genetischen Algorithmus vor. Auch Brogi et al. [BFGL19b] nutzen einen genetischen Algorithmen in ihrer Arbeit. Sie schlagen eine Kombination aus genetischen Algorithmen und Monte-Carlo-Simulationen vor, um die Verteilung von Komponenten effizienter durchführen zu können. Insbesondere werden dabei Faktoren wie Kosten sowie bestimmte Quality of Service Eigenschaften beachtet.

Einige der existierenden Ansätze, zum Beispiel die Vorschläge von Lakshmanan et al. [LLS10] und Gedik et al. [GÖÖ16], sind speziell für Cluster-Umgebungen konzipiert. Dabei wird für die Übertragung von Daten allerdings eine Netzwerklatenz von nahezu Null angenommen. Dementsprechend sind diese Art von Ansätzen nicht für Fälle geeignet, in denen die Netzwerklatenz einen großen Einfluss auf die Leistung und Ausführungszeit einer Anwendung hat und daher nicht ignoriert werden kann. Zum Beispiel, wenn Komponenten von datenintensiven Anwendungen über verschiedene Standorte und heterogene Umgebungen verteilt werden müssen.

Andere verwandte Arbeiten [ABQ13; EL16; FSB13; XCTS14], die auch die Eigenschaften innerhalb eines Netzwerkes berücksichtigen, versuchen vor allem die Menge der zwischen den einzelnen Rechenknoten ausgetauschten Daten zu minimieren. Während Fischer et al. [FSB13] dafür ein Graphpartitionierungsverfahren verwenden, schlagen Eidenbenz et al. [EL16] einen heuristischen Algorithmus vor, der auch die Transferkosten der Daten berücksichtigt. Weiterhin wird in den Arbeiten von Aniello et al. [ABQ13] und Xu et al. [XCTS14] eine Greedy Heuristik vorgeschlagen, um den Transfer von Daten zwischen den Knoten zu minimieren. Im Ansatz von Aniello et al. besteht das Hauptziel darin, die Operatoren auf der Grundlage des Kommunikationsaufwands zwischen ihnen, möglichst auf einem einzigen Knoten zusammenzufassen. Bei Xu et al. dagegen, werden die Standorte der Operatoren in absteigender Reihenfolge des eingehenden sowie ausgehenden Datenverkehrs zugewiesen. Die Arbeit von Ferdaus et al. [FMCB17] hat das

Optimierungsziel, die Entfernungen zwischen verarbeitenden Komponenten innerhalb eines Datenzentrums zu minimieren. Dadurch sollen Daten nicht über weite Distanzen versendet werden müssen. Besondere Priorität haben dabei die Komponenten, die ihre Daten direkt von Datenbanken beziehen. Zur Umsetzung wird hier ebenfalls eine Greedy Heuristik verwendet.

Im Bereich von Sensornetzwerken verwenden Abrams und Liu [AL06] Platzierungsheuristiken sowie baumstrukturierte Anwendungsgraphen, um die Platzierung dahingehend zu optimieren, die Menge der benötigten Bandbreite zu reduzieren. Gu et al. [GZG+16] untersuchten, wie die Kommunikationskosten für datenstromverarbeitende Anwendungen in geografisch verteilten Rechenzentren minimiert werden können. Dazu untersuchten sie die Kostenunterschiede zwischen Rechenzentren bei der Platzierung virtueller Maschinen. In Bezug auf geografisch verteilte Systeme zur Verarbeitung von Datenströmen berücksichtigen Zhu et al. [ZA08] explizit Kommunikationsverzögerungen bei ihrer Platzierung. In ihrem vorgeschlagenen Ansatz gehen sie jedoch davon aus, dass ein Knoten höchstens einen einzigen Operator hosten kann, was eine sehr stark begrenzende Annahme darstellt.

Oppenheimer et al. [OCP+06] untersuchten die Platzierung und Migration von Diensten in föderierten Netzwerken über weite physikalische Entfernungen. Diese Arbeit berücksichtigte dabei sowohl die verfügbaren Ressourcen, die Netzwerkbedingungen und die Kosten für die Migration von Diensten zwischen Standorten in Bezug auf Ressourcen und Latenzzeiten.

Weitere Forschungsarbeiten mit dem Fokus auf die Optimierung der Verteilung von Cloud-Anwendungen, wie zum Beispiel die MOCCA-Methode von Leymann et al. [LFM+11], das Optimal Distribution Framework von Andrikopoulos et al. [AGLW14], Kingfisher von Sharma et al. [SSSS11], CloudMIG von Frey und Hasselbring [FH11a], CloudGenius von Menzel und Ranjan [MR12] sowie weitere zahlreiche Arbeiten [ÁCJ+16; ASL13; BCS19; KGSS12; MGAD13; ZZZB12] optimieren die Gruppierung und Platzierung der Komponenten typischerweise auf Basis der Angebote konkreter Cloud Service Provider und berücksichtigen dabei hauptsächlich Faktoren wie entstehende Kosten oder Quality of Service Eigenschaften wie beispielsweise

Verfügbarkeit. Relevante Faktoren hinsichtlich der Verarbeitung von Daten, um unter anderem die Verteilung und damit die Performance einer Anwendung zu verbessern, wie zum Beispiel der konkrete Datenfluss innerhalb einer Anwendung, werden in diesen Arbeiten jedoch nicht betrachtet.

Neben der Platzierung und Gruppierung von Anwendungskomponenten, beschäftigen sich einige Forschungsarbeiten auch insbesondere mit der Platzierung von Datenquellen, wie beispielsweise Datenbanken oder datenerzeugende Dienste. Hierbei werden die Platzierungen der datenverarbeitenden Komponenten typischerweise als bereits bekannt vorausgesetzt. Naas et al. [NLBR18; NPBL17] schlagen beispielsweise eine Teile-und-herrsche-Heuristik vor, um Daten möglichst sinnvoll in Fog-Infrastrukturen zu platzieren. Hierzu zerlegen sie das Platzierungsproblem in mehrere kleine Teilprobleme und betrachten zunächst nur bestimmte Ausschnitte der gesamten Infrastruktur. Weiterhin unterschieden sie zwischen Datenkonsumenten, die bereits in der Infrastruktur vorhanden sind, sowie Datenproduzenten, welche platziert werden müssen. Das Optimierungsziel ihrer Arbeit ist dabei, eine möglichst geringe Latenz zu erhalten. Hierzu werden Techniken der ganzzahligen linearen Optimierung genutzt.

Die Arbeit von Lera et al. [LGJ18] versucht, die entstehenden Daten so nah wie möglich an den IoT-Geräten zu speichern, wo sie erzeugt sowie benötigt werden. Hierzu wird innerhalb des Netzwerks der einflussreichste Knoten gesucht. Der einflussreichste Knoten wird dabei aus verschiedenen Werten, wie beispielsweise der Anzahl an Verbindungen zu anderen Knoten sowie der Entfernung zu diesen Knoten berechnet. Das Ziel ist es auch hier, dadurch die Netzwerkauslastung zu minimieren.

Die in diesem Abschnitt vorgestellten Arbeiten stellen verschiedene Ansätze und Algorithmen zur Optimierung der Platzierung im Hinblick auf verschiedene Optimierungsziele vor, ohne dabei jedoch die Struktur der Anwendung, ihren Datenfluss und die Bereitstellung der Anwendung selbst hinsichtlich der Platzierung ihrer Komponenten ganzheitlich zu betrachten. Da weiterhin viele der Forschungsarbeiten in diesem Bereich ausschließlich auf theoretischen Simulationsstudien basieren, besteht weiterhin das Problem einer

praktischen Lösung, die die Modellierung, Platzierung, Bereitstellung sowie das Management beliebiger Cloud- und IoT-Anwendungen unterstützt. Auch werden in einigen der vorgestellten Arbeiten zuvor zu erhebende Daten benötigt um die Konzepte anwenden zu können. So werden beispielsweise historische Daten zum Trainieren von Platzierungs-Algorithmen benötigt oder eine sehr große Anzahl an vorab zu berechnenden Parametern erwartet, um eine optimale Lösung berechnen zu können.

Was insgesamt fehlt, ist ein modellbasierter Ansatz, der nicht nur die Leistung von datenverarbeitenden Cloud- und IoT-Anwendungen mit Hinblick auf ihren Datenfluss optimiert, sondern den gesamten Lebenszyklus einer Anwendung berücksichtigt und den Entwickler somit von der Modellierung der Anwendung, über das automatisierte Deployment der einzelnen Komponenten, bis hin zum Management der Anwendung unterstützt.

2.4 Anwendungstopologien, Deployment und Orchestrierung

Der Aufbau moderner in der Cloud betriebener Anwendungssysteme, typischerweise bestehend aus vielen verteilten und heterogenen Diensten und Anwendungskomponenten, führt zu einer zunehmenden Komplexität, da die einzelnen Dienste und Komponenten zur Inbetriebnahme der Gesamtanwendung zuerst installiert, konfiguriert sowie miteinander integriert werden müssen [BKH05; GHS10; JSW20; ZBL18]. Bei solchen komplexen Systemen ist die manuelle Ausführung dieser Schritte jedoch (i) ineffizient, (ii) zeit- und kostenintensiv sowie (iii) fehleranfällig [BBK+13a; EKK+06; Opp03].

Dementsprechend besteht hierfür der Bedarf nach modellbasierten und automatisierten Lösungen [BBKL14; BK06; Ley09; ZBF+17b]. In diesem Abschnitt werden daher zunächst zur Beschreibung von Anwendungen verwendete Anwendungstopologien vorgestellt und relevante Begriffe erklärt. Anschließend werden verschiedene Ansätze zur Automatisierung des Deployments vorgestellt und diskutiert sowie weiterhin Möglichkeiten zur Orchestrierung von verteilten Anwendungen erläutert.

Anwendungstopologien sind typischerweise graphbasierte Modelle und beschreiben den strukturellen Aufbau einer Anwendung [BBKL14]. Grundsätzlich bestehen sie aus Knoten und gerichteten Kanten, wobei die Knoten die einzelnen Infrastruktur- und Softwarekomponenten der Anwendung repräsentieren und die Kanten die Relationen zwischen zwei Komponenten repräsentieren [AGLW14]. Beispielsweise kann somit die Abhängigkeit einer Komponente zu einer anderen Komponenten modelliert werden. Weiterhin wird damit auch die Beschreibung der Modularisierung [Gra86], also die Aufteilung der Gesamtanwendung in einzelne Module ermöglicht. Durch die Typisierung der Knoten und Kanten sowie die damit ermöglichte Definition der Semantik der Knoten und Kanten mittels diesen Typen wird eine einfache Wiederverwendbarkeit dieser Elemente ermöglicht.

Neben der grafischen Visualisierung einer Anwendung ermöglichen Anwendungstopologien im Vergleich zu klassischen Architekturmodellen, welche vor allem zur Dokumentation und Kommunikation genutzt werden, jedoch auch das automatisierte Deployment und Verwaltung der modellierten Anwendung [INS+14; LFM+11]. Hierfür können Anwendungstopologien neben den Knoten und Kanten weitere technische Informationen bezüglich den einzelnen Komponenten, der Relationen oder der Gesamtanwendung beinhalten. Beispielsweise Eigenschaften zur genaueren Spezifikation einer Komponente, wie beispielsweise Leistungsangaben bezüglich der Menge an verfügbarem Arbeitsspeicher oder die Anzahl an Rechenkernen einer virtuellen Maschine. Weiterhin können mögliche Operationen für das Management einer Komponente definiert werden, zum Beispiel zum Installieren, Konfigurieren sowie Starten einer Komponente. Um das Deployment technisch zu realisieren, können weiterhin ausführbare Artefakte, welche die zuvor genannten Operationen implementieren, an die einzelnen Knoten und Kanten einer Anwendungstopologie gehangen werden. Ebenfalls können ausführbare Artefakte, welche die Businesslogik der Komponenten beinhalten, den jeweiligen Knoten zugeordnet werden. Im Rahmen dieser Arbeit ist eine Anwendungstopologie, angelehnt an Binz [Bin15], wie folgt definiert:

Definition 2.3 (Anwendungstopologie)
Eine Anwendungstopologie beschreibt den strukturellen Aufbau sowie die technischen Details zur automatisierten Bereitstellung und Verwaltung einer Anwendung mittels eines formalen und maschinenlesbaren Modells. Weiterhin besteht eine Anwendungstopologie auf höchster Ebene aus Knoten, welche die Komponenten der Anwendung definieren, sowie Kanten, welche die Relationen zwischen den Komponenten definieren. Diese Elemente können wiederum weitere Informationen, beispielsweise deren Konfiguration, sowie ausführbare Artefakte beinhalten. ■

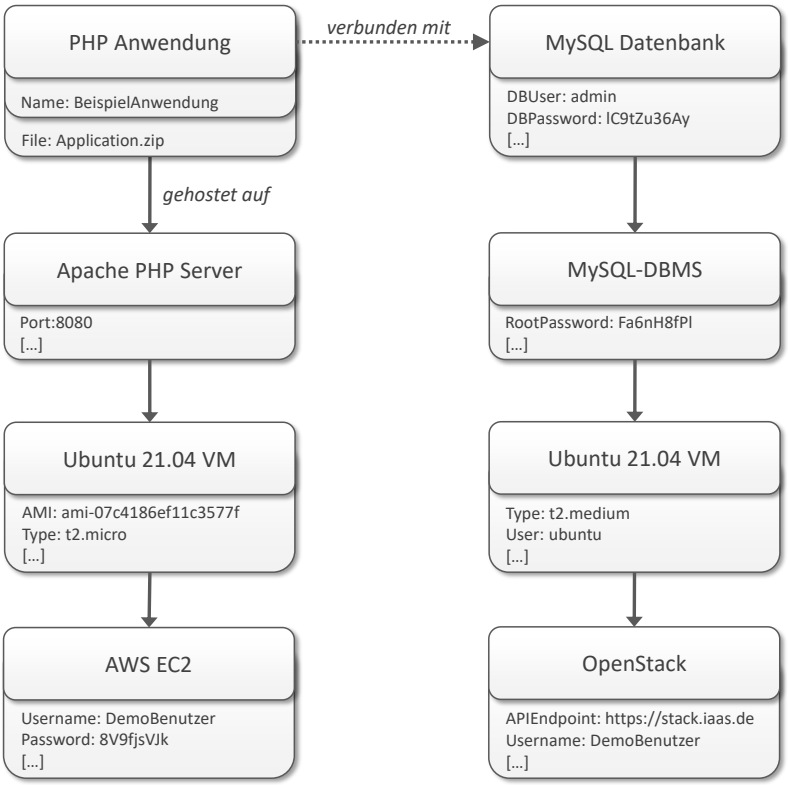


Abbildung 2.3: Beispiel einer Anwendungstopologie

In Abbildung 2.3 wird eine beispielhafte Anwendungstopologie, bestehend aus einer PHP-basierten Anwendung und einer MySQL-Datenbank sowie jeweils benötigter Middleware- und Infrastruktur-Komponenten, auf Basis der grafischen Notation *Vino4TOSCA*[1] [BBK+12] dargestellt. Neben den einzelnen Komponenten der Anwendung sind weiterhin die Relationen zwischen diesen Komponenten dargestellt. Also zum Beispiel, dass eine Komponente auf einer anderen Komponente gehostet wird oder dass eine Komponente mit einer anderen Komponente eine Verbindung aufbaut. Ebenfalls sind weitere technische Informationen über die einzelnen Komponenten, welche für das Deployment benötigt werden, in der Anwendungstopologie enthalten. Dazu gehören zum Beispiel die Benutzerdaten, die zur Erstellung einer virtuellen Maschine mithilfe von AWS EC2[2] benötigt werden, der Port, über den eine Anwendung erreichbar sein soll, sowie zum Deployment benötigte Artefakte, wie hier beispielsweise die angehängte Datei *Application.zip*, welches die Implementierung der PHP-Anwendung beinhaltet. Neben der Darstellung dieses Beispiels wird Vino4TOSCA weiterhin auch für die Darstellung von Anwendungstopologien in der gesamten Arbeit genutzt.

In der Literatur lassen sich viele unterschiedliche Modellierungssprachen zur Modellierung von Cloud-Anwendungen finden [BBF+18; BWKG14], wovon einige im Folgenden kurz vorgestellt werden. Mit *Blueprint Templates* stellen Nguyen et al. [NLPH12; NLT+11] beispielsweise ein Konzept zur technologieagnostischen Beschreibung von Service-basierten Anwendungen vor. Der Fokus des Ansatzes liegt bei der Beschreibung der Verteilung der Anwendung vor allem auf den verschiedenen Cloud Computing Service-Modellen SaaS, PaaS und IaaS, sowie einer möglichst von allen technischen Details abstrahierenden Darstellung. *GENTL*[3] von Andrikopoulos et al. [ARSL14; ARXL14] ist eine Modellierungssprache mit dem Ziel die Bereitstellungskonfiguration von Cloud-Anwendungen zu optimieren. Hierzu werden Annotationen unterstützt, um zum Beispiel durch Preisangabe der Cloud-Anbieter Kostenkalkulationen durchzuführen oder Informationen bezüglich dem Management

[1] Visual Notation for TOSCA
[2] `https://aws.amazon.com/de/ec2`
[3] Generalized Topology Language

von Komponenten zu definieren. GENTL unterstützt weiterhin die bidirektionale Transformation von Blueprints. Der Ansatz *CloudML*[1] von Ferry et al. [FCS+18; FRC+13] ermöglicht sowohl die Cloud-Anbieter-unabhängige Modellierung der Anwendungsbereitstellung, als auch eine Cloud-Anbieter-spezifische Modellierung der Anwendungsbereitstellung. Es können dabei sowohl die Anwendungskomponenten, sowie auch deren Beziehungen zueinander, zum Beispiel Abhängigkeiten oder Kommunikationskanäle zwischen zwei Komponenten, modelliert werden.

StratusML[2] von Hamdaqa et al. [HLT11; HT15] stellen auch eine Möglichkeit zur technologieagnostischen Modellierung einer Anwendung zur Verfügung. Ihr Ansatz bietet dabei zusätzlich fünf Ansichten, um verschiedene Eigenschaften und Anforderungen bezüglich der Anwendung, zum Beispiel hinsichtlich den entstehenden Kosten, abbilden zu können. Außerdem wird die Generierung von Anbieter-spezifischen Artefakten unterstützt. *CadaML*[3] von Jumagaliyev und Elkhatib [JE19a; JE19b] ermöglicht die Beschreibung von mandantenfähigen Cloud-Anwendungen mit besonderem Fokus auf die jeweilige Datenarchitektur. Der vorgestellte Ansatz ermöglicht weiterhin die Generierung von Code für den Datenzugriff auf unterschiedliche Arten von Cloud-basierten Datenspeichern. Der *MOCCA*[4] Ansatz von Leymann et al. [LFM+11] beinhaltet eine Modellierungssprache zur Beschreibung der Migration von bestehenden Anwendungen in eine Cloud-Umgebung. Dabei werden unterschiedliche Modellansichten, wie zum Beispiel eine Architekturansicht, eine Deploymentansicht sowie eine Ansicht mit den einzelnen implementierten Artefakten der Anwendung unterstützt.

VAMP[5] von Etchevers et al. [ECB+11; ECBP11] ermöglicht ebenfalls die automatisierte Provisionierung von verteilten Legacy-Anwendungen in der Cloud. Der Fokus liebt dabei vor allem auf der Beschreibung der virtuellen

[1] Cloud Modelling Language
[2] Stratus Modeling Language
[3] Modeling Language for the Design and Implementation of Cloud Application Data Architectures
[4] Move to Clouds for Composite Applications
[5] Virtual Applications Management Platform

Maschinen, auf denen die Komponenten betrieben werden sollen, sowie benötigte Kommunikationsverbindungen zwischen ihnen. Im Ansatz *PDS*[1] von Lu et al. [LSS+13] werden sogenannte *System Pattern* zur Beschreibung der Anwendung genutzt. Dabei wird die technische Sicht der Anwendung, unter anderem bestehend aus den Komponenten und deren Beziehungen auf Cloud-Ressourcen abstrahiert. Davon ausgehend können sogenannte Deployment Pläne zur Bereitstellungsautomatisierung erzeugt werden. Ein Deployment Plan definiert dabei die einzelnen Schritte, sowie deren Reihenfolge, welche zur Bereitstellung der Anwendung ausgeführt werden müssen.

Weiterhin gibt es einige UML-basierte Modellierungssprachen, welche den UML Standard [OMG07] unter anderem mittels eigenen UML-Profilen erweitern. Guillén et al. stellen mit *MULTICLAPP*[2] [GMMC13a; GMMC13b] einen solchen UML-basieren Ansatz vor, welcher die Modellierung von Multi-Cloud-Anwendungen in einer von konkreten Cloud-Anbietern unabhängigen Form ermöglicht. Bei Bedarf kann dann von einem solchen Modell ausgehend der benötigte Code zur Nutzung konkreter Cloud-Dienste generiert werden. Im Gegensatz dazu, ermöglicht *Model4Cloud* von Bernal et al. [BCV+19] die direkte Modellierung von Cloud-Anwendungen, einschließlich der zugrundeliegenden Infrastruktur, der definierten Benutzer-Anforderungen hinsichtlich der Leistung und benötigter Ressourcen sowie notwendigen Interaktionen mit dem jeweiligen Cloud-Anbieter.

Ähnlich dazu ermöglicht *CAML*[3] von Bergmayr et al. [BBK+16; BTN+14] die UML-basierte Beschreibung von Cloud-Anwendungen mit speziellem Fokus auf technische sowie nichttechnische Anforderungen. Als technische Anforderungen werden dabei beispielsweise Anforderungen hinsichtlich benötigter Ressourcen gesehen, wohingegen nichttechnische Anforderungen zum Beispiel durch die Preise der Cloud-Anbieter dargestellt werden. *CAMEL*[4] [AKR+19; KSMM19] ist eine weitere auf UML basierende Modellierungssprache zur Modellierung von Multi-Cloud-Anwendungen. Insbesonde-

[1] Pattern-based Deployment Service
[2] Multicloud Migratable and Interoperable Applications
[3] Cloud Application Modeling Language
[4] Cloud Application Modelling and Execution Language

re ermöglicht CAMEL nicht nur die Verwaltung von Modellen zur Designzeit, sondern auch die Verwaltung von Modellinstanzen zur Laufzeit.

Darüber hinaus gibt es einige Modellierungssprachen, bei denen sich der Hauptfokus ausschließlich auf die Beschreibung der Infrastruktur sowie Cloud-Ressourcen richtet und keine ganzheitliche Modellierung der Anwendungsstruktur ermöglichen. Silva et al. schlagen zum Beispiel mit *Cloud DSL* [SRC14] ein Vokabular zur Beschreibung von Cloud-Plattformen, Services und Ressourcen vor. Ihre Konzentration liegt dabei jedoch vor allem auf IaaS-Diensten. Das Vokabular soll unter anderem zur erleichterten Kommunikation mit allen Beteiligten beitragen und mittels Codegenerierung die Entwicklung der Anwendungen erleichtern. Der Ansatz *CloudMIG* von Frey und Hasselbring [FH10; FH11b] soll die Migration von bestehenden Anwendungen in die Cloud erleichtern. Dabei liegt der Fokus bei der technischen Umsetzung der Migration vor allem auf der Verwendung von PaaS- und IaaS-Diensten. Ihr vorgestellter Ansatz ermöglicht weiterhin die Beachtung von definierten Nutzer-Anforderungen, zum Beispiel hinsichtlich benötigter Ressourcen oder verursachenden Kosten der Cloud-Dienste.

RESERVOIR [CEM+10; RBL+09] ist ein weiterer Ansatz zur Beschreibung von vor allem (virtuellen) Infrastrukturen. Mittels zu definierenden Regeln soll dabei zudem unter anderem die Skalierbarkeit einer Anwendung zur Laufzeit über mehrere Cloud-Anbieter hinweg ermöglicht werden. Der Schwerpunkt von *CloudML*[1] von Gonçalves et al. [GES+11] liegt ebenfalls auf der reinen Beschreibung der Infrastrukturebene einer Anwendung. In ihrem graphbasierten Modell nutzen sie Knoten und Kanten zur Darstellung von Rechen- und Netzwerk-Ressourcen sowie deren Verbindungen, welche weitere Angaben zum Beispiel bezüglich vorhandenem Arbeitsspeicher, Speicherplatz oder Rechenleistung beinhalten können.

Weiterhin existiert mit der *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [BBKL14; BBLS12; OAS13a; OAS13b] ein OASIS-Standard, zur portablen sowie interoperablen Modellierung von Cloud-Anwendungen. Dabei wird sowohl die Modellierung von Softwarekompo-

[1] Cloud Modeling Language

nenten sowie Infrastrukturkomponenten, als auch die Modellierung der Beziehungen zwischen diesen Komponenten ermöglicht. Außerdem können mittels sogenannten Managementplänen explizit die auszuführenden Operationen zur automatisierten Bereitstellung einer solchen modellierten Anwendung beschrieben werden. Weiterhin nutzen auch einige der zuvor vorgestellten Modellierungsansätze, zum Beispiel CAML, CloudDSL sowie GENTL, TOSCA entweder intern direkt als Modellierungssprache oder ermöglichen eine Transformation, um die automatisierte Bereitstellung von Cloud-Anwendungen mittels TOSCA zu ermöglichen.

Da mittels TOSCA modellierte Anwendungstopologien den strukturellen Aufbau einer Anwendung beschreiben, also wie die Anwendung nach erfolgreichem Deployment aufgebaut sein soll, handelt es sich dabei um einen deklarativen Modellierungsansatz, wobei die Bereitstellungslogik zur automatisierten Bereitstellung der Anwendung typischerweise zur Laufzeit vom Modell der Anwendung abgeleitet wird [BBK+14]. Aufgrund der vom Standard definierten Managementplänen unterstützt TOSCA jedoch auch zusätzlich die imperative Modellierung des Deployments einer Anwendung [EBF+17]. Der deklarative Ansatz wird in der Industrie sowie Wissenschaft jedoch als bevorzugter Ansatz für das automatisierte Deployment und Management von Anwendungen angesehen [BBF+18; HAW11].

Konzepten wie Konfigurationsmanagement [DJV10] oder Infrastructure as Code [Mor16] folgend, existieren neben den vor allem anfänglich verwendeten, komplexen und fehleranfälligen Perl, Python und Shell Skripten [Bur95; WBL14a], auch eine Vielzahl an verschiedenen in der Industrie genutzten deklarativen Deployment sowie Orchestrierungs Technologien [BPR16; EGHS16; WBF+19]. Von diesen werden im Folgenden einige der bekanntesten und am meisten genutzten Technologien hinsichtlich ihrer Verwendungsmöglichkeiten eingeordnet, gruppiert und diskutiert.

Nach Wurster et al. [WBF+19] lassen sich die Technologien in die folgenden drei Kategorien einordnen: (i) allgemein verwendbare Deployment Technologien, (ii) Anbieter-spezifische Deployment Technologien sowie (iii) Plattform-spezifische Deployment Technologien. Allgemein verwendbare

Deployment Technologien ermöglichen das Deployment von Single-, Hybrid- und Multi-Cloud-Anwendungen. Weiterhin unterstützen sie die Verwendung von XaaS Diensten. Zu dieser Gruppe gehören die Technologien *Ansible* [RH], *Chef* [Che], *Cloudify* [CP], *Juju* [Can], *OpenStack HEAT* [OSF], *Puppet* [Pup], *SaltStack* [VMW] und *Terraform* [HC].

Anbieter-spezifische Deployment Technologien unterstützen ebenfalls XaaS Dienste. Da die Anbieter-spezifischen Deployment Technologien von einem bestimmten Cloud-Anbieter angeboten werden und dementsprechend nur dessen Cloud-Dienste unterstützt, sind sie im Gegensatz zu den allgemein verwendbaren Deployment Technologien, jedoch nur für Single-Cloud Implementierungen geeignet. Beispiele hierfür sind unter anderem *AWS CloudFormation* [AWS] sowie *Azure Resource Manager* [Mic].

Plattform-spezifische Deployment Technologien unterstützen im Gegensatz zu den Anbieter-spezifischen Deployment Technologien mehrere Cloud-Provider. Allerdings sind sie im Vergleich zu den allgemein verwendbaren Deployment Technologien nicht auf die Verwendung für beliebige XaaS Dienste ausgelegt, sondern sind an bestimmte Plattformen gebunden. Zu dieser Kategorie gehören unter anderem *CFEngine* [NTAS], *Docker Compose* [Doc] sowie *Kubernetes* [CNCF]. Wurster et al. haben weiterhin in einer systematischen Studie die wesentlichen Elemente und Funktionalitäten dieser deklarativen Deployment-Automatisierungstechnologien abgeleitet und sie auf das *Essential Deployment Metamodel (EDMM)* abgebildet [WBF+19].

Im Rahmen dieser Arbeit wird TOSCA als Basis der Modellierungssprache für Anwendungstopologien genutzt, da TOSCA (i) sowohl Anbieter-agnostisch als auch Technologie-agnostisch ist [BSW+14], (ii) ontologisch erweiterbar ist [BBF+18] sowie (iii) kompatibel mit dem Essential Deployment Metamodel ist [WBB+19]. Zur technischen Umsetzung, der in dieser Arbeit dargestellten Forschungsbeiträge, können jedoch auch andere Modellierungssprachen mit vergleichbaren Konzepten hinsichtlich der Beschreibung von Anwendungstopologien verwendet werden.

2.5 Zusammenfassung und Diskussion

In diesem Kapitel wurden zunächst die Eigenschaften der verschiedenen Computing-Paradigmen wie Cloud Computing, Fog Computing, Edge Computing sowie Internet of Things vorgestellt. Dabei hat sich gezeigt, dass je nach Anwendungsfall verschiedene Anforderungen einer Anwendung bezüglich ihrer Zielumgebung berücksichtigt werden müssen. Rechen- und datenintensive Anwendungen benötigen beispielsweise typischerweise die Rechenleistung und den Speicherplatz einer Cloud-Umgebung. Für zeitkritische Anwendungen dagegen, zum Beispiel im Bereich Industrie 4.0, ist häufig vor allem eine geringe Latenz wichtig. Daher müssen in solchen Fällen die datenverarbeitenden Komponenten möglichst nahe an den jeweiligen Datenquellen, beispielsweise Maschinen in einer Fabrik, platziert werden.

Insbesondere in komplexen verteilten Systemen sowie Anwendungsfällen, in denen die Verteilung der Komponenten und Daten nicht direkt ersichtlich ist, können Konzepte angewendet werden, um die Komponenten einer Anwendung sinnvoll platzieren und verteilen zu können, um sie mit den zu verarbeitenden Daten zusammenzubringen. Diesbezüglich wurden in diesem Kapitel verschiedene Ansätze sowie Konzepte vorgestellt, die hierzu sowohl unterschiedliche Techniken nutzen als auch unterschiedliche Optimierungsziele ermöglichen. Weiterhin wurden verschiedene Ansätze und Technologien zur Automatisierung des Deployments einer Anwendung vorgestellt sowie weiterhin Möglichkeiten zur Orchestrierung von verteilten Anwendungen erläutert und diskutiert.

Zusammenfassend zeigen die vorherigen Abschnitte, dass ein gesamtheitlicher Ansatz fehlt, der (i) die Gruppierung und Platzierung von Anwendungskomponenten unter Berücksichtigung des Datenflusses einer Anwendung optimiert, (ii) dabei die Definition verschiedener Anforderungen ermöglicht, die bei der Optimierung berücksichtigt werden sollen, (iii) die Verwendung bewährter deklarativer Modellierungskonzepte erlaubt sowie (iv) den Nutzer über den gesamten Lebenszyklus einer Anwendung hinweg, unter anderem durch Automatisierungsmöglichkeiten unterstützt.

Kapitel 3

Methode zum Shippen von Funktionen und Daten

In diesem Kapitel wird die Shipping-Methode für das automatisierte Platzieren und Shippen von Funktionen und Daten verteilter Anwendungen in heterogenen IT-Umgebungen vorgestellt. Die Methode bildet den ersten Forschungsbeitrag der vorliegenden Arbeit. Einzelne Schritte der Methode werden durch die weiteren Forschungsbeiträge dieser Arbeit umgesetzt.

Im Vergleich zu den in Abschnitt 2.2 und Abschnitt 2.3 vorgestellten Ansätzen, ermöglicht die Shipping-Methode unter anderem das technologieunabhängige sowie anbieterunabhängige Shipping ganzer Softwarestacks, und nicht nur einzelner Funktionen, mit dem Fokus auf einer Datenfluss optimierenden Platzierung der einzelnen Komponenten einer Anwendung. Aufgrund des modellbasierten Ansatzes wird dabei weiterhin der gesamte Lebenszyklus einer Anwendung, also von der Modellierung der Anwendung, über die Implementierung der einzelnen Komponenten, das automatisierte Deployment sowie das Management der Anwendung ermöglicht.

Das Kapitel ist im weiteren Verlauf wie folgt gegliedert: Im folgenden Abschnitt 3.1 werden Anforderungen und Herausforderungen an die Methode definiert. Anschließend wird in Abschnitt 3.2 die Shipping-Methode im Detail präsentiert sowie in Abschnitt 3.3 mögliche Varianten der Methode vorgestellt. In Abschnitt 3.4 werden die Automatisierungsmöglichkeiten der einzelnen Schritte der Methode, welche in den folgenden Kapiteln der Arbeit noch genauer vorgestellt werden, diskutiert. Abschließend fasst Abschnitt 3.5 dieses Kapitel nochmals zusammen, stellt die Umsetzung der erhobenen Anforderungen vor und diskutiert die Shipping-Methode im Kontext der in Kapitel 2 vorgestellten Grundlagen und verwandten Arbeiten.

3.1 Anforderungen und Herausforderungen

In diesem Abschnitt werden die, sich vor allem aus den vorherigen Kapiteln abgeleiteten, Anforderungen und Herausforderungen an die Methode zum Shippen von Funktionen und Daten benannt und erläutert. Im weiteren Verlauf des Kapitels wird die jeweilige Umsetzung dieser aufgezeigt.

Anforderung 1 - Automatisierung: Sowohl das manuelle Deployment einer Anwendung, als auch die konkrete Gruppierung und Platzierung der einzelnen Komponenten, ist fehleranfällig und zeitaufwändig. Daher ist die Automatisierung dieser Schritte eine wichtige Anforderung an die Methode.

Anforderung 2 - Deklarative Modellierung: Die deklarative Modellierung gilt als bevorzugter Modellierungsansatz für das automatisierte Deployment und Management von Anwendungen. Weiterhin wird bei der deklarativen Modellierung, aufgrund der Darstellung der Anwendungsstruktur, eine einfachere Identifikation von Fehlern oder Problemen bei der Verteilung von Komponenten ermöglicht. Dementsprechend soll zur Umsetzung ein deklaratives Modell als Basis der gesamten Methode genutzt werden.

Anforderung 3 - Beibehaltung der Funktionalität: Die konkrete Platzierung der einzelnen Komponenten einer Anwendung darf keine Auswirkungen auf die fachliche Funktionalität der Anwendung haben.

Anforderung 4 - Technologie-Agnostik: Es existiert eine Vielzahl unterschiedlichster in der Praxis genutzter Technologien zum Deployment und Management von Anwendungen. Daher soll die Verwendung der Methode unabhängig der Nutzung einer bestimmten Technologie ermöglicht werden.

Anforderung 5 - Anbieter-Agnostik: Neben Technologie-agnostisch, soll die Methode auch Anbieter-agnostisch sein. Dementsprechend soll die Platzierung von Komponenten unabhängig von bestimmten Anbietern und auf beliebigen Plattformen ermöglicht werden.

Anforderung 6 - Manuelle Überprüfbarkeit und Beeinflussbarkeit der Platzierung: Um mögliche Fehler bei der Platzierung von Komponenten, die zum Beispiel aufgrund einer fehlerhaften initialen Anwendungsmodellierung entstanden sind, vor dem Deployment der Anwendung entdecken und nachvollziehen zu können, muss dem Nutzer eine Kontrollmöglichkeit sowie eine manuelle Einflussnahme auf die Platzierung ermöglicht werden.

Anforderung 7 - Wiederverwendbarkeit von Expertenwissen: Beim Betrieb einer verteilten Anwendung, deren Komponenten in unterschiedlichen Lokalitäten platziert sind, müssen gegebenenfalls bestimmte Anforderungen, zum Beispiel bezüglich dem Datenschutz, beachtet werden. Diese Anforderungen sind typischerweise sehr komplex und erfordern spezielles Expertenwissen. Dementsprechend sollte während der Durchführung der Methode die Einbeziehung von (bestehendem) Expertenwissen ermöglicht werden. Neben der Compliance kann es sich dabei aber auch um Expertenwissen bezüglich Best Practices, zum Beispiel im Hinblick auf die Modellierung des Deployments und Managements von Anwendungen, handeln.

Anforderung 8 - Erweiterbarkeit und Änderbarkeit: Um nötige Erweiterungen und Änderungen, zum Beispiel bezüglich den Kriterien für die Platzierung der Anwendungskomponenten, umsetzen zu können, muss die Methode selbst sowie ihre technische Realisierung entsprechend erweiterbar gestaltet sein, beispielsweise durch die Verwendung eines Plugin-Systems.

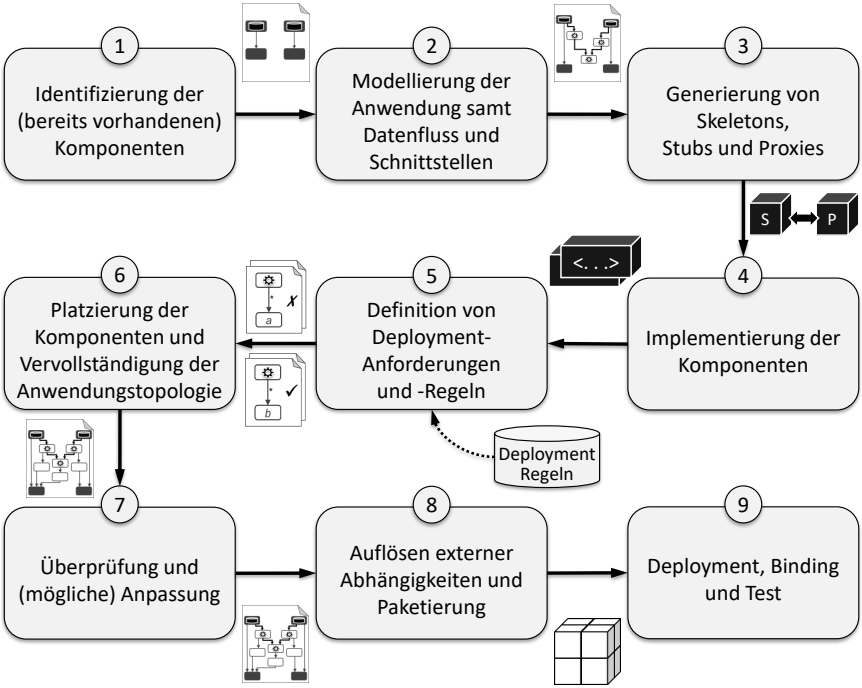


Abbildung 3.1: Übersicht der einzelnen Schritte der Shipping-Methode

3.2 Shipping-Methode

Die Shipping-Methode unterstützt die Benutzer, typischerweise die für eine Anwendung verantwortlichen Softwareentwickler, bei der Modellierung, Entwicklung und dem Deployment verteilter Anwendungen sowie einer dabei automatisierten Platzierung der beinhalteten Komponenten und Daten. Insgesamt umfasst die Methode neun Schritte, die in diesem Abschnitt im Detail vorgestellt werden. Eine Übersicht der einzelnen Schritte der Shipping-Methode wird in Abbildung 3.1 gegeben. In der Abbildung sind weiterhin die Eingabe- beziehungsweise die resultierenden Ausgabe-Artefakte der einzelnen Schritte der Methode skizziert, welche in den folgenden Abschnitten zu den jeweiligen Schritten jedoch nochmals genauer vorgestellt werden.

Durch die Verwendung bereits bestehender Artefakte oder dem Auslassen von optionalen Schritten, ergeben sich verschiedene Varianten der Methode, welche anschließend in Abschnitt 3.3 vorgestellt werden. In Abschnitt 3.4 werden zudem die Umsetzung sowie die Automatisierungsmöglichkeiten der einzelnen Schritte aufgezeigt. Für die durch weitere Forschungsbeiträge dieser Arbeit realisierten Schritte der Methode, wird zusätzlich auf die jeweiligen Abschnitte verwiesen. Im Falle der Wiederverwendung und Nutzung von bestehenden Forschungsarbeiten und Konzepten zur Umsetzung einzelner Schritte wird diesbezüglich auf die entsprechenden Arbeiten verwiesen.

3.2.1 Schritt 1: Identifizierung der Anwendungskomponenten

Der erste Schritt der Methode dient der Identifizierung aller für die Anwendung relevanten Komponenten. Dazu gehören neben typischen berechnenden oder verarbeitenden Softwarekomponenten, wie beispielsweise einem Dienst zur Berechnung von Durchschnittswerten oder einem Dienst zur Fehlervorhersage mittels der Analyse von Zeitreihen, auch Komponenten zur Datenübertragung sowie Datenhaltung. Eine Komponente zur Datenübertragung kann zum Beispiel ein vorgelagerter Adapter sein, welcher zur Anbindung sowie Datenabfrage einer Datenquelle dient. Im Bereich von Cyber-physischen Systemen [Jaz14; LBK15] wäre eine solche Datenquelle beispielsweise eine Maschine, auf die unter Umständen nur mit speziellen Protokollen zugegriffen werden kann [BOH+17]. Eine Komponente zur Datenhaltung ist weiterhin zum Beispiel eine Datenbank oder auch eine virtuelle Maschine, auf der die entsprechenden Daten nur lose auf dem Filesystem abgelegt werden. Falls bereits konkrete Daten in der Komponente vorliegen, kann diese Komponente darüber hinaus auch als eine Datenquelle der Anwendung betrachtet werden.

Weiterhin muss bei der Identifizierung aller beteiligten Komponenten zwischen (i) bereits deployten und betriebsbereiten Komponenten sowie (ii) noch zu deployenden Komponenten unterschieden werden. Dies ist vor allem für die spätere Gruppierung und Platzierung der Komponenten in

Schritt 6 (siehe Abschnitt 3.2.6) entscheidend, da die noch zu deployenden Komponenten, abgesehen von den in Schritt 5 zu definierenden Deployment-Anforderungen und Deployment-Regeln (siehe Abschnitt 3.2.5), beliebig und damit flexibel platziert werden können. Im Gegensatz dazu haben die bereits deployten Komponenten bei der späteren Gruppierung aller Anwendungskomponenten eine eher einschränkende Wirkung, da um den Datentransfer zu beschleunigen, die datenverarbeitenden Komponenten der Anwendung möglichst nahe beisammen liegen sollten (vgl. Abschnitt 2.2). Vor allem im Falle von Datenquellen, welche große Mengen an zu verarbeitenden Daten beinhalten, ist der Transfer der Daten zu den verarbeitenden Komponenten über eine große Distanz hinweg, typischerweise die schlechteste Wahl. Dementsprechend müssen in einem solchen Szenario insbesondere die bereits vorhandenen Datenquellen für die spätere Platzierung der restlichen Anwendungskomponenten identifiziert und berücksichtigt werden [ZBK+20].

Für die bereits betriebenen Anwendungskomponenten wird für die weiteren Schritte weiterhin jeweils das Instanzmodell, also die technische Struktur bestehend aus den vor allem zum Betrieb und Management der Anwendungskomponenten notwendigen Middleware- und Infrastrukturkomponenten und deren Relationen zueinander, benötigt. Vor allem die Infrastrukturkomponenten sind dabei wichtig, da sie Aufschluss darüber geben, wo genau eine bereits laufende Komponente betrieben wird. Ohne diese Informationen oder im Falle von falschen Informationen kann eine sinnvolle Gruppierung und Platzierung der Anwendungskomponenten im späteren Verlauf der Methode nicht erfolgreich durchgeführt werden. Die Erhebung des Instanzmodells kann dabei entweder durch Expertenwissen manuell oder durch automatisierte Konzepte [BBKL13; Bin15] erfolgen. Die Ergebnisse dieses Schrittes sind somit (i) die Identifizierung der für die Realisierung der Anwendung benötigten Komponenten sowie (ii) die entsprechenden Instanzinformationen im Falle von bereits deployten und aktuell betriebenen Komponenten.

3.2.2 Schritt 2: Modellierung der Anwendung samt Datenfluss und Schnittstellen der Komponenten

Der zweite Schritt der Methode ist die detaillierte Modellierung der Anwendung mittels einer deklarativen Modellierungssprache (Anforderung 2). Hierzu werden alle im ersten Schritt identifizierten anwendungsspezifischen Komponenten sowie deren Relationen zueinander als Anwendungstopologie modelliert (vgl. Abschnitt 2.4). Zusätzlich werden der Datenfluss innerhalb der Anwendung sowie der Datenfaktor und die Schnittstellen der einzelnen Komponenten modelliert. Der Datenfluss beschreibt den jeweils nötigen Transfer von Daten zwischen den Komponenten einer Anwendung durch die Spezifikation von Quellen und Senken. Falls Datenquellen mit bereits enthaltenen Daten Teil der Anwendung sind, sollten diese ebenfalls, falls möglich mit enthaltener Datenmenge, entsprechend modelliert werden. Der Datenfaktor dagegen bezieht sich jeweils auf eine einzelne Komponente und stellt deren Auswirkung auf die zu transferierende Datenmenge innerhalb der Anwendung dar. Der Datenfaktor einer datenverarbeitenden Anwendungskomponente wird im Rahmen dieser Arbeit wie folgt definiert:

Definition 3.1 (Datenfaktor)
Der Datenfaktor gibt die Auswirkung der Datenverarbeitung einer einzelnen Komponente auf die Datenmenge wieder. Sei DF_k der Datenfaktor einer datenverarbeitenden Komponente k, dann gilt:

- $DF_k < 1$, wenn die Datenmenge um DF_k reduziert wird,
- $DF_k > 1$, wenn die Datenmenge um DF_k vergrößert wird,
- $DF_k = 1$, wenn die Datenmenge bei der Verarbeitung gleich bleibt. ■

Falls die zu übertragende Datenmenge einer Komponente beispielsweise durch die Aggregation von Zeitreihendaten halbiert wird, entspricht der Datenfaktor dieser Komponente somit 0,5. Der Datenfaktor ist vor allem von der Verarbeitungslogik der jeweiligen Komponente abhängig. Da er jedoch zusätzlich von den zu verarbeitenden Daten beeinflusst werden kann, muss er typischerweise durch einen Experten der konkreten Anwendung bestimmt

bzw. abgeschätzt werden. Der Datenfaktor wird später dazu verwendet, um beispielsweise eine Komponente, die die Menge der zu übertragenden Daten reduziert, in der Nähe ihrer Datenquelle zu platzieren. Andererseits sollte eine Komponente, die die Menge der Daten erhöht, in der Nähe ihrer Datensenke platziert werden. Im Falle einer sich der Datenmenge gegenüber neutral verhaltenden Komponente (DF = 1), kann diese dagegen sowohl in der Nähe der Datenquelle als auch der Datensenke platziert werden, da sie die zu übertragende Datenmenge nicht beeinflusst. Der Datenfluss der Anwendung sowie der Datenfaktor der einzelnen Komponenten werden daher für die spätere Gruppierung und Platzierung der Komponenten benötigt (siehe Abschnitt 3.2.6). Die Modellierung der Schnittstellen der einzelnen Komponenten ist dagegen für die Generierung von Skeletons, Stubs und Proxies vorgesehen (siehe Abschnitt 3.2.3), um den Entwickler sowohl bei der Implementierung der einzelnen Komponenten an sich, als auch bei der Realisierung von Kommunikationsmöglichkeiten zwischen den verschiedenen Komponenten zu unterstützen. Das Ergebnis dieses Schrittes ist somit eine Anwendungstopologie, bestehend aus den anwendungsspezifischen Komponenten der zu deployenden Anwendung, welche zusätzlich mit weiteren Informationen, wie dem Datenfluss der Anwendung sowie dem Datenfaktor und den Schnittstellen der einzelnen Komponenten, angereichert ist.

3.2.3 Schritt 3: Generierung von Skeletons, Stubs und Proxies

Im dritten Schritt der Methode können, auf Basis der im vorherigen Schritt modellierten Schnittstellen der Anwendungskomponenten, Skeletons, Stubs sowie Proxies der entsprechenden Komponenten generiert werden. Der gewählte Ansatz folgt damit dem Proxy-Entwurfsmuster [GHJV94]. Der clientseitige Proxy ist dabei für die Initiierung der Kommunikation mit der anderen zu kommunizierenden Komponente verantwortlich. Weiterhin übergibt der Proxy die zur weiteren Verarbeitung benötigten Daten an den Stub und kümmert sich um die Abfrage des Ergebnisses der Bearbeitung. Dem Gegenüber ist der Stub für die gegebenenfalls nötige Übersetzung und Weitergabe der Daten an den auszuführenden Code verantwortlich. Das generierte Ske-

leton gibt die Grundstruktur einer Anwendungskomponente vor, die vom Entwickler mit der gewünschten Anwendungslogik ausgebaut werden muss. Mittels der Generierung von Skeletons, Stubs und Proxies wird somit die Implementierung der Komponenten sowie deren Möglichkeit zur gegenseitigen Kommunikation erleichtert und dementsprechend auch beschleunigt, da sich der Entwickler auf die Implementierung der Anwendungslogik fokussieren kann und die Kommunikation zwischen den Komponenten nicht manuell realisieren muss [ZBL17; ZBL18]. Da diese Konzepte weiterhin in die Modellierung der Anwendungstopologie integriert sind und von der entsprechenden Bereitstellungsumgebung unterstützt werden, kann somit zudem die ansonsten nötige Registrierung und das Binding der Komponenten automatisiert werden. Darüber hinaus erleichtert dieses Vorgehen für den Entwickler auch, anstelle der Entwicklung eines Monolithen, die Funktionen der Anwendung mithilfe separater Komponenten zu realisieren und von den entsprechenden Vorteilen zu profitieren [New15]. Das Ergebnis dieses Schrittes sind die generierten Skeletons, Stubs und Proxies der Komponenten der in Schritt 2 (siehe Abschnitt 3.2.2) modellierten Anwendung.

3.2.4 Schritt 4: Implementierung der Komponenten

In diesem Schritt werden die einzelnen, für die Realisierung der Anwendung benötigten, Komponenten implementiert. Der oder die Entwickler können dabei auf die im vorherigen Schritt generierten Skeletons, Stubs und Proxies zurückgreifen und damit zum Beispiel von den, auf Basis der in Schritt 2 (siehe Abschnitt 3.2.2) modellierten Schnittstellen der Komponenten, generierten Methodensignaturen und der damit vorgegebenen Struktur profitieren. Aufgrund der Zerlegung der Anwendungsfunktionalitäten in separate Komponenten sowie der zusätzlichen Abstraktion der Kommunikation mittels der generierten Stubs und Proxies, kann die Implementierung sowie Verantwortung der einzelnen Komponenten einfach auf mehrere Entwickler oder Teams verteilt und somit auch die jeweiligen Zuständigkeiten auf diese klar zugewiesen werden. Das Ergebnis dieses Schrittes sind somit die ausimplementierten Komponenten der Anwendung.

3.2.5 Schritt 5: Definition von Deployment-Anforderungen und Deployment-Regeln

Der korrekte Umgang mit Daten, zum Beispiel im Hinblick auf Datensicherheit und Compliance, ist ein wichtiges Thema in der IT [MKL09; ZBKL18; Zha18]. In Bereichen wie unter anderem IoT oder Industrie 4.0 kann beispielsweise das Sammeln und Auswerten von Daten zu neuartigen Erkenntnissen führen [HPO16; ZBF+17a]. Aber auch weitere Informationen, zum Beispiel bezüglich der innerhalb eines Unternehmens verfügbaren Infrastruktur, können zu sensiblen und schützenswerten Daten gezählt werden [WGR17; ZBF+17b; ZR13]. Allerdings ist die Umsetzung von unternehmensweiten Vorgaben bezüglich der Datenverarbeitung und damit auch dem Deployment von Anwendungen oft nicht trivial, insbesondere wenn verschiedene Deployment Technologien verwendet werden [BBF+14; BBK+13b; ZBF+17b].

Daher wird in diesem Schritt eine Möglichkeit zur einheitlichen Spezifikation solcher, innerhalb eines Unternehmens wiederverwendbaren, Regeln ermöglicht. Neben der reinen dokumentarischen Spezifikation ermöglichen sie auch die automatisierte Kontrolle sowie gegebenenfalls Verhinderung des Deployments bei Nichterfüllung. Da die Modellierung solcher Regeln zudem typischerweise eine andere Expertise als die Modellierung von Bereitstellungsmodellen erfordert, wird die Modellierung jeweils getrennt voneinander ermöglicht. Weiterhin erlaubt dies auch die Regeln, bei neu umzusetzenden oder veränderten Vorgaben, anzupassen ohne die Anwendungstopologie verändern zu müssen. Somit können beispielsweise Datenschutz und Compliance Experten wiederverwendbare und grundsätzliche für das Unternehmen geltende Regeln, unabhängig von konkreten Anwendungstopologien, spezifizieren sowie modellieren (Anforderung 7).

Neben den Deployment-Regeln, welche vor allem zur Umsetzung von Restriktionen und Vorgaben bezüglich zum Beispiel Datenschutz benötigt werden, können in diesem Schritt auch technische Anforderungen, zum Beispiel erforderliche Rechenleistung oder Speicherkapazität, für den Betrieb einer einzelnen Anwendungskomponente selbst spezifiziert werden. Da sich

diese Anforderungen typischerweise auf einen konkreten Anwendungsfall beziehen, können diese direkt in der entsprechenden Anwendungstopologie, welche in Schritt 2 (siehe Abschnitt 3.2.2) erstellt wurde, definiert werden. Da einige Anforderungen jedoch von der konkreten Implementierung der jeweiligen Komponente, zum Beispiel hinsichtlich der verwendeten Technologien und nötiger Middleware abhängig ist, kann die Definition dieser Anforderungen erst in Anschluss an Schritt 4 (siehe Abschnitt 3.2.4) erfolgen. Das Ergebnis dieses Schrittes ist somit die Definition der für die Anwendung geltenden Anforderungen bezüglich des Deployments sowie die Deployment-Regeln. Wobei die Deployment-Regeln entweder neu erstellt oder aus einem bestehendem Repository wiederverwendet werden können.

3.2.6 Schritt 6: Bestimmung der Platzierung der Komponenten und Vervollständigung der Anwendungstopologie

Die optimale Platzierung von Anwendungskomponenten kann abhängig von verschiedenen Merkmalen, zum Beispiel Herkunft der Daten, den entstehenden Kosten oder Anforderungen an die datenverarbeitenden Komponenten, variieren (vgl. Abschnitt 2.3). Sollen beispielsweise Daten aus verschiedenen Datensätzen, die sich in unterschiedlichen Rechenzentren oder Ländern befinden, analysiert und verarbeitet werden, kann es für die Performance einer Anwendung einen erheblichen Unterschied machen, wo die Verarbeitungslogik angesiedelt ist und ob sich die Datenmenge durch eine Vorverarbeitung reduzieren lässt, zum Beispiel durch die Aggregation einer Zeitreihe, oder ob sich diese vergrößert, beispielsweise beim Entpacken von komprimierten Archiven (vgl. Abschnitt 2.2). Darüber hinaus können auch unterschiedliche Datenschutzrichtlinien oder Gesetze die mögliche Nutzung von bestimmten Lokalitäten vorschreiben oder auch verhindern [Mah11; ZBK+20]. Bei komplexen Anwendungen, die mehrere verteilte Datensätze verarbeiten und aus mehreren Verarbeitungskomponenten bestehen, ist der manuelle Aufwand für die Evaluierung aller grundsätzlichen Möglichkeiten hinsichtlich der Gruppierung und Platzierung der Komponenten und Datensätzen immens. Dementsprechend wird in diesem Schritt der Methode, basierend auf der

in Schritt 2 (siehe Abschnitt 3.2.2) modellierten Anwendungstopologie und den in Schritt 5 (siehe Abschnitt 3.2.5) definierten Deployment Anforderungen und Regeln, eine Gruppierung und Platzierung der Komponenten automatisiert bestimmt (Anforderung 1). Hierzu werden insbesondere die in der Anwendungstopologie hinterlegten Informationen bezüglich dem Datenfluss der Anwendung, der Größe von existierenden Datensätzen sowie den Datenfaktoren der datenverarbeitenden Komponenten herangezogen.

Anwendungstopologien beschreiben die Anwendungskomponenten, einschließlich Middleware- und Infrastrukturkomponenten wie Webserver oder virtuelle Maschinen, sowie ihre Beziehungen zueinander und ermöglichen damit eine herstellerunabhängige und portable Beschreibung der Anwendungsimplementierung (vgl. Abschnitt 2.4). Die jeweils zur Verfügung stehenden Middleware- und Infrastrukturkomponenten können sich jedoch je nach Zielumgebung unterscheiden. Falls beispielsweise Anwendungen für Dritte implementiert und bereitgestellt werden sollen oder Teile der IT-Infrastruktur ausgelagert werden, ist die Zielumgebung nicht immer im Voraus bekannt [ZBF+17b]. Dementsprechend ist in solchen Fällen eine endgültige Modellierung der vollständigen Anwendungstopologie bereits in Schritt 2 der Methode (siehe Abschnitt 3.2.2) typischerweise nicht möglich. Daher wird in diesem Schritt, neben der Bestimmung der Gruppierung und Platzierung der anwendungsspezifischen Komponenten, ebenfalls auch die bisher unvollständige Anwendungstopologie ggf. automatisiert vervollständigt und damit entsprechend deploybar gemacht.

Abhängig von der Bestimmung der Gruppierung und Platzierung der Komponenten kann es eventuell vorkommen, dass die Anwendungstopologie nicht korrekt vervollständigt werden kann, beispielsweise weil für den erfolgreichen Betrieb der Anwendung benötigte Komponenten in der ermittelten Zielumgebung nicht verfügbar sind oder dort nicht betrieben werden können. Dementsprechend werden die Platzierung der Komponenten sowie die Vervollständigung der Anwendungstopologie zusammen in einem Schritt durchgeführt, um bei auftretenden Problemen bei der Vervollständigung, die Platzierung automatisch erneut, aber ohne die zuvor ermittelte und proble-

matische Zielumgebung, durchführen zu können. Damit soll sichergestellt werden, dass die Anwendungstopologie nach Beendigung dieses Schrittes vollständig ist und somit erfolgreich deployt werden kann. Das Ergebnis dieses Schrittes ist demzufolge eine vollständige und damit lauffähige Anwendungstopologie, mit einer speziell auf die Datenverarbeitung und den Datenfluss der Anwendung angepassten Platzierung der Komponenten.

3.2.7 Schritt 7: Überprüfung und Anpassung

In diesem Schritt wird dem Nutzer die manuelle Kontrolle und gegebenenfalls Anpassung der im vorherigen Schritt vervollständigten Anwendungstopologie ermöglicht (Anforderung 6). Dadurch wird auch sichergestellt, dass mögliche Fehler oder Probleme noch vor dem Deployment der Anwendung entdeckt und korrigiert werden können. Diese können beispielsweise durch eine bereits fehlerhafte Modellierung der Anwendungstopologie in Schritt 2 (siehe Abschnitt 3.2.2), fehlerhaft modellierten Deployment Anforderungen oder Regeln in Schritt 5 (siehe Abschnitt 3.2.5) oder Problemen bei der Vervollständigung der Anwendungstopologie in Schritt 6 (siehe Abschnitt 3.2.6) verursacht werden. Beispielweise, falls unerfüllbare Anforderungen definiert sind. Vor allem die Einhaltung von Datenschutz und Compliance Richtlinien, die bei Nichteinhaltung hohe Kosten verursachen könnten [Vos16], kann hier nochmals überprüft werden. Das Ergebnis dieses Schrittes ist somit eine, möglicherweise manuell angepasste, vollständige und in den jeweiligen Zielumgebungen der einzelnen Komponenten lauffähige Anwendungstopologie.

3.2.8 Schritt 8: Auflösen externer Abhängigkeiten sowie die Paketierung der Anwendung

Während des Deployments einer Anwendung müssen typischerweise externe Abhängigkeiten der Anwendung in der jeweiligen Zielumgebung aufgelöst, heruntergeladen und installiert werden [WBL14a]. In Linux-basierten Systemen wird beispielsweise häufig das Paketverwaltungsprogramm *apt-get* in Skripten oder Konfigurationsmanagementtools verwendet, um erforderliche

Abhängigkeiten zu installieren. Mit dem Befehl `apt-get install python` werden beispielsweise die zur Installation einer Python-Laufzeitumgebung benötigten Dateien heruntergeladen sowie installiert. Dies ermöglicht die Erstellung von leichtgewichtigen Deployment-Paketen, da externe Abhängigkeiten erst während der Deployment-Phase aufgelöst und heruntergeladen werden und nicht bereits im Voraus im Deployment-Pakete enthalten sein müssen [ZBH+20]. Dies minimiert nicht nur die Größe der Deployment-Pakete, sondern ermöglicht auch die ausschließliche Verwendung von Skripten. Grundsätzlich können zwei verschiedene Arten von externen Abhängigkeiten unterschieden werden: (i) Abhängigkeiten, die für die Bereitstellung und Verwaltung der Anwendung erforderlich sind (vgl. Definition 4.23), sowie (ii) Abhängigkeiten der Anwendung selbst, die für deren ordnungsgemäße Ausführung erforderlich sind (vgl. Definition 4.24). Falls allerdings der Zugang zum Internet begrenzt, instabil oder überhaupt nicht vorhanden ist, zum Beispiel aus Sicherheitsgründen wie in manchen IT-Umgebungen im Bereich Industrie 4.0 [TFK+18], kann das Auflösen und Herunterladen externer Abhängigkeiten jedoch die Bereitstellung erheblich verlangsamen oder verhindern. Weiterhin können extern verfügbare Abhängigkeiten im Laufe der Zeit auch veralten sowie aus dem Internet verschwinden und somit die Wiederverwendbarkeit unmöglich machen [MB16]. In solchen Fällen werden vollständige Deployment-Pakete benötigt, die alles enthalten was für die Bereitstellung und Verwaltung der Anwendung erforderlich ist.

Daher werden in diesem Schritt zuerst, falls vom Nutzer gewünscht, alle externen Abhängigkeiten aufgelöst und heruntergeladen sowie anschließend ein Deployment-Paket der Anwendung erstellt. Inhalt dieses Pakets sind dementsprechend unter anderem die aus Schritt 6 (siehe Abschnitt 3.2.6) beziehungsweise Schritt 7 (siehe Abschnitt 3.2.7) resultierende Anwendungstopologie der Anwendung, die in Schritt 5 (siehe Abschnitt 3.2.5) definierten Deployment-Regeln sowie alle zum Deployment, der Verwaltung und dem Betrieb der Anwendung benötigten Artefakte, wie beispielsweise die in Schritt 4 (siehe Abschnitt 3.2.4) implementierten Komponenten oder Konfigurationsdateien. Ein solches Deployment-Paket kann von einer

kompatiblen Bereitstellungsumgebung, typischerweise *„Deployment Engine“* genannt [BSLR10; FCR+13; FME12; WBKL16], interpretiert und verarbeitet werden, und dementsprechend zum automatisierten Deployment der Anwendung genutzt werden. Das Ergebnis dieses Schrittes ist ein Deployment-Paket der Anwendung, dass optional alle zum Deployment der Anwendung benötigten externen Abhängigkeiten enthält.

3.2.9 Schritt 9: Deployment, Verbindung und Test

Im letzten Schritt der Methode wird die Anwendung mithilfe der im vorherigen Schritt erstellten Deployment-Pakete sowie einer kompatiblen Deployment Engine automatisiert bereitgestellt. Die Wahl des hierzu genutzten Bereitstellungssystems ist dementsprechend von der in den vorherigen Schritten verwendeten Modellierungssprache abhängig. Grundsätzlich ist für die Anwendung der vorgestellten Methode jedoch jede Modellierungssprache mit kompatiblem Bereitstellungssystem nutzbar, insofern sie deklarative Anwendungstopologien, mit entsprechender Modellierung der Komponenten und deren Beziehungen zueinander, sowie das Anhängen von ausführbaren Artefakten und weiteren benötigten Informationen ermöglicht. Neben dem reinen Deployment der einzelnen Anwendungskomponenten werden auch die zum erfolgreichen Betrieb der Anwendung benötigten Verbindungen, zum Beispiel zwischen den Anwendungskomponenten sowie zu den bestehenden Datenquellen, in diesem Schritt hergestellt (engl. *„binding“* [LRS02]). Weiterhin können in diesem Schritt anschließend automatisierte Tests durchgeführt werden, welche die erfolgreiche Bereitstellung und Anbindung der Anwendung überprüfen. Das Ergebnis dieses Schrittes und damit auch der ganzen Methode, ist somit die getestete und erfolgreiche Bereitstellung der modellierten Anwendung, mit einer auf die Datenverarbeitung der Anwendung angepassten Gruppierung und Platzierung der einzelnen Komponenten sowie die Anbindung an gegebenenfalls vorhandene Datenquellen.

3.3 Varianten der Methode

Abhängig von der konkreten Ausprägung und dem Entwicklungsstand einer Anwendung, lassen sich unterschiedliche Varianten der vorgestellten Methode ableiten und anwenden. Weiterhin können je nach Kontext, in der die Methode angewandt wird, einige Schritte als optional angesehen und gegebenenfalls ausgelassen werden. Im Folgenden werden daher verschiedene mögliche Variationen der Shipping-Methode betrachtet.

3.3.1 Variante 1: Anwendung der Methode bei bereits implementierten Anwendungskomponenten

Diese Variante der Methode, dargestellt in Abbildung 3.2, kann bei bereits vorhandenen und implementierten Anwendungskomponenten angewandt werden. In einem solchen Fall kann nach der Modellierung der Anwendungstopologie sowie dem Datenfluss der Anwendung in Schritt 2 (siehe Abschnitt 3.2.2) direkt zur Definition der Deployment-Anforderungen und Deployment-Regeln in Schritt 5 (siehe Abschnitt 3.2.5) übergegangen werden. In Schritt 2 kann hier weiterhin die zusätzliche Modellierung der Schnittstellen der Anwendungskomponenten, welche eigentlich für die Generierung der Stubs und Proxies in Schritt 3 (siehe Abschnitt 3.2.3) und somit als Grundlage für die Implementierung der Komponenten in Schritt 4 (siehe Abschnitt 3.2.4) genutzt werden, weggelassen werden. Es wird hierbei davon ausgegangen, dass die einzelnen Anwendungskomponenten nicht nur bereits hinsichtlich ihrer Anwendungslogik, sondern auch zusätzlich mit einer Möglichkeit zur gegenseitigen Kommunikation, zum Beispiel mit entsprechenden REST Schnittstellen [Fie00], realisiert sind.

Ist dies nicht oder nur zum Teil der Fall, kann auch weiterhin die Möglichkeit zur Modellierung von Schnittstellen in der Anwendungstopologie und der anschließenden Generierung von Stubs und Proxies zur späteren Etablierung einer Kommunikationsmöglichkeit genutzt werden [ZBL17; ZBL18]. Hierbei ist allerdings zu beachten, dass diese grundsätzlich abhängig von-

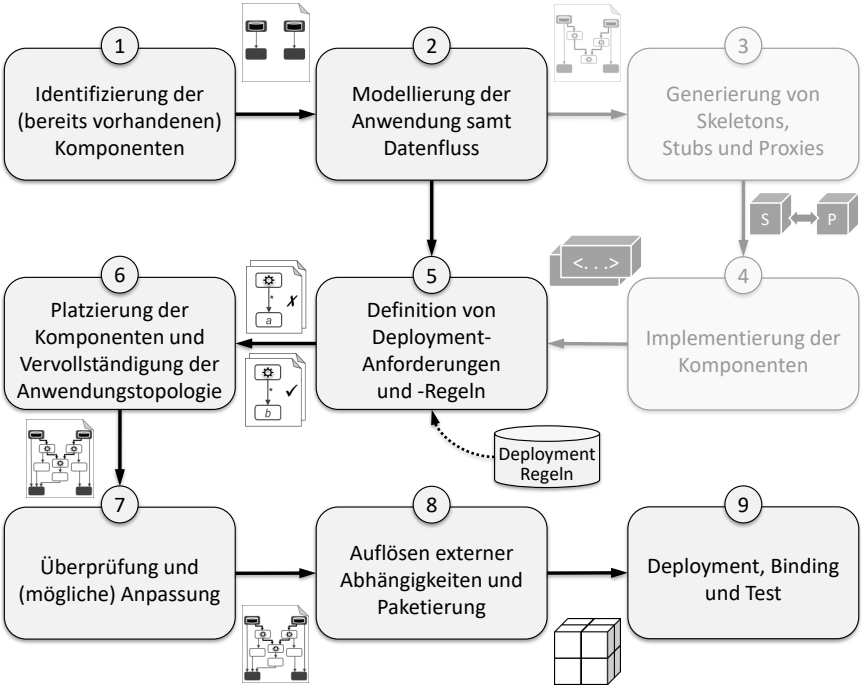


Abbildung 3.2: Variante der Shipping-Methode zur Anwendung bei bereits implementierten Anwendungskomponenten

einander sind und jeweils paarweise zusammen genutzt werden müssen. Weiterhin müssen diese im Anschluss an die Generierung jeweils in den bereits bestehenden Code der jeweiligen Komponente integriert werden.

Falls weiterhin alle zum Betrieb der Anwendung benötigten Komponenten, also auch benötigte Datenquellen, bereits zu Beginn der Methode bekannt sind, kann zusätzlich auf Schritt 1 der Methode (siehe Abschnitt 3.2.1) verzichtet werden und direkt mit der Modellierung der Anwendungstopologie in Schritt 2 begonnene werden. Dies könnte beispielsweise dann der Fall sein, wenn die Anwendung bereits betrieben wird und von einer manuellen Deployment-Lösung auf ein automatisiertes Deployment umgestellt werden soll. Die auf Schritt 5 folgenden Schritte der Methode bleiben identisch.

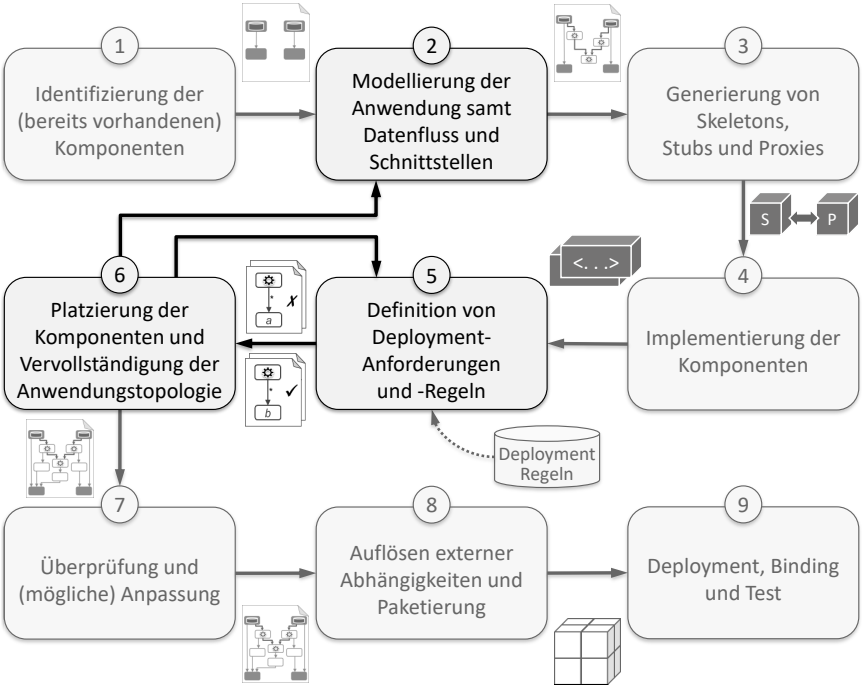


Abbildung 3.3: Variante der Shipping-Methode mit Rückschleife bei Platzierungs- oder Vervollständigungsproblemen

3.3.2 Variante 2: Rückschleife zur Behebung von auftretenden Problemen bei der Platzierung oder Vervollständigung

In Schritt 6 der Methode (siehe Abschnitt 3.2.6) können unter bestimmten Umständen Probleme auftreten, die die Durchführung dieses Schrittes verhindern. Auch wenn grundsätzlich bei auftretenden Problemen, zum Beispiel bei einer problematischen und fehlerhaften Kombination aus einer Komponente und der ihr zugeteilten Zielumgebung, die Ausführung dieses Schrittes, ohne erneute Berücksichtigung der problematischen Kombination aus Komponente und Zielumgebung, wiederholt wird, kann es möglicherweise Szenarien geben, in der kein erfolgreicher Ausführungszustand erreicht

werden kann. Beispielsweise aufgrund einer fehlerhaften Modellierung der Anwendungstopologie in Schritt 2 oder falls die in Schritt 5 definierten Deployment-Anforderungen und -Regeln sich gegenseitig ausschließen.

Dementsprechend ist in dieser Ausführungsvariante der Shipping-Methode eine Rückschleife zu Schritt 2 sowie Schritt 5 vorgesehen, um mögliche Fehler bei der Modellierung oder widersprüchliche und zu Problemen führende Deployment-Anforderungen und Deployment-Regeln gegebenenfalls beheben oder anpassen zu können. Diese Variante wird in Abbildung 3.3 dargestellt, wobei die zusätzlich hinzugefügte Kontrollschleife sowie die beteiligten Schritte optisch hervorgehoben sind. Falls zur Überarbeitung der Anwendungstopologie zurück zu Schritt 2 gegangen wird, können im Anschluss daran die beiden Schritte 3 und 4 übersprungen werden und mit Schritt 5 der Methode fortgesetzt werden. Anschließend kann die Platzierung der Komponenten sowie die Vervollständigung der Anwendungstopologie erneut ausgeführt werden, wobei alle zuvor, aufgrund der aufgetretenen Probleme, ausgeschlossenen Kombination wieder ermöglicht werden. Die auf Schritt 6 folgenden Schritte können dann normal durchgeführt werden.

3.3.3 Variante 3: Möglichkeit zur nachträglichen manuellen Einflussnahme

Auch wenn Schritt 6 erfolgreich ausgeführt wurde, kann die resultierende Anwendungstopologie unter Umständen noch Fehler enthalten oder dem Nutzer missfallen. Beispielsweise kann der Nutzer, aufgrund von persönlichen Erfahrungen, andere Cloud-Anbieter als die in Schritt 6 ausgewählten Anbieter bevorzugen oder Anwendungskomponenten lieber lokal betreiben, anstatt in den eigentlich für sie bestimmten Lokalitäten. Weiterhin kann die aus Schritt 6 resultierende Anwendungstopologie, aufgrund einer fehlerhaften Modellierung in Schritt 2 oder einer fehlerhaften Definition von Deployment-Anforderungen oder Deployment-Regeln in Schritt 5, aus Nutzersicht fehlerhaft sein, falls implizite Erwartungen oder Anforderungen des Nutzers an die Anwendungstopologie nicht erfüllt werden. Da dies jedoch zu einer eigentlich deploybaren, aber aus Nutzersicht fehlerhaften,

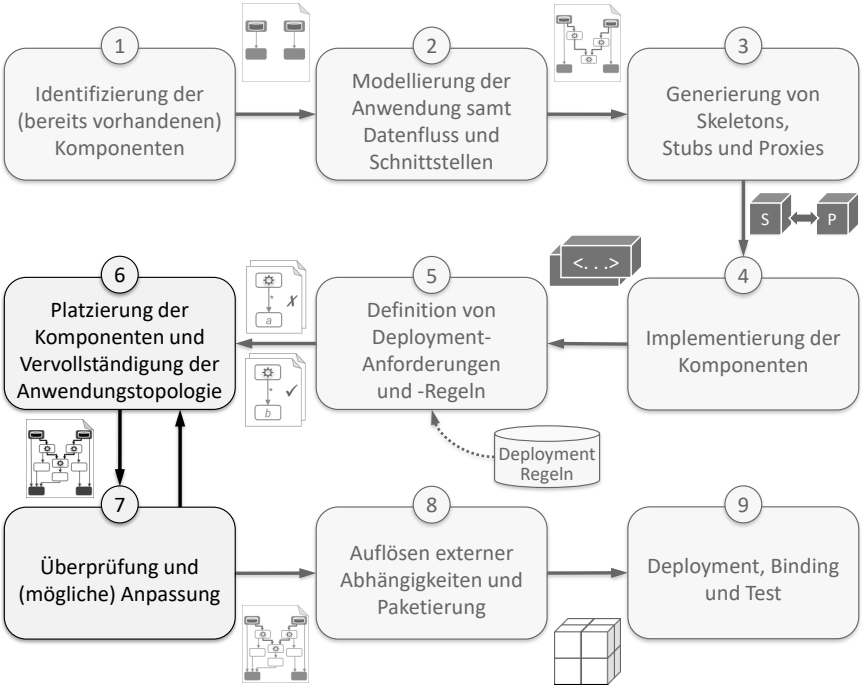


Abbildung 3.4: Variante der Shipping-Methode mit Möglichkeit zur nachträglichen manuellen Anpassung

Anwendungstopologie führt, würde dies daher im Vergleich zu der zuvor vorgestellten Variante nicht zu einem Abbruch bei der Durchführung von Schritt 6 führen und dementsprechend nicht automatisch auffallen.

Daher wird in dieser Variante der Methode dem Nutzer in Schritt 7 (siehe Abschnitt 3.2.7) eine letzte Möglichkeit gegeben, die Gruppierung und Platzierung der Anwendungskomponenten und damit das Deployment der gesamten Anwendung zu beeinflussen. Besonders im Hinblick auf Datenschutzrichtlinien und Compliance ist dieser letzte Schritt zur Kontrolle und Einflussnahme wichtig. Der Unterschied zur normalen Durchführung der Methode besteht darin, dass in dieser Variante die Anwendungstopologie oder die Deployment-Regeln nicht zwingend nur manuell angepasst werden

und anschließend mit Schritt 8 (siehe Abschnitt 3.2.8) fortgesetzt wird, sondern im Anschluss zurück zu Schritt 6 gegangen wird um die Platzierung und Vervollständigung erneut automatisiert durchzuführen. Dies ermöglicht beispielsweise nicht zur Einbeziehung bei der Platzierung vorgesehene Cloud-Anbieter nachträglich davon auszuschließen. Weiterhin kann eine fehlerhafte Definition oder Auswahl von Deployment-Regeln einen großen Einfluss auf die Gruppierung und Platzierung der Komponenten und damit auf die finale Anwendungstopologie haben, weshalb in solchen Fällen, anstatt die Anwendungstopologie manuell anzupassen, der automatisierte Schritt 6 zu bevorzugen ist. Eine Übersicht dieser Variante wird in Abbildung 3.4 dargestellt, wobei die im Vergleich zur ursprünglichen Methode angepassten Stellen hervorgehoben sind. Es kann beliebig oft von Schritt 7 zu Schritt 6 zurückgegangen werden. Falls die Anwendungstopologie schließlich vom Nutzer akzeptiert wird, kann die Methode normal fortgeführt werden.

3.3.4 Variante 4: Beschleunigung der Methodendurchführung durch das Weglassen optionaler (Teil-)Schritte

Nicht immer werden alle Funktionen und Schritte, die die Methode zur Verfügung stellt, gewünscht oder benötigt. Daher werden in dieser Ausführungsvariante die optionalen (Teil-)Schritte aufgezeigt und beschrieben, wann und warum sie unter gewissen Umständen bei der Durchführung der Methode weggelassen werden können. Grundsätzlich können bestimmte (Teil-)Schritte der Methode vor allem zur Beschleunigung der Methodendurchführung weggelassen werden, zum Beispiel falls nur ein Testdurchlauf vorgenommen wird oder zu Demonstrationszwecken möglicherweise nur mit Mockups gearbeitet wird. Die optionalen Schritte beziehungsweise Teilschritte sind in Abbildung 3.5 mit gestricheltem Rand dargestellt. Der erste optionale Schritt der Methode ist der Schritt 5, zur Definition von Deployment-Anforderungen und Deployment-Regeln. Falls beispielsweise keine sensiblen und schützenswerten Daten verarbeitete werden und es somit unerheblich ist, wo die Komponenten schlussendlich platziert werden sowie weiterhin auch keine sonstigen besonderen Anforderungen nötig sind,

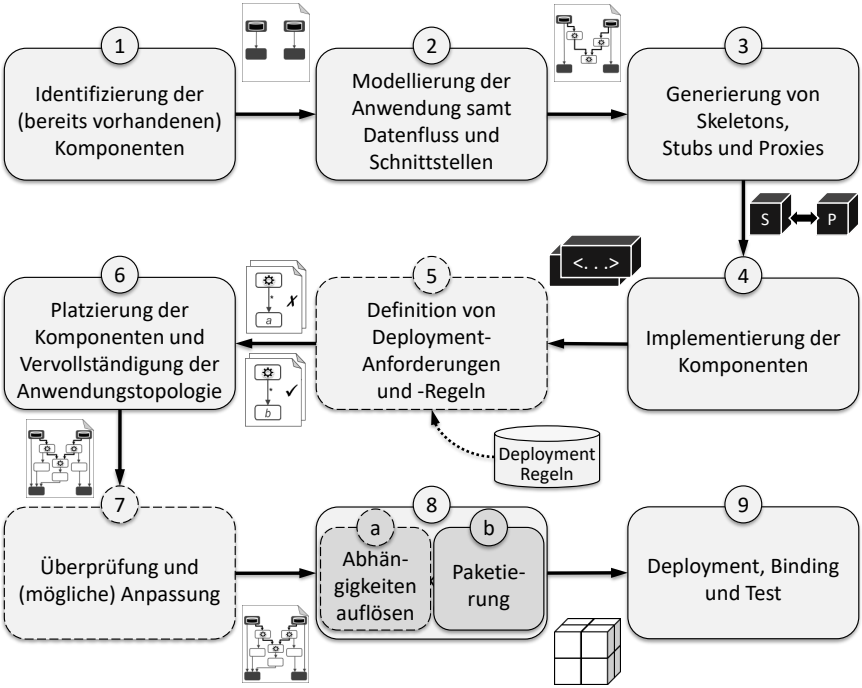


Abbildung 3.5: Übersicht der optionalen (Teil-)Schritte der Shipping-Methode

kann auf diesen Schritt verzichtet werden und im Anschluss an Schritt 4 mit Schritt 6 fortgesetzt werden. Entsprechend kann auch Schritt 7 aus den gleichen Gründen, dass es keine besonderen Anforderungen und Richtlinien bezüglich des Deployments der Anwendung zu beachten gilt, übersprungen werden und somit auf eine letzte manuelle Prüfung verzichtet werden. Wie zuvor erläutert, birgt dies jedoch die Gefahr, dass manuelle Modellierungsfehler dadurch unentdeckt bleiben und nicht korrigiert werden können.

Da Schritt 8 aus den beiden miteinander verknüpften Teilschritten (a) *externe Abhängigkeiten auflösen* sowie (b) *Paketierung* bestehen, wurden diese in der Abbildung entsprechend als Teilschritte von Schritt 8 dargestellt. Während der zweite Teilschritt, also die Paketierung der Anwendungstopologie sowie

aller weiteren benötigten Artefakte, zwingend nötig ist, um die paketierte Anwendung anschließend entweder zu versenden oder mithilfe einer entsprechenden Deployment Engine zu deployen, ist der erste Teilschritt von Schritt 8 optional und wird nur in bestimmten Szenarien benötigt. Zum Beispiel, falls in der Zielumgebung kein oder nur begrenztes oder eingeschränktes Internet zur Verfügung steht und somit das Herunterladen von externen Abhängigkeiten das Deployment der Anwendung verhindern oder zumindest stark verlangsamen würde. Ebenso kann die Auflösung und Paketierung von externen Abhängigkeiten in diesem Schritt genutzt werden um möglicherweise nur temporär zur Verfügung stehende Ressourcen für eine Verwendung zu einem späteren Zeitpunkt zu sichern.

Alle weiteren Schritte der Methode werden, wie in ihrem jeweiligen Abschnitt vorgestellt, normal ausgeführt. Die optionalen (Teil-)Schritte haben somit keinen weiteren direkten Einfluss auf den weiteren Durchführungsverlauf der Methode und können daher je nach Szenario und Kontext einfach und flexibel ohne benötigte Anpassungen der Methode ausgeführt oder weggelassen werden. Auch können je nach Anforderungen und Bedarf alle sowie auch nur einzelne der optionalen (Teil-)Schritte weggelassen werden.

3.4 Automatisierung und Umsetzung der Shipping-Methode

In diesem Abschnitt wird beschrieben, welche Schritte der Shipping-Methode manuell ausgeführt werden müssen und welche Schritte automatisiert beziehungsweise teilautomatisiert durchgeführt werden können. Weiterhin wird für die durch weitere Forschungsbeiträge dieser Arbeit realisierten Schritte jeweils auf das dazugehörige Kapitel verwiesen, indem der jeweilige Forschungsbeitrag im Detail vorgestellt wird.

Konzeptionell kann die Shipping-Methode vollständig manuell durchgeführt werden. Da jedoch bereits die manuelle Ausführung nur einzelner Schritte, zum Beispiel das Deployment einer Anwendung, bei komplexen Anwendungen in der Regel ineffizient, fehleranfällig sowie zeit- und kostenintensiv ist (vgl. Abschnitt 2.4) und zudem für die Durchführung der gesamten Methode

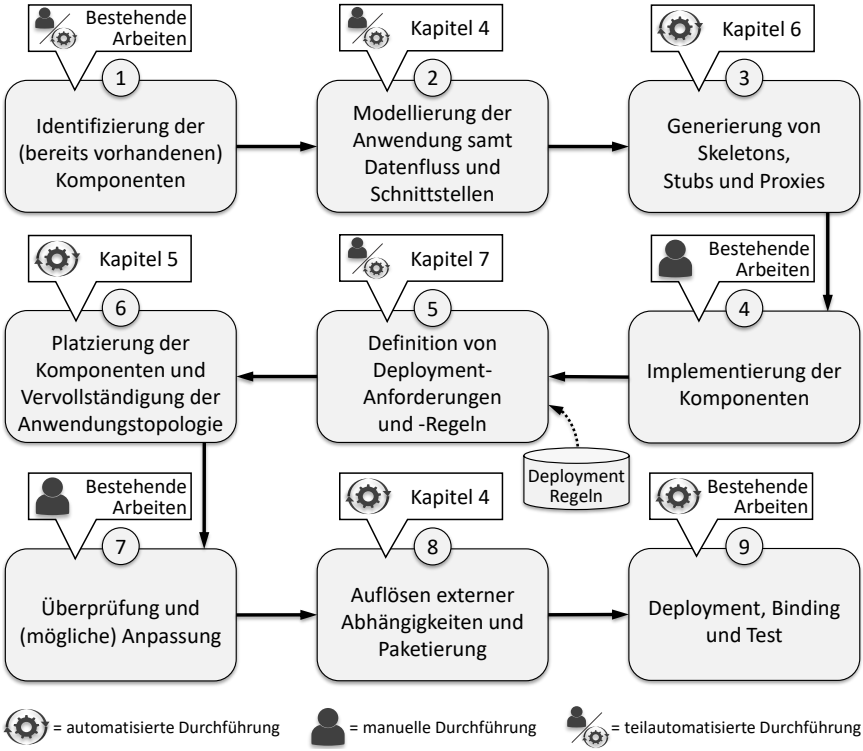


Abbildung 3.6: Automatisierung und Umsetzung der Shipping-Methode

typischerweise unterschiedliche Experten benötigt werden, zum Beispiel zur Modellierung der Anwendungstopologie, der Implementierung der einzelnen Anwendungskomponenten sowie der Einhaltung von Compliance-Richtlinien, ist die automatisierte Durchführung möglichst vieler Schritte zu bevorzugen (Anforderung 1). In Abbildung 3.6 wird daher eine Übersicht der Shipping-Methode gegeben, in der dargestellt ist, welche Schritte manuell durchgeführt werden müssen und welche Schritte automatisiert durchgeführt werden können. Weiterhin wird in der Abbildung auf die jeweiligen Kapitel verwiesen, in denen die erarbeiteten Forschungsbeiträge dieser Arbeit zur Umsetzung der Schritte detailliert vorgestellt werden.

Der erste Schritt der Methode, die Identifizierung der benötigten Anwendungskomponenten sowie die Modellierung der Instanzmodelle der bestehenden Komponenten, kann zum Teil automatisiert erfolgen. Während die Identifizierung der benötigten Anwendungskomponenten manuell durchgeführt werden muss, können für die Erstellung von Instanzmodellen bereits vorhandener Komponenten bestehende und automatisierte Konzepte [BBKL13; FAB+11; HBLE14; MDW+00] genutzt werden.

Zur Umsetzung des zweiten Schrittes der Methode wird in Forschungsbeitrag 2 (siehe Kapitel 4) eine Möglichkeit zur Erstellung von Anwendungstopologien verteilbarer Anwendungen sowie den Schnittstellen der einzelnen Anwendungskomponenten vorgestellt. Grundsätzlich muss dieser Schritt, insbesondere die Modellierung der Abhängigkeiten zwischen den einzelnen Komponenten sowie deren Schnittstellen, manuell durchgeführt werden. Falls jedoch ein Datenflussdiagramm der Anwendung besteht, kann die Anwendungstopologie basierend darauf generiert werden (siehe Abschnitt 5.2). Weiterhin werden verschiedene Modellierungskonzepte für die Anwendung in unterschiedlichen Szenarien betrachtet und diskutiert.

Schritt 3 wird mit Forschungsbeitrag 4 (siehe Kapitel 6) umgesetzt. Dabei werden basierend auf der zuvor modellierten Anwendungstopologie Skeletons, Stubs und Proxies generiert, welche zur Entwicklung der verteilbaren und kommunizierenden Komponenten der Anwendung genutzt werden können. Dadurch kann die Kommunikation abstrahiert sowie die Registrierung und das Binding der Komponenten automatisiert werden [ZBL17; ZBL18].

Basierend darauf muss anschließend die individuelle Anwendungslogik der einzelnen Komponenten in Schritt 4 manuell implementiert werden. Hierzu existieren unterschiedlichste wissenschaftliche Arbeiten, zum Beispiel bezüglich Best Practices bei der Analyse und Verarbeitung von Big Data in der Cloud [ACB+15; FRL+14; RCBW16; RFG+18].

Die Definition von Deployment-Anforderungen und Deployment-Regeln wird in Forschungsbeitrag 5 (siehe Kapitel 7) vorgestellt. Falls neue Anforderungen oder Regeln benötigt werden, müssen diese manuell definiert und modelliert werden. Falls jedoch beispielsweise alle Compliance-Richtlinien

eines Unternehmens berücksichtigt werden sollen, können diese aus einem bestehenden Repository entnommen und automatisiert angewandt werden.

Die Gruppierung und Platzierung der Anwendungskomponenten sowie die Vervollständigung der Anwendungstopologie in Schritt 6 wird mittels Forschungsbeitrag 3 (siehe Kapitel 5) automatisiert ermöglicht. Hierzu werden unter anderem der modellierte Datenfluss der Anwendung sowie die zuvor definierten Deployment-Anforderungen und Deployment-Regeln genutzt.

Schritt 7 der Shipping-Methode dient der Überprüfung und gegebenenfalls der Anpassung der Anwendungstopologie. Da die nachfolgenden Schritte automatisiert ausgeführt werden und diese die Anwendungstopologie nicht mehr verändern, wird dem Nutzer hier eine letzte Möglichkeit zur Kontrolle sowie, bei Bedarf, zur Beeinflussung der finalen Anwendungstopologie gegeben. Dementsprechend ist die manuelle Ausführung für diesen Schritt vorgesehen. Die Problemerkennung und Überprüfung von verteilten Anwendungen beziehungsweise deren Anwendungstopologien kann durch existierende Arbeiten [BDS17; CIP04; PTBP08; SBKL18; WCO03] umgesetzt werden. Dies ist daher nicht Teil dieser Arbeit.

Die gegebenenfalls nötige Auflösung externer Abhängigkeiten sowie die Paketierung der Anwendungstopologie samt allen weiteren benötigten Artefakten in Schritt 4, wird mittels Forschungsbeitrag 2 (siehe Kapitel 4) realisiert. Da die Modellierung sowie die Paketierung von Anwendungstopologien eng miteinander verknüpft sind, werden diese Punkte zusammen und in Abhängigkeit voneinander vorgestellt.

Der letzte Schritt der Shipping-Methode, bestehend aus dem Deployment, Konfigurieren und Verbinden der Komponenten sowie gegebenenfalls dem Ausführen von Systemtests, kann unter anderem mithilfe der in Abschnitt 2.4 vorgestellten Konzepten sowie weiteren existierenden Arbeiten [BEK+16; DSVT07; HMD+15; PBL+17; WBKL18] automatisiert umgesetzt werden und wird daher im Rahmen der vorliegenden Arbeit nicht im Detail betrachtet. Im Rahmen der Vorstellung der prototypischen Implementierung der in dieser Arbeit vorgestellten Konzepte (siehe Kapitel 8) wird jedoch eine zur Modellierungssprache kompatible Laufzeitumgebung vorgestellt.

3.5 Zusammenfassung und Diskussion

Die vorgestellte Shipping-Methode ermöglicht das automatisierte Verteilen und Platzieren von Funktionen und Daten verteilter Anwendungen in heterogenen IT-Umgebungen (vgl. Abschnitt 2.1). Der Fokus bei der Platzierung der Anwendungskomponenten liegt dabei vor allem auf der Optimierung des Datenflusses der Anwendung sowie einer möglichst hohen Datenlokalität (siehe Abschnitt 2.2). Weiterhin wird der Nutzer durch die Durchführung der Methode von der Entwicklung der Anwendung bis hin zum Betrieb der Anwendung unterstützt. Die Methode umfasst somit neben der Platzierung der Komponenten auch die deklarative Modellierung der Anwendung (Anforderung 2), die Implementierung der einzelnen Komponenten, die Definition von Richtlinien, die bei der Verteilung und dem Deployment eingehalten werden müssen, sowie das Deployment und Management der Anwendung. Die weiteren Forschungsbeiträge dieser Arbeit, welche die einzelnen Schritte der Shipping-Methode umsetzen und zum Teil automatisieren (Anforderung 1), werden in den nachfolgenden Kapiteln im Detail vorgestellt.

Da die Anwendungskomponenten sowie deren Abhängigkeiten zueinander bereits in Schritt 2 modelliert werden und in Schritt 6 die Anwendungstopologie nur mit den nötigen Middleware- und Infrastrukturkomponenten ergänzt wird, bleibt die Funktionalität der Anwendung erhalten und wird von der konkreten Platzierung der Anwendungskomponenten nicht beeinflusst (Anforderung 3). Da weiterhin die deklarative Modellierung sowie das Deployment der Anwendung grundsätzlich unabhängig von konkreten Technologien und Anbietern ist, und diese erst bei der Vervollständigung der Anwendungstopologie in Schritt 6, in Abhängigkeit von den in Schritt 4 jeweils mit einer beliebigen Programmiersprache implementierten Komponenten, in Betracht gezogen werden, ist die Methode sowohl Technologie-agnostisch (Anforderung 4) als auch Anbieter-agnostisch (Anforderung 5).

Durch die Aufteilung der Methode in die einzelnen Schritte und der damit erreichten Trennung von Zuständigkeiten, wie zum Beispiel der Modellierung, der Implementierung sowie der Definition und Auswahl von einzuhalten-

den Richtlinien beim Deployment, ermöglicht die Methode außerdem die einfache Einbeziehung verschiedener Experten beziehungsweise die Wiederverwendung von bestehendem Expertenwissen (Anforderung 7). Da die Methode weiterhin sowohl aus automatisierten Schritten, beispielsweise die Gruppierung und Platzierung der Komponenten, als auch manuellen Schritten besteht, unter anderem zum Beispiel Schritt 7 zur Überprüfung und gegebenenfalls Anpassung der Anwendungstopologie, können mögliche Fehlerquellen einerseits durch die Automatisierung vermindert werden und andererseits durch die Möglichkeit der manuellen Kontrolle noch vor dem Deployment der Anwendung rechtzeitig erkannt werden (Anforderung 6).

Im Rahmen dieser Arbeit beziehen sich die in Schritt 5 zu definierenden Deployment-Anforderungen sowie Deployment-Regeln vor allem auf die Spezifikation von technischen Anforderungen sowie Richtlinien zur Einhaltung von Datensicherheit und Compliance. Weiterhin liegt der Fokus dieser Arbeit bei der Gruppierung und Platzierung der Komponenten in Schritt 6, vor allem hinsichtlich der Optimierung des Datenflusses der Anwendung. Es können jedoch grundsätzlich auch weitere Anforderungen und Kriterien definiert werden, welche bei der Gruppierung und Platzierung der Komponenten sowie bezüglich der allgemeinen Auswahl an bestimmten Anbietern herangezogen werden sollen (Anforderung 8). Verschiedene Beispiele sowie konzeptionelle Ansätze hierzu wurden in Abschnitt 2.3 vorgestellt. Um die Methode hierfür zu erweitern, müssen die beiden Schritte 5 und 6 angepasst werden um die zusätzlichen Kriterien und Optimierungsziele definieren zu können beziehungsweise zu berücksichtigen. Die restlichen Schritte der Shipping-Methode können unverändert beibehalten werden.

Zusammengefasst ermöglicht die Shipping-Methode, anders als die in Abschnitt 2.2 und Abschnitt 2.3 vorgestellten Arbeiten, das Verteilen und Platzieren sowohl einzelner Anwendungskomponenten und Datenquellen als auch ganzer Softwarestacks unter Berücksichtigung des Datenflusses einer datenverarbeitenden Anwendung sowie weiteren definierten Anforderungen und Richtlinien, zum Beispiel bezüglich des Datenschutzes. Die Methode verfolgt dabei einen modellbasierten deklarativen Ansatz und unterstützt die

Nutzer damit ganzheitlich von der Entwicklung einer anbieterunabhängigen sowie technologieunabhängigen Anwendung bis zu deren Betrieb: von der Modellierung der Anwendung, über die Implementierung der einzelnen Anwendungskomponenten, der automatisierten Gruppierung und Platzierung aller benötigten Komponenten, der Paketierung der Anwendung, bis hin zum automatisierten Deployment und dem Betrieb der Anwendung.

Kapitel 4

Modellierung und Paketierung von verteilbaren Anwendungen

Die Modellierung der zu verteilenden Anwendungen ist einer der wichtigsten Schritte der in Kapitel 3 vorgestellten Shipping-Methode, da die Ausführung der einzelnen darauf folgenden Schritte auf diesem Modell basieren. Daher wird in diesem Kapitel das zur deklarativen Modellierung einer verteilbaren Anwendung nutzbare D^{3}M-Metamodell[1] vorgestellt. Dieses erlaubt neben der Modellierung von Komponenten und deren Relationen zueinander (vgl. Definition 2.3) auch die Modellierung der zur Durchführung der Shipping-Methode zusätzlich benötigten Informationen, wie zum Beispiel

[1] Declarative Deployment and Component Distribution Model

die Datenfaktoren (vgl. Definition 3.1) sowie Schnittstellen der einzelnen datenverarbeitenden Anwendungskomponenten. Weiterhin ist die erfolgreiche Paketierung der modellierten Anwendung sowie aller benötigter Artefakte in einem automatisiert deploybaren und portablen Format, im vorletzten Schritt der Methode, ebenfalls entscheidend für die effiziente Nutzbarkeit sowie Durchführbarkeit der gesamten Methode. Da die Modellierung und Paketierung einer Anwendung eng miteinander verknüpft sind, werden diese beiden Bereiche zusammen in diesem Kapitel behandelt. Dieses Kapitel stellt somit Forschungsbeitrag 2 dieser Arbeit dar.

In Abschnitt 4.1 wird zunächst das D^{3}M-Metamodell als Modellierungssprache zur Modellierung von verteilbaren Anwendungen vorgestellt. Anschließend werden in Abschnitt 4.2 drei verschiedene Modellierungskonzepte bezüglich des Deployments von D^{3}M-basierten Anwendungen vorgestellt. In Abschnitt 4.3 werden die Erstellung und Paketierung von eigenständigen Deployment-Paketen für D^{3}M-basierte Anwendungen erläutert. In Abschnitt 4.4 wird abschließend das eingeführte D^{3}M-Metamodell sowie die vorgestellten Modellierungs- und Paketierungskonzepte im Kontext dieser Arbeit diskutiert und das Kapitel zusammengefasst.

4.1 D^{3}M-Metamodell

Die in Kapitel 3 vorgestellte Shipping-Methode basiert auf deklarativen Anwendungstopologien. Daher wird mit dem D^{3}M-Metamodell zunächst eine Formalisierung zur deklarativen Modellierung von verteilbaren Anwendungen definiert, womit sich alle für die Durchführung der Methodenschritte benötigten Informationen, wie die Datenfaktoren der Anwendungskomponenten, die Schnittstellen der Anwendungskomponenten, die Menge der zu verarbeitenden Daten sowie der Datenfluss der Anwendung modellieren lassen. Angelehnt ist die Sprache an TOSCA [OAS13b], EDMM [WBF+19] sowie DMMN [Bre16]. In Abbildung 4.1 wird eine Übersicht des D^{3}M-Metamodells gegeben. Darin sind die verschiedenen Klassen und deren Relationen aufgezeigt, wobei gestrichelte Rechtecke abstrakte Klassen darstellen.

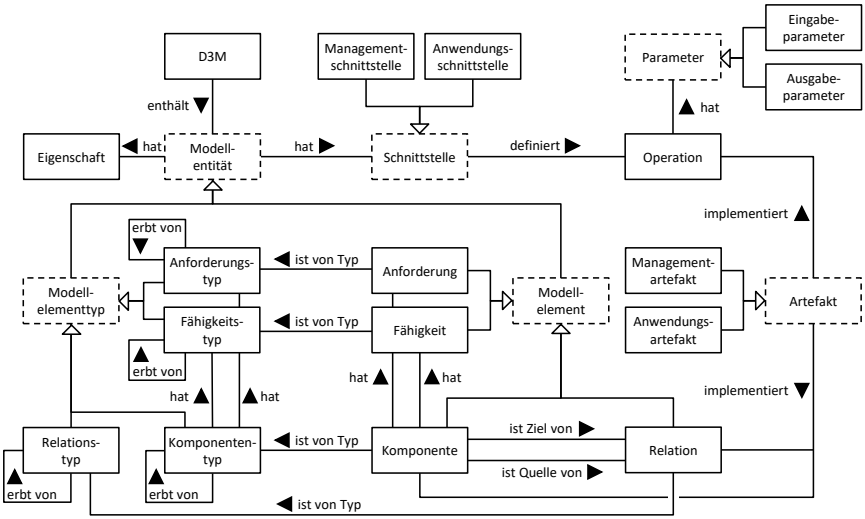


Abbildung 4.1: Übersicht des D^{3}M-Metamodells

Bevor die einzelnen Bestandteile des D^{3}M-Metamodells in den nachfolgenden Abschnitten vorgestellt werden, wird der Abbildung 4.1 folgend zunächst ein D^{3}M formal wie folgt definiert:

Definition 4.1 (D^{3}M)
Ein *Declarative Deployment and Component Distribution Model* ist ein gerichteter sowie gewichteter Multigraph, der die Struktur einer Anwendung beschreibt. Sei D die Menge aller D^{3}Ms, dann ist $d \in D$ folgendermaßen als ein Tupel definiert:

$$
\begin{aligned}
d = (&K_d, R_d, KT_d, RT_d, A_d, F_d, AT_d, FT_d, E_d, P_d,\\
&O_d, S_d, AE_d, typ_d, supertyp_d, anforderungen_d,\\
&fähigkeiten_d, eigenschaften_d, parameter_d, operationen_d,\\
&mSchnittstellen_d, aSchnittstellen_d, mArtefakte_d, aArtefakte_d)
\end{aligned}
$$

Hierbei bezeichnet

- K_d die Menge aller Komponentenelemente (siehe Definition 4.2),
- R_d die Menge aller Relationselemente (siehe Definition 4.3),
- KT_d die Menge aller Komponententypelemente (siehe Definition 4.4),
- RT_d die Menge aller Relationstypelemente (siehe Definition 4.5),
- A_d die Menge aller Anforderungselemente (siehe Definition 4.6),
- F_d die Menge aller Fähigkeitselemente (siehe Definition 4.7),
- AT_d die Menge aller Anforderungstypelemente (siehe Definition 4.9),
- FT_d die Menge aller Fähigkeitstypelemente (siehe Definition 4.8),
- E_d die Menge aller Eigenschaftselemente (siehe Definition 4.12),
- P_d die Menge aller Parameterelemente (siehe Definition 4.18),
- O_d die Menge aller Operationselemente (siehe Definition 4.15),
- S_d die Menge aller Schnittstellenelemente (siehe Definition 4.19) und
- AE_d die Menge aller Artefaktelemente (siehe Definition 4.22)

in d. Weiterhin bezeichnet

- typ_d die Abbildung, die jedem Komponentenelement, Relationselement, Anforderungselement sowie Fähigkeitselement in d, jeweils dessen Komponententypelement, Relationstypelement, Anforderungstypelement sowie Fähigkeitstypelement zuordnet (siehe Definition 4.25),
- $supertyp_d$ die Abbildung, die jedem Komponententypelement, Relationstypelement, Anforderungstypelement sowie Fähigkeitstypelement in d, falls vorhanden, jeweils dessen Supertypelement zuordnet (siehe Definition 4.26),
- $anforderungen_d$ die Abbildung, die jedem Komponententypelement beziehungsweise Komponentenelement in d, jeweils dessen Anforderungstypelemente beziehungsweise Anforderungselemente zuordnet (siehe Definition 4.27),

- $fähigkeiten_d$ die Abbildung, die jedem Komponententypelement beziehungsweise Komponentenelement in d, jeweils dessen Fähigkeitstypelemente beziehungsweise Fähigkeitselemente zuordnet (siehe Definition 4.28),
- $eigenschaften_d$ die Abbildung, die jedem Komponentenelement, Komponententypelement, Relationselement, Relationstypelement, Anforderungselement, Anforderungstypelement, Fähigkeitselement sowie Fähigkeitstypelement in d, jeweils dessen Eigenschaftselemente zuordnet (siehe Definition 4.29),
- $parameter_d$ die Abbildung, die jedem Operationselement in d, jeweils dessen Eingabeparameterelemente sowie Ausgabeparameterelemente zuordnet (siehe Definition 4.30),
- $operationen_d$ die Abbildung, die jedem Managementschnittstellenelement sowie Anwendungsschnittstellenelement in d, jeweils dessen Operationselemente zuordnet (siehe Definition 4.31),
- $mSchnittstellen_d$ die Abbildung, die jedem Komponentenelement, Komponententypelement, Relationselement sowie Relationstypelement in d, jeweils dessen Managementschnittstellenelemente zuordnet (siehe Definition 4.32),
- $aSchnittstellen_d$ die Abbildung, die jedem Komponentenelement sowie Komponententypelement in d, jeweils dessen Anwendungsschnittstellenelemente zuordnet (siehe Definition 4.33),
- $mArtefakte_d$ die Abbildung, die jedem Komponentenelement, Relationselement sowie Operationselement in d, jeweils dessen Managementartefaktelemente zuordnet (siehe Definition 4.34) und
- $aArtefakte_d$ die Abbildung, die jedem Komponentenelement sowie Operationselement in d, jeweils dessen Anwendungsartefaktelemente zuordnet (siehe Definition 4.35). ■

Nachfolgend sind die einzelnen Elemente eines D^{3}Ms nochmals detaillierter beschrieben sowie formal definiert.

Grundlage einer Anwendungstopologie sind Knoten zur Repräsentation der Komponenten einer Anwendung sowie Kanten zur Repräsentation der Relationen zwischen diesen Komponenten. Diese Elemente bilden die grundlegende Struktur einer Anwendung ab. Beispiele für solche in einer Anwendungstopologie vorhandenen Komponenten sind unter anderem virtuelle Maschinen, IoT-Geräte, Server, Betriebssysteme, Datenbanken, Middleware-Komponenten, wie beispielsweise Anwendungsserver oder Message Oriented Middleware (MOM), sowie Anwendungskomponenten, welche die eigentliche Geschäftslogik einer Anwendung implementieren.

Definition 4.2 (Komponentenelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $K_d = \{k_1, k_2, ..., k_n\}$ die Menge aller Komponentenelemente in d. Jedes $k_i \in K_d$ repräsentiert dabei eine Komponente der Anwendung, wobei eine Komponente dabei von logischer, physikalischer oder funktionaler Natur sein kann. Es gilt $k_i = (Id, Name)$, wobei $Id \in \Sigma^+$ das Komponentenelement global eindeutig identifiziert und $Name \in \Sigma^+$ das Komponentenelement eindeutig in d repräsentiert. Dabei stellt Σ^+ die Menge aller nicht-leeren Zeichenketten dar. ■

Relationen beschreiben zum Beispiel, dass eine Komponente *Bestell-App* auf einer darunter liegenden Komponente *PHP-Server* betrieben wird. Weitere Beispiele solcher Relationen wären, dass die Komponente *Bestell-App* eine Verbindung zu einer anderen Komponente *Bestellungen-DB* aufbaut oder die erfolgreiche Bereitstellung und Verwendung der Komponente von bestimmten weiteren Komponenten abhängig ist.

Definition 4.3 (Relationselemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $R_d = \{r_1, r_2, ..., r_n\}$ die Menge aller Relationselemente in d. Es gilt $r_i = (Id, Name)$, wobei $Id \in \Sigma^+$ das Relationselement global eindeutig identifiziert und $Name \in \Sigma^+$ das Relationselement eindeutig in d repräsentiert. Weiterhin gilt, dass $R_d \subseteq K_d \times K_d$ ist und jedes $r_i = (k_s, k_z) \in R_d$ eine Relation zwischen zwei Komponenten k_s und k_z beschreibt. Wobei k_s die Startkomponente und k_z die Zielkomponente der Relation r_i repräsentiert und $k_s \neq k_z$ gilt. ■

Komponentenelemente sowie Relationselemente sind typisiert. Mittels eindeutigen Typen kann so die Semantik der jeweiligen Komponenten und Relationen definiert werden. Beispielweise können hier Eigenschaften oder Schnittstellen vorgegeben werden. Diese Typen ermöglichen dadurch eine erhöhte und vereinfachte Wiederverwendbarkeit der Komponenten und Relationen. Die Komponente *Bestell-App* könnte beispielsweise den Komponententyp *PHP-Anwendung* besitzen und die Komponente *Bestellungen-DB* könnte den Komponententyp *MySQL-Datenbank* besitzen. Weiterhin können Typen zur Verfeinerung von anderen Typen erben. Der Komponententyp *PHP-Anwendung* kann zum Beispiel vom Komponententyp *Anwendung* erben.

Definition 4.4 (Komponententypelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $KT_d = \{kt_1, kt_2, ..., kt_n\}$ die Menge aller Komponententypelemente in d. Ein Komponententyp kt_i ist dabei eine wiederverwendbare Entität, die die Semantik der zu diesem Typ gehörigen Komponente(n) der Anwendung spezifiziert. ■

In TOSCA [OAS13a] werden beispielsweise für die oben genannten Beispiele der Relationen zwischen den Komponenten hierfür die konkreten Relationstypen *hostedOn*, *ConnectsTo* sowie *DependsOn* vorgeschlagen.

Definition 4.5 (Relationstypelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $RT_d = \{rt_1, rt_2, ..., rt_n\}$ die Menge aller Relationstypelemente in d. Ein Relationstyp rt_i ist dabei eine wiederverwendbare Entität, die die Semantik der zu diesem Typ gehörigen Relation(en) der Anwendung spezifiziert. ■

Komponenten können Anforderungen definieren, die für einen erfolgreichen Betrieb und damit für ein zulässiges Deployment erfüllt sein müssen. Solche Anforderungen können zum Beispiel benötigte Leistungen oder Funktionalitäten beschreiben, aber auch beispielsweise Vorgaben bezüglich des Datenschutzes machen. Die Komponente *Bestell-App* vom Typ *PHP-Anwendung* könnte beispielsweise die Anforderung *"benötige PHP Webserver"* haben, da dieser für den erfolgreichen Betrieb der Komponente benötigt wird.

Definition 4.6 (Anforderungselemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $A_d = \{a_1, a_2, ..., a_n\}$ die Menge aller Anforderungselemente in d. Jedes $a_i \in A_d$ repräsentiert dabei eine Anforderung einer Komponente der Anwendung. Es gilt $a_i = (Id, Name)$, wobei $Id \in \Sigma^+$ das Anforderungselement global eindeutig identifiziert und $Name \in \Sigma^+$ das Anforderungselement eindeutig in d repräsentiert. ■

Entsprechend den Anforderungen gibt es auch Fähigkeiten, die von den Komponenten angeboten beziehungsweise bereitgestellt werden können. Um ein erfolgreiches Deployment sowie den Betrieb einer Anwendung zu gewährleisten, müssen daher alle Anforderungen der einzelnen Komponenten von angebotenen Fähigkeiten der mit diesen Komponenten jeweils in Relation stehenden Komponenten erfüllt werden. Beispielsweise könnte die Komponente *PHP-Server* eine entsprechende Fähigkeit besitzen, um die oben genannte offene Anforderung der Komponente *Bestell-App* zu erfüllen.

Definition 4.7 (Fähigkeitselemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $F_d = \{f_1, f_2, ..., f_n\}$ die Menge aller Fähigkeitselemente in d. Jedes $f_i \in F_d$ repräsentiert dabei eine Fähigkeit einer Komponente der Anwendung. Es gilt $f_i = (Id, Name)$, wobei $Id \in \Sigma^+$ das Fähigkeitselement global eindeutig identifiziert und $Name \in \Sigma^+$ das Fähigkeitselement eindeutig in d repräsentiert. ■

Anforderungen und Fähigkeiten können unter anderem dazu verwendet werden, unvollständige Anwendungstopologien automatisiert zu vervollständigen. Dazu werden für Komponenten mit offenen Anforderungen, Komponenten mit passenden Fähigkeiten gesucht, welche diese offenen Anforderungen erfüllen, und der Anwendungstopologie hinzugefügt. Entsprechend den Komponenten und Relationen stammen auch die Anforderungen und Fähigkeiten, zur Erhöhung der Wiederverwendbarkeit, jeweils von abstrakten Typen ab, welche zur Definition der Semantik genutzt werden können.

Definition 4.8 (Fähigkeitstypelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $FT_d = \{ft_1, ft_2, ..., ft_n\}$ die Menge aller Fähigkeitstypelemente in d. Wobei $ft_i \in \Sigma^+$ gilt und eindeutig in d ist. ■

Definition 4.9 (Anforderungstypelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $AT_d = \{at_1, at_2, ..., at_n\}$ die Menge aller Anforderungstypelemente in d, wobei $at_i = (Name, benötigteFähigkeit)$ gilt. Dabei repräsentiert $Name \in \Sigma^+$ das Anforderungstypelement eindeutig in d und $benötigteFähigkeit \in FT_d$ definiert ein Fähigkeitstypelement, das sowohl selbst sowie dessen Supertypen dieses Anforderungstypelement erfüllen. ■

Um die folgenden Definitionen sowie die Verwendung der bisherigen Definitionen zu vereinfachen, kann die vereinigte Menge aus Komponentenelementen, Relationselementen, Anforderungselementen sowie Fähigkeitselementen zu Modellelementen zusammengefasst und wie folgt definiert werden.

Definition 4.10 (Modellelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $ME_d = K_d \cup R_d \cup A_d \cup F_d$. ■

Entsprechend kann zur vereinfachten Verwendung der Definitionen auch die vereinigte Menge aus Komponententypelementen, Relationstypelementen, Anforderungstypelementen sowie Fähigkeitstypelementen zu Modellelementtypen zusammengefasst und wie folgt definiert werden.

Definition 4.11 (Modellelementtypen)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $MET_d = KT_d \cup RT_d \cup AT_d \cup FT_d$. ■

Um den Ist-Zustand, den gewünschten Soll-Zustand oder die Konfiguration einer Komponente oder Relation zu beschreiben, können Eigenschaftselemente genutzt werden. Für eine bereits bestehende Komponente, wie beispielsweise eine vorhandene und laufende virtuelle Maschine, können so zum Beispiel deren IP-Adresse sowie zur Anmeldung benötigte Sicherheitsmerkmale, wie Nutzername und Passwort, definiert werden. Auch der Datenfaktor (vgl. Definition 3.1) einer datenverarbeitenden Komponente kann mittels der Eigenschaftselemente angegeben werden. Ebenso kann auch zum Beispiel für eine noch bereitzustellende Komponente, wie beispielsweise die Komponente *Bestell-App*, der Port, über den diese Komponente nach erfolgreicher Bereitstellung erreichbar sein soll, konfiguriert werden.

Weiterhin können definierte Anforderungen sowie Fähigkeiten entsprechend mittels Eigenschaften verfeinert und individualisiert werden.

Definition 4.12 (Eigenschaftselemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $E_d = \{e_1, e_2, ..., e_n\}$ die Menge aller Eigenschaftselemente in d, wobei $e_i = (Name, Wert)$ gilt. *Name* repräsentiert dabei den Namen der Eigenschaft und *Wert* repräsentiert den Wert der Eigenschaft. Weiterhin gilt sowohl $Name \in \Sigma^+$ als auch $Wert \in \Sigma^+ \cup \{\emptyset\}$. Falls $Wert = \emptyset$ gilt, ist der Wert der Eigenschaft noch undefiniert. ■

Zur Verwaltung beziehungsweise zum Management der Komponenten und Relationen eines D^{3}Ms werden verschiedene Operationen genutzt, die von den jeweiligen zu verwaltenden Komponenten beziehungsweise Relationen angeboten werden. Die Komponente *Bestell-App* könnte beispielsweise eine Operation *install* zum Installieren der Komponente, eine Operation *configure* zur Konfiguration der Komponente, zum Beispiel auf Basis der ihr vorgegebenen Eigenschaften, sowie eine Operation *start* zum Starten der Komponente anbieten. Diese Operationen werden Managementoperationen genannt und zunächst informell eingeführt.

Definition 4.13 (Managementoperation - informell)
Eine Managementoperation ist eine ausführbare Prozedur die zum Management einer Komponente oder Relation einer Anwendung ausgeführt wird.■

Zur Realisierung dieser Operationen können beliebige Technologien, wie beispielsweise komplexe Deployment-Automatisierungstechnologien (vgl. Abschnitt 2.4) oder Skripte, genutzt werden. Neben Managementoperationen gibt es auch Anwendungsoperationen, welche die eigentliche Funktionalität einer Komponente nach außen hin zur Verfügung stellen. So könnte die Komponente *Bestell-App* beispielsweise eine Operation *order* zum Bestellen eines Produkts zur Nutzung anbieten oder die Komponente *Bestellungen-DB* eine Operation *storeOrder* zum Abspeichern dieser Bestellungen anbieten. Wie Managementoperationen können auch Anwendungsoperationen mit beliebigen Technologien, zum Beispiel Python oder Java, realisiert werden.

Definition 4.14 (Anwendungsoperation - informell)
Eine Anwendungsoperation ist eine ausführbare Prozedur die zur direkten Verwendung der Geschäftslogik einer Komponente aufgerufen wird. ■

Aus formaler Sicht werden sowohl Managementoperationen als auch Anwendungsoperationen identisch beschrieben. Daher können sie zusammengefasst, als allgemeine Operationselemente, wie folgt formal definiert werden.

Definition 4.15 (Operationselemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $O_d = \{o_1, o_2, ..., o_n\}$ die Menge aller Operationselemente in d, wobei $o_i = (Name, EP_o, AP_o)$ gilt. $Name \in \Sigma^+$ repräsentiert dabei den in d eindeutigen Namen der Operation. Weiterhin repräsentiert $EP_o \subseteq EP_d$ die Menge an Eingabeparametern (siehe Definition 4.16) und $AP_o \subseteq AP_d$ die Menge an Ausgabeparametern (siehe Definition 4.17) der Operation. Eine Operation o_i ist dabei eine ausführbare Prozedur, die entweder (i) zum Management einer Komponente oder Relation oder (ii) zur Nutzung einer Komponente der Anwendung verwendet werden kann.■

Zur erfolgreichen Ausführung der Operationen werden häufig bestimmte Informationen benötigt, die typischerweise im Vorfeld, also während der Modellierung der Anwendungstopologie, noch nicht bekannt waren. Solche Daten, typischerweise Instanzinformationen von Komponenten der bereitzustellenden Anwendung, können dann mittels Eingabeparameter der Operation zur Ausführung übergeben werden. Der Komponente *Bestell-App* müssen beispielsweise die zur Verbindung benötigten Daten, wie IP-Adresse, Port und Zugangsdaten, der Komponente *Bestellungen-DB* übergeben werden. Weiterhin könnte auch der zuvor in den Eigenschaften der Komponente definierte Port als ein solcher Eingabeparameter einer Managementoperation genutzt werden. Im Nachfolgenden werden daher die in Operationselementen genutzten Eingabeparameterelemente formal definiert.

Definition 4.16 (Eingabeparameterelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $EP_d = \{ep_1, ep_2, ..., ep_n\}$ die Menge aller Eingabeparameterelemente in d, wobei $ep_i = (Name, Typ)$ gilt. $Name \in \Sigma^+$

repräsentiert dabei den Namen des Eingabeparameters und $Typ \in \Sigma^+$ repräsentiert den Typ des Eingabeparameters. ■

Weiterhin können die Ausgabeparameter einer Operation genutzt werden, um solche Instanzinformationen bereitstellen zu können. Eine Operation *createVM*, zum Erstellen einer virtuellen Maschine, würde beispielsweise typischerweise einen Ausgabeparameter *VM-IP* definieren, um die erst nach der erfolgreichen Erstellung bekannte IP-Adresse der virtuellen Maschine zurückgeben zu können und in der entsprechend definierten Eigenschaft der Komponente als Instanzinformation ablegen zu können. Nachfolgend werden dementsprechend Ausgabeparameterelemente formal definiert.

Definition 4.17 (Ausgabeparameterelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $AP_d = \{ap_1, ap_2, ..., ap_n\}$ die Menge aller Ausgabeparameterelemente in d, wobei $ap_i = (Name, Typ)$ gilt. $Name \in \Sigma^+$ repräsentiert dabei den Namen des Ausgabeparameters und $Typ \in \Sigma^+$ repräsentiert den Typ des Ausgabeparameters. ■

Eingabeparameterelemente sowie Ausgabeparameterelemente können zur Vereinfachung als Parameterelemente zusammengefasst werden.

Definition 4.18 (Parameterelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $P_d = \{p_1, p_2, ..., p_n\}$ die Menge aller Parameterelemente in d. Ein Parameterelement $p_i \in P_d$ kann dabei (i) ein Eingabeparameterelement oder (ii) ein Ausgabeparameterelement sein. Dementsprechend gilt $P_d = EP_d \cup AP_d$. ■

Die jeweiligen Operationen einer Komponente oder Relation werden in Schnittstellenelementen gesammelt und gruppiert. Dabei wird zwischen Schnittstellenelementen speziell für Managementoperationen sowie für Anwendungsoperationen unterschieden. Beide Arten können allerdings als allgemeine Schnittstellenelemente zusammengefasst und definiert werden.

Definition 4.19 (Schnittstellenelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $S_d = \{s_1, s_2, ..., s_n\}$ die Menge aller Schnittstellenelemente in d. Ein Schnittstellenelement $s_i \in S_d$ kann dabei (i) ein Managementschnittstellenelement oder (ii) ein Anwendungsschnittstellenelement sein. Es gilt $S_d = MS_d \cup AS_d$, wobei MS_d die Menge aller Managementschnittstellenelemente (siehe Definition 4.20) und AS_d die Menge aller Anwendungsschnittstellenelemente (siehe Definition 4.21) in d bezeichnet.■

Managementoperationen können sowohl für Komponenten als auch Relationen definiert werden. Dementsprechend können Managementschnittstellenelemente ebenso für Komponenten als auch Relationen definiert werden.

Definition 4.20 (Managementschnittstellenelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $MS_d = \{ms_1, ms_2, ..., ms_n\}$ die Menge aller Managementschnittstellenelemente in d, wobei $ms_i = (Name, O_m)$ gilt. $Name \in \Sigma^+$ repräsentiert den Namen einer konkreten Managementschnittstelle und $O_m \subseteq O_d$ repräsentiert eine Menge an Managementoperationen, die zum Management einer Komponente oder Relation angeboten werden.■

Anwendungsoperationen können im Gegensatz zu Managementoperationen jedoch nur für Komponenten definiert werden, da sie die angebotene Geschäftslogik dieser Komponenten widerspiegeln. Daher können Anwendungsschnittstellenelemente ebenfalls nur für Komponenten definiert werden.

Definition 4.21 (Anwendungsschnittstellenelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $AS_d = \{as_1, as_2, ..., as_n\}$ die Menge aller Anwendungsschnittstellenelemente in d, wobei $as_i = (Name, O_a)$ gilt. $Name \in \Sigma^+$ repräsentiert dabei den Namen einer konkreten Anwendungsschnittstelle und $O_a \subseteq O_d$ repräsentiert eine Menge an Anwendungsoperation aus d, die zur Nutzung einer Komponente der Anwendung angeboten werden. ■

Die verschiedenen Operationen und damit auch die Schnittstellen der Komponenten und Relationen einer Anwendung werden durch Artefakte realisiert.

Da auch hier zwischen Artefakten zur Umsetzung von Managementoperationen und Anwendungsoperationen unterschieden werden muss, wird zunächst die vereinigte Menge dieser beiden Artefaktarten definiert.

Definition 4.22 (Artefaktelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $AE_d = \{ae_1, ae_2, ..., ae_n\}$ die Menge aller Artefaktelemente in d. Ein Artefaktelement $ae_i \in AE_d$ kann dabei (i) ein Managementartefaktelement oder (ii) ein Anwendungsartefaktelement sein. Dementsprechend gilt $AE_d = MA_d \cup AA_d$, wobei MA_d die Menge aller Managementartefaktelemente (siehe Definition 4.23) und AA_d die Menge aller Anwendungsartefaktelemente (siehe Definition 4.24) in d bezeichnet. ■

Die konkreten ausführbaren Dateien, welche die Managementschnittstellen sowie deren einzelne Managementoperationen zur Verwaltung der Komponenten und Relationen einer Anwendung unter Verwendung beliebiger Technologien implementieren, werden mithilfe von Managementartefaktelementen definiert. So könnte die von der Komponente *Bestell-App* angebotene Managementoperation *install*, zum Installieren der Komponente, beispielsweise mittels eines Skripts *install.sh* implementiert sein.

Definition 4.23 (Managementartefaktelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $MA_d = \{ma_1, ma_2, ..., ma_n\}$ die Menge aller Managementartefaktelemente in d, wobei $ma_i = (Name, Typ, Referenz)$ gilt. $Name \in \Sigma^+$ repräsentiert dabei den Namen des Managementartefakts und $Typ \in \Sigma^+$ beschreibt den Typ des Managementartefakts. Weiterhin zeigt $Referenz \in \Sigma^+$ mithilfe einer URI auf eine konkrete Datei, welche das Managementartefakt realisiert. Dabei kann es sich um eine relative URI handeln, falls das Artefakt lokal vorhanden ist, oder um eine absolute URI, falls das Artefakt von einer entfernten Stelle bezogen werden muss. ■

Die Komponenten selbst, beziehungsweise deren Geschäftslogik und allgemeine Funktionalität, werden mithilfe von Anwendungsartefakten realisiert. Diese stellen somit typischerweise die ausführbaren Softwareteile zur Implementierung ihrer repräsentierenden Komponente dar. Im Falle der

Bestell-App vom Typ *PHP-Anwendung* würde ein solches Anwendungsartefakt beispielsweise alle zum Betrieb der Komponente benötigten Dateien, also etwa PHP-Dateien, Konfigurationsdateien sowie Grafiken, beinhalten. Ein Anwendungsartefakt kann jedoch auch nicht ausführbare Daten, welche die entsprechende Komponente darstellen, beinhalten. Eine Komponente *Machine-Data* könnte zum Beispiel Rohdaten von Sensoren oder Maschinenlogs als Anwendungsartefakt beinhalten, die dann beispielsweise während der Ausführung der Anwendung bearbeitet oder analysiert werden sollen.

Definition 4.24 (Anwendungsartefaktelemente)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $AA_d = \{aa_1, aa_2, ..., aa_n\}$ die Menge aller Anwendungsartefaktelemente in d, wobei $aa_i = (Name, Typ, Referenz)$ gilt. $Name \in \Sigma^+$ repräsentiert dabei den Namen des Anwendungsartefakts und $Typ \in \Sigma^+$ beschreibt den Typ des Anwendungsartefakts. Weiterhin zeigt $Referenz \in \Sigma^+$ mithilfe einer URI auf eine konkrete Datei, welche das Anwendungsartefakt realisiert. Dabei kann es sich um eine relative URI handeln, falls das Artefakt lokal vorhanden ist, oder um eine absolute URI, falls das Artefakt von einer entfernten Stelle bezogen werden muss. Anwendungsartefakte realisieren die Komponenten einer Anwendung und implementieren die angebotenen Anwendungsoperationen einer Komponente. ■

Da ein Artefakt, welches als Skript realisiert ist, möglicherweise anders behandelt und verarbeitet werden muss als ein Artefakt, welches als Java Anwendung realisiert ist oder Rohdaten enthält, kann für die einzelnen Artefakte jeweils ein entsprechender Typ definiert werden. Bei der Verarbeitung der Artefakte kann dann mithilfe der Typen eine einheitliche und passende Verarbeitung der jeweiligen Artefakte vorgegeben und sichergestellt werden.

Nachdem die einzelnen Elemente eines D^{3}M eingeführt, vorgestellt und definiert wurden, werden im Folgenden die nötigen Abbildungen zur Definition eines D^{3}Ms beschrieben und formal definiert. Zur Abbildung der Modellelemente auf ihren jeweiligen Modellelementtyp, also beispielsweise, dass die Komponente *Bestell-App* vom Komponententyp *PHP-Anwendung* abstammt, wird die Abbildung typ_d genutzt, die wie folgt definiert ist.

Definition 4.25 (Abbildung: typ_d)
Sei $d \in D$ ein D^{3}M, dann bezeichnet typ_d die Abbildung, die jedem Modellelement in d, jeweils dessen Modellelementtypen zuordnet. Somit gilt:

$$
\begin{aligned}
typ_d : ME_d &\rightarrow MET_d \\
me_d &\mapsto met_d \\
\text{mit } met_d &\in KT_d \text{ für } me_d \in K_d \\
\text{und } met_d &\in RT_d \text{ für } me_d \in R_d \\
\text{und } met_d &\in AT_d \text{ für } me_d \in A_d \\
\text{und } met_d &\in FT_d \text{ für } me_d \in F_d
\end{aligned}
$$

■

Weiterhin kann ein Modellelementtyp selbst auch wieder von einem anderen Modellelementtyp abstammen. Um dies darzustellen, wird die nachfolgende Abbildung $supertyp_d$ definiert. Dadurch kann eine Vererbungshierarchie aufgebaut werden, die es erlaubt, die Semantik für jede einzelne Vererbungsschicht individualisieren und verfeinern zu können. Dabei werden beispielsweise unter anderem definierte Eigenschaften, Anforderungen, Fähigkeiten sowie Schnittstellen vererbt. Zum Beispiel kann der Komponententyp *PHP-Anwendung*, der vom Komponententyp *Anwendung* abstammt, somit speziell für PHP-basierte Anwendungen angepasst beziehungsweise erweitert werden. Zu beachten gilt allerdings, dass ein Modellelementtyp nicht von sich selbst abstammen darf und die Vererbungshierarchie eines Modellelementtyps somit immer azyklisch sein muss.

Definition 4.26 (Abbildung: $supertyp_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $supertyp_d$ die Abbildung, die jedem Modellelementtypen in d, jeweils dessen Superelementtyp zuordnet. Dadurch kann die Semantik von einem Modellelementtyp zu einem anderen Modellelementtyp vererbt werden. Dementsprechend gilt:

$$
\begin{aligned}
supertyp_d : MET_d &\rightarrow MET_d \cup \{\perp\} \\
met_t &\mapsto met_s
\end{aligned}
$$

Dabei gilt $met_t \neq met_s$ sowie $met_s = \perp$, falls ein Modellelementtyp kein Superelementtyp besitzt. ■

Um den vorhandenen Komponententypelementen beziehungsweise Komponentenelementen in einem D^{3}M jeweils deren Anforderungen zuzuweisen, wird die Abbildung $anforderungen_d$ genutzt, die wie folgt formal definiert ist. Damit kann beispielsweise dem Komponententyp *PHP-Anwendung* die Anforderung *"benötige PHP Webserver"* zugewiesen werden.

Definition 4.27 (Abbildung: $anforderungen_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $anforderungen_d$ die Abbildung, die jedem Komponententypelement bzw. Komponentenelement in d, jeweils dessen Anforderungstypelemente bzw. Anforderungselemente zuordnet:

$$\begin{aligned} anforderungen_d : KT_d \cup K_d &\rightarrow \mathcal{P}(AT_d) \cup \mathcal{P}(A_d) \\ x &\mapsto y \\ \text{mit } y \in \mathcal{P}(AT_d) &\text{ für } x \in KT_d \\ \text{und } y \in \mathcal{P}(A_d) &\text{ für } x \in K_d \end{aligned}$$ ■

Dementsprechend wird die Abbildung $fähigkeiten_d$ genutzt, um den Komponententypelementen beziehungsweise Komponentenelementen jeweils deren Fähigkeiten zuzuweisen. Die Abbildung ist wie folgt definiert.

Definition 4.28 (Abbildung: $fähigkeiten_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $fähigkeiten_d$ die Abbildung, die jedem Komponententypelement bzw. Komponentenelement in d, jeweils dessen Fähigkeitstypelemente bzw. Fähigkeitselemente zuordnet:

$$\begin{aligned} anforderungen_d : KT_d \cup K_d &\rightarrow \mathcal{P}(FT_d) \cup \mathcal{P}(F_d) \\ x &\mapsto y \\ \text{mit } y \in \mathcal{P}(FT_d) &\text{ für } x \in KT_d \\ \text{und } y \in \mathcal{P}(F_d) &\text{ für } x \in K_d \end{aligned}$$ ■

Um den Modellelementtypen sowie Modellelementen eines D^{3}Ms jeweils Eigenschaften zuzuordnen, wird die Abbildung $eigenschaften_d$ genutzt.

Definition 4.29 (Abbildung: $eigenschaften_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $eigenschaften_d$ die Abbildung, die jedem Komponentenelement, Komponententypelement, Relationselement, Relationstypelement, Anforderungselement, Anforderungstypelement, Fähigkeitselement sowie Fähigkeitstypelement in d, jeweils dessen Eigenschaftselemente zuordnet:

$$\begin{aligned} eigenschaften_d : M_d &\to \mathcal{P}(E_d), \text{ wobei } M_d = MET_d \cup ME_d \\ m_d &\mapsto p_d \end{aligned}$$

Wobei Eigenschaften dabei immer eindeutig sein müssen:

$$\begin{aligned} &\forall m_d \in M_d \forall (n_1, w_1), (n_2, w_2) \in eigenschaften_d(m_d) : \\ &\qquad n_1 = n_2 \implies w_1 = w_2 \end{aligned}$$ ■

Um Operationselementen in einem D^{3}M jeweils deren Eingabeparameter sowie Ausgabeparameter zuzuweisen, wird die Abbildung $parameter_d$ genutzt.

Definition 4.30 (Abbildung: $parameter_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $parameter_d$ die Abbildung, die jedem Operationselement in d, jeweils dessen Eingabeparameterelemente sowie Ausgabeparameterelemente zuordnet:

$$\begin{aligned} parameter_d : O_d &\to \mathcal{P}(EP_d) \times \mathcal{P}(AP_d) \\ o_d &\mapsto (ep_o, ap_o) \end{aligned}$$

Die Parameter müssen dabei innerhalb einer Operation eindeutig sein:

$$\begin{aligned} &\forall o_d \in O_d \forall (n_1, t_1), (n_2, t_2) \in parameter_d(o_d) : \\ &(n_1, t_1), (n_2, t_2) \in EP_d \wedge n_1 = n_2 \implies t_1 = t_2 \\ &(n_1, t_1), (n_2, t_2) \in AP_d \wedge n_1 = n_2 \implies t_1 = t_2 \end{aligned}$$ ■

Um den Schnittstellenelementen jeweils deren Operationen zuzuweisen, wird die Abbildung $operationen_d$ genutzt, welche wie folgt definiert ist.

Definition 4.31 (Abbildung: $operationen_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $operationen_d$ die Abbildung, die jedem Managementschnittstellenelement sowie Anwendungsschnittstellenelement in d, jeweils dessen Operationselemente zuordnet:

$$operationen_d : S_d \rightarrow \mathcal{P}(O_d), \text{ wobei } S_d = MS_d \cup AS_d$$
$$s_d \mapsto p_d$$ ■

Da bei den Schnittstellenelementen zwischen Managementschnittstellenelementen und Anwendungsschnittstellenelementen unterschieden werden muss, werden zwei verschiedene Abbildungen hierfür benötigt. Um den Komponentenelementen, Komponententypelementen, Relationselementen sowie Relationstypelementen in einem D^{3}M jeweils deren Managementschnittstellen zuzuweisen, wird die Abbildung $mSchnittstellen_d$ genutzt.

Definition 4.32 (Abbildung: $mSchnittstellen_d$)
Sei $d \in D$ ein D^{3}M, dann ordnet die Abbildung $mSchnittstellen_d$ jedem Komponentenelement, Komponententypelement, Relationselement und Relationstypelement in d, dessen Managementschnittstellenelemente zu:

$$mSchnittstellen_d : KR_d \rightarrow \mathcal{P}(MS_d), \text{ wobei } KR_d = K_d \cup R_d \cup KT_d \cup RT_d$$
$$kr_d \mapsto p_d$$ ■

Um dagegen den Komponentenelementen sowie Komponententypelementen in einem D^{3}M jeweils deren Anwendungsschnittstellenelemente zuzuweisen, wird die Abbildung $aSchnittstellen_d$ genutzt.

Definition 4.33 (Abbildung: $aSchnittstellen_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $aSchnittstellen_d$ die Abbildung, die jedem Komponentenelement sowie Komponententypelement in d, jeweils dessen

Anwendungsschnittstellenelemente zuordnet:

$$aSchnittstellen_d : KK_d \rightarrow \mathcal{P}(AS_d), \text{ wobei } KK_d = KT_d \cup K_d$$
$$kk_d \mapsto p_d$$ ■

Ebenso muss für Artefaktelemente zwischen Managementartefaktelementen und Anwendungsartefaktelementen unterschieden werden. Um den Komponentenelementen, Relationselementen sowie Operationselementen in einem D^{3}M jeweils deren Managementartefaktelemente zuzuweisen, wird die Abbildung $mArtefakte_d$ genutzt, welche wie folgt formal definiert ist.

Definition 4.34 (Abbildung: $mArtefakte_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $mArtefakte_d$ die Abbildung, die jedem Komponentenelement, Relationselement sowie Operationselement in d, jeweils dessen Managementartefaktelemente zuordnet:

$$mArtefakte_d : KRO_d \rightarrow \mathcal{P}(MA_d), \text{ wobei } KRO_d = K_d \cup R_d \cup O_d$$
$$kro_d \mapsto p_d$$ ■

Um dagegen den Komponentenelementen sowie Operationselementen in einem D^{3}M jeweils deren Anwendungsartefaktelemente zuzuweisen, wird die Abbildung $aArtefakte_d$ genutzt, welche formal wie folgt definiert ist.

Definition 4.35 (Abbildung: $aArtefakte_d$)
Sei $d \in D$ ein D^{3}M, dann bezeichnet $aArtefakte_d$ die Abbildung, die jedem Komponentenelement sowie Operationselement in d, jeweils dessen Anwendungsartefaktelemente zuordnet:

$$aArtefakte_d : KO_d \rightarrow \mathcal{P}(AA_d), \text{ wobei } KO_d = K_d \cup O_d$$
$$ko_d \mapsto p_d$$ ■

4.2 Deployment-Modellierungskonzepte

Die zunehmende Menge an gesammelten Daten, zum Beispiel Sensordaten in den Bereichen Internet of Things und Industrie 4.0, ermöglichen umfassende Datenanalysen, die wertschöpfende Möglichkeiten erlauben, wie beispielsweise die vorausschauende Wartung in Fertigungssystemen (vgl. Abschnitt 2.1). Allerdings erfordert die Analyse der gesammelten beziehungsweise abrufbaren Daten in der Regel komplexe und domänenspezifische Algorithmen sowie Anwendungen, die zum Teil nicht von einem Unternehmen oder der entsprechenden Abteilung selbst entwickelt werden können. Daher wird diese Arbeit häufig an, beispielsweise auf maschinelles Lernen spezialisierte, externe Dienstleister oder eine separate Abteilung abgegeben. Eine solche Zusammenarbeit kann jedoch komplex werden, da die zu analysierenden Daten für das Unternehmen typischerweise von entscheidender Bedeutung sind und oft nicht an Dritte weitergegeben werden dürfen [SWW15; ZR13]. Infolgedessen wird die datenverarbeitende Anwendung häufig in einer anderen Infrastruktur entwickelt, als sie dann später bereitgestellt und ausgeführt wird [ZBF+17b].

Die Installation, Konfiguration und Ausführung komplexer Anwendungen sind für die verantwortlichen Administratoren und Entwickler allerdings eine erhebliche Herausforderung, da detaillierte Fachkenntnisse über die bereitzustellende Anwendung selbst als auch über die Zielumgebung erforderlich sind: (i) die von der Anwendung benötigte Middleware und weitere Abhängigkeiten müssen bereitgestellt werden, (ii) die Anwendung beziehungsweise ihre Anwendungskomponenten selbst müssen bereitgestellt und (iii) entsprechend der Zielumgebung konfiguriert werden, um die erforderlichen Daten erhalten und verarbeiten zu können. Vor allem bei der Bereitstellung komplexer Anwendungen in heterogenen IT-Umgebungen, mit unterschiedlicher physischer oder virtueller Hardware, Anwendungsplattformen sowie Virtualisierungstechnologien kann dies zu Problemen führen, insbesondere falls die Bereitstellung manuell durchgeführt werden soll (vgl. Abschnitt 2.4). Dementsprechend ist die Bereitstellung solcher

Anwendungen in einer fremden IT-Infrastruktur eine komplexe Aufgabe, die typischerweise sowohl technische als auch organisatorische Schwierigkeiten mit sich bringt [BBK+13a; EKK+06].

Nachfolgend werden daher drei deklarative Modellierungsansätze auf Basis des D^{3}M-Metamodells vorgestellt, die es ermöglichen, datenverarbeitende Anwendungen so zu beschreiben, dass sowohl (i) ihre automatisierte Bereitstellung und Ausführung in einer entfernten und fremden IT-Infrastruktur als auch (ii) die Verknüpfung der Anwendungskomponenten mit den dort vorhandenen Daten ermöglicht wird. Die verschiedenen Modellierungsansätze unterscheiden sich dabei unter anderem hinsichtlich der jeweils über die Anwendung sowie die Zielumgebung benötigten Informationen sowie der sich ergebenden Komplexität der daraus resultierenden Modelle.

In Abschnitt 4.2.1 werden zuerst *Vollständige Anwendungstopologien* vorgestellt, die alle Informationen zur Bereitstellung und Konfigurationen einer Anwendung enthalten. Danach werden in Abschnitt 4.2.2 *Konfigurierbare Anwendungstopologien* vorgestellt, welche die grundsätzlichen Informationen der Anwendung enthalten, aber bezüglich der Zielumgebung frei konfigurierbar sind. In Abschnitt 4.2.3 werden anschließend *Variable Anwendungstopologien* vorgestellt, welche nur die minimal benötigten Informationen einer Anwendung enthalten. Abschließend werden in Abschnitt 4.2.4 diese drei Modellierungskonzepte verglichen und deren Vorteile sowie Nachteile bezüglich ihrer Nutzung diskutiert.

4.2.1 Vollständige Anwendungstopologien

Die Grundidee dieser Anwendungstopologien besteht darin, jedes Detail der Bereitstellung im Modell zu spezifizieren, sodass die Anwendungstopologie nicht parametrisiert werden muss. Für ein deklaratives Bereitstellungsmodell bedeutet dies, dass (i) jede Komponente, (ii) jede Beziehung zwischen den Komponenten sowie (iii) jede Eigenschaft von Komponenten und Beziehungen in der Anwendungstopologie genau spezifiziert sind. Wenn die auszuliefernde beziehungsweise bereitzustellende Funktionalität beispielsweise

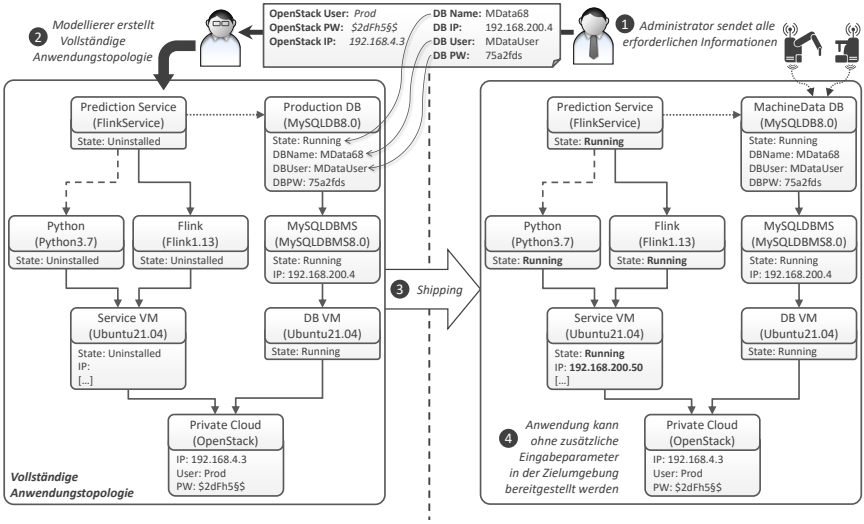


Abbildung 4.2: Beispiel einer Vollständigen Anwendungstopologie

per Python implementiert ist, zur Ausführung eine Flink sowie Python Laufzeitumgebung benötigt wird, die auf einem Ubuntu-Betriebssystem laufen, das auf einer Cloud-Management-Plattform wie OpenStack gehostet wird, müssen alle diese Details im deklarativen Modell angegeben werden. Das bedeutet, dass zum Beispiel die genaue Version des Ubuntu Betriebssystems sowie die IP-Adresse, der Benutzername und das Passwort von OpenStack angegeben werden müssen, um die Instanziierung der benötigten virtuellen Maschine zu ermöglichen. Darüber hinaus muss die Anwendungstopologie die Daten, die von der Anwendung verarbeitet werden sollen, genau spezifizieren. Beispielsweise muss die Anwendungstopologie eine Datenbankkomponente mit allen Eigenschaften enthalten, die zum Abrufen von Daten aus dieser Datenbank erforderlich sind, also beispielsweise IP-Adresse, Benutzername und Passwort zum Zugriff auf die Datenbank sowie den Namen der Tabelle, welche die zu verarbeitenden Daten enthält.

In Abbildung 4.2 ist ein solches D^3M dargestellt, das diese Merkmale einer Vollständigen Anwendungstopologie erfüllt. Die dargestellte Vollständige

Anwendungstopologie zeigt auf der linken Seite die Komponenten *Prediction Service*, zur Auswertung und Verarbeitung der Daten, die *Python* sowie *Flink* Laufzeitumgebung, die *Service VM* sowie die *Private Cloud* Komponente. Auf der rechten Seite sind weiterhin die beiden Komponenten *Production DB* sowie *MySQLDBMS* zu sehen. In der Anwendungstopologie sind weiterhin die Relationen *hostedOn*, als durchgestrichene Linie, *dependsOn*, als gestrichelte Linie, sowie *connectsTo*, als gepunktete Linie, dargestellt. Ebenfalls sind einige beispielhafte Eigenschaften der unterschiedlichen Komponenten, zum Beispiel Nutzernamen („User“) und Passwörter („PW“), abgebildet. Um Darzustellen, ob eine Komponente bereits in Betrieb ist oder noch bereitgestellt werden muss, wird die Eigenschaft *State* genutzt. So ist die Komponente *Prediction Service* als noch nicht installiert („Uninstalled“) und die Komponente *Production DB*, welche die Daten enthält und in der Zielumgebung bereits vorhanden ist („Running“), entsprechend gekennzeichnet.

Eine Vollständige Anwendungstopologie erfordert alle Informationen über die gewünschte Bereitstellung. Wenn also die Anwendung in eine entfernte Umgebung ausgeliefert werden soll, müssen alle benötigten Informationen im Modell enthalten sein. Zum Beispiel müssen die IP-Adresse, der Benutzername und das Passwort des OpenStacks den Entwicklern bekannt sein, damit sie ein solches Modell erstellen können. Das bedeutet allerdings, dass das Unternehmen, das diese externen Entwickler beauftragt, darauf vertrauen muss, dass diese Informationen nicht missbraucht werden. Diese Art der Modellierung von Anwendungstopologien eignet sich daher nur für vertrauenswürdige Kooperationen sowie interne Entwicklungen.

Durch die Verwendung einer Vollständigen Anwendungstopologie kann der Entwickler die Anwendung in einer völlig eigenständigen Weise ausliefern. Für die Bereitstellung und Ausführung ist anschließend kein weiteres Fachwissen erforderlich, da alle Informationen vollständig im Modell enthalten sind. Das Unternehmen, das diese Anwendung nutzen möchte, muss also nur das entsprechende Modell ausführen. Die Eingaben von Parametern oder weiteren Informationen sind nicht erforderlich. Allerdings ist eine Vollständige Anwendungstopologie eng an die Zielumgebung gekoppelt. Ändert sich

also beispielsweise der Benutzername oder das Passwort des verwendeten OpenStack-Accounts oder dessen IP-Adresse, muss die Anwendungstopologie angepasst werden. Während solche Eigenschaften noch einfach selbst in der Anwendungstopologie angepasst werden können, können bei anderen Änderungen, zum Beispiel beim Umzug der zu verarbeitenden Daten in eine andere Art von Datenbank, größere Anpassungen in der Anwendungstopologie beziehungsweise den einzelnen Artefakten nötig werden.

4.2.2 Konfigurierbare Anwendungstopologien

Die Grundidee dieser Anwendungstopologien besteht darin, die Bereitstellung aller Komponenten zu modellieren, ohne jedoch dabei alle für die Bereitstellung, Konfiguration und Verbindung der Komponenten erforderlichen Details spezifizieren zu müssen. Daraus ergibt sich eine parametrisierbare Anwendungstopologie, die dementsprechend konfigurierbar ist. Bei der Verwendung einer Konfigurierbaren Anwendungstopologie werden, ähnlich wie beim ersten Ansatz, (i) die Komponenten der Anwendung und (ii) die Beziehungen zwischen diesen Komponenten in der Anwendungstopologie definiert. Allerdings werden bei diesem Ansatz nicht alle Eigenschaften im Modell angegeben. Anstatt beispielsweise eine MySQL-Datenbank mit allen ihren Eigenschaften wie IP-Adresse, Benutzername und Kennwort zu modellieren, werden nur die Komponenten selbst modelliert, ohne dass die konkreten Eigenschaftswerte angegeben werden. Diese Eigenschaften werden bei der Bereitstellung der Anwendung entweder (i) von der Person, die die Bereitstellung initiiert, oder (ii) von der entsprechenden Laufzeitumgebung gefüllt. Da weiterhin nur die Informationen zur Verbindung mit der Datenbank benötigt werden, ist es außerdem nicht notwendig weitere Details zu definieren, beispielsweise auf welcher Version des Ubuntu Betriebssystems die Datenbank betrieben wird.

In Abbildung 4.3 ist, dem vorherigen Szenario folgend, das entsprechende Beispiel einer solchen Konfigurierbaren Anwendungstopologie abgebildet. Im Unterschied zu Vollständigen Anwendungstopologien sind hier jedoch

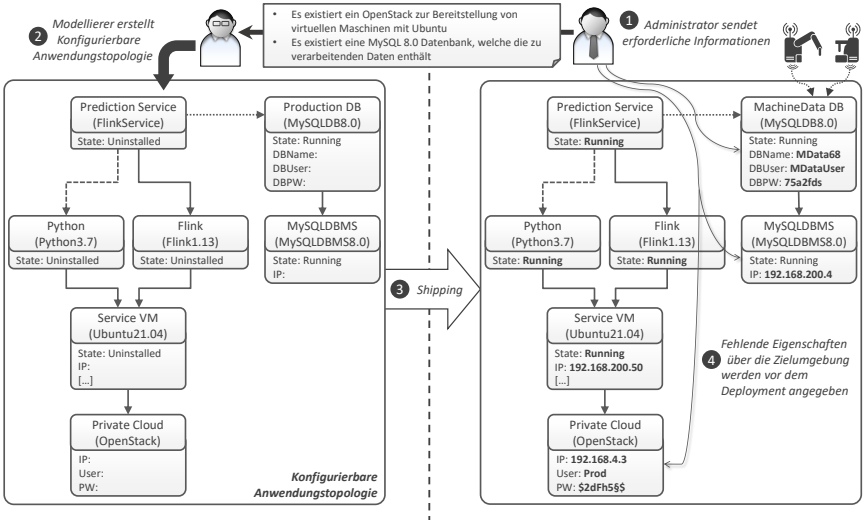


Abbildung 4.3: Beispiel einer Konfigurierbaren Anwendungstopologie

einige der definierten Eigenschaftsfelder offen gelassen worden. Dadurch wird gekennzeichnet, dass für die Bereitstellung der Anwendung in der Zielumgebung gegebenenfalls zusätzliche Informationen, wie zum Beispiel die IP-Adresse von OpenStack oder der Benutzername und das Passwort der MySQL-Datenbank, benötigt werden. Von der entsprechenden Laufzeitumgebung wird dann entweder dem Benutzer die Möglichkeit gegeben, die fehlenden Eigenschaftswerte selbst zu ergänzen, oder versucht, diese in der Zielumgebung automatisiert zu bestimmen.

Anstatt die Eigenschaftswerte leer zu lassen, können diese auch mittels „getInput()“ markiert werden, wie es unter anderem auch in TOSCA [OAS19] vorgeschlagen wird. In der dargestellten Konfigurierbaren Anwendungstopologie sind zum Beispiel aus Datenschutzgründen alle Eigenschaften der in der Zielumgebung bereits betriebenen Komponenten wie OpenStack oder die MySQL-Datenbank offen gelassen worden. So kann der Entwickler die Anwendungstopologie erstellen, obwohl die, für die eigentliche Bereitstellung der Anwendung, benötigten Parameter noch fehlen. So ist beispielsweise

die Information, auf welchem Betriebssystem die Datenbank gehostet wird, für die Bereitstellung des Analysestacks und dementsprechend für die Verbindung der Komponente *Prediction Service* mit der *MachineData DB* nicht erforderlich. Eine Konfigurierbare Anwendungstopologie benötigt somit nicht alle Informationen über das gewünschte und finale Deployment. Stattdessen muss es so gestaltet werden, dass es konfigurierbar und wiederverwendbar ist. So müssen beispielsweise nur die Komponenten der Anwendung, wie beispielsweise eine MySQL8.0-Datenbank, im entsprechendem Modell angegeben werden. Zusätzliche Informationen, wie zum Beispiel die IP-Adresse, der Benutzername und das Passwort, die für den Zugriff auf die Datenbank benötigt werden, werden erst während des Deployments festgelegt.

Bei diesem Ansatz müssen also, wie in Abbildung 4.3 dargestellt, grundsätzliche Informationen über die Infrastruktur in der Zielumgebung, beispielsweise das Vorhandensein von OpenStack, sowie über bereits laufende Komponenten, zum Beispiel der MySQL8.0-Datenbank, ausgetauscht werden, um vor allem externen Entwicklern die Erstellung einer solchen Konfigurierbaren Anwendungstopologie zu ermöglichen. Im Gegensatz zum ersten Ansatz, den Vollständigen Anwendungstopologien, sind hier jedoch keine Detail- oder Berechtigungsinformationen über die Zielumgebung erforderlich. Konfigurierbare Anwendungstopologien eignen sich daher dann, wenn grundlegende Informationen über die vorhandene Infrastruktur der Zielumgebung mit Dritten geteilt werden können, aber detaillierte Informationen, wie IP-Adressen, Benutzernamen und Passwörter, geheim gehalten werden sollen.

Dieser Ansatz ermöglicht die Erstellung von flexiblen und wiederverwendbaren Anwendungstopologien. Darüber hinaus sind für die Erstellung der Konfigurierbaren Anwendungstopologien keine Detail- oder Anmeldeinformationen über die Infrastruktur der Zielumgebung erforderlich. Dementsprechend müssen für die Erstellung von Konfigurierbaren Anwendungstopologien nur die verfügbaren Komponenten in der Zielumgebung bekannt sein. Dadurch wird eine nur lose an die Zielumgebung gekoppelte Anwendungstopologie erreicht, die nicht bei jeder kleinsten Änderung angepasst werden muss. Für die Modellierung und Nutzung dieser Anwendungstopologie macht es

beispielsweise keinen Unterschied, unter welcher IP-Adresse die Datenbank erreichbar ist oder wie die Anmeldeinformationen für das in der Zielumgebung verfügbare OpenStack lauten. Dies hat allerdings zur Folge, dass für die anschließende Bereitstellung und Ausführung in der Zielumgebung weiteres Fach- und Detailwissen erforderlich ist. Weiterhin muss auch hier bei größeren Änderungen, wie dem Umzug der Daten in eine andere Datenbank, die Anwendungstopologie gegebenenfalls entsprechend angepasst werden.

4.2.3 Variable Anwendungstopologien

Die Grundidee bei dieser Art von Anwendungstopologien ist, dass so wenig Informationen wie möglich benötigt werden, um eine Variable Anwendungstopologie zu modellieren. Beispielsweise werden bei diesem Ansatz nur die Geschäftslogik beziehungsweise Kernfunktionalität implementierenden Komponenten, die ausgeliefert werden sollen, sowie deren Anforderungen modelliert. Wenn beispielsweise ein datenverarbeitender Dienst auf Apache Flink gehostet werden muss, eine Abhängigkeit zu Python besteht und eine Verbindung zu einer MySQL-Datenbank erforderlich ist, müssen diese Anforderungen in der Anwendungstopologie angegeben werden. Diese werden verwendet, um in der Zielumgebung passende Komponenten zu finden, die den Anforderungen folgend entsprechende Fähigkeiten bieten.

In Abbildung 4.4 ist das bekannte Szenario wieder als D^3M dargestellt, welche die beschriebenen Merkmale erfüllt. Zur Realisierung von Variablen Anwendungstopologien enthält das D^3M-Metamodell die Möglichkeit zur Definition von Anforderungen und Fähigkeiten für Komponenten. Weiterhin können Anforderungstypen und Fähigkeitstypen als wiederverwendbare Entitäten modelliert werden. Mit Anforderungen und Fähigkeiten kann zum Beispiel definiert werden, dass eine Komponente eine bestimmte Funktionalität benötigt, das von einer anderen Komponente bereitgestellt wird.

Die in der Abbildung dargestellte Komponente *Prediction Service* gibt beispielsweise an, dass sie einen SQL-Endpunkt für den Zugriff auf die gespeicherten und zu verarbeitenden Daten erwartet, sowie außerdem eine

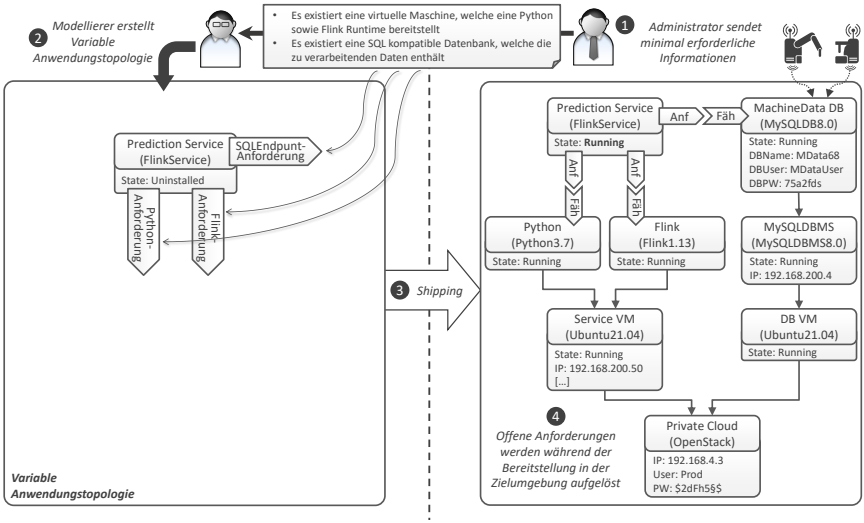


Abbildung 4.4: Beispiel einer Variablen Anwendungstopologie

Python sowie Flink Laufzeitumgebung benötigt. In der Zielumgebung müssen dann von der genutzten Laufzeitumgebung entsprechende Komponenten mit passenden Fähigkeiten gefunden werden, welche die definierten Anforderungen der bereitzustellenden Komponente erfüllen und damit die bisher unvollständige Anwendungstopologie vervollständigen.

Variable Anwendungstopologien enthalten lediglich die auszuliefernden Komponenten sowie Anforderungen an deren Bereitstellung. Es werden also keine detaillierten Informationen über die Zielinfrastruktur benötigt. Beispielsweise benötigt der Entwickler nur die Information, dass eine Apache Flink Laufzeitumgebung verfügbar ist, aber weder auf welchem Betriebssystem sie gehostet wird, noch ob sie auf einem lokalen OpenStack oder einem anderen Hypervisor oder Cloud-Anbieter läuft. Weiterhin ist beispielsweise die Definition der konkreten Version der in der Zielumgebung genutzten MySQL Datenbank zur Modellierung von Variablen Anwendungstopologien unerheblich. Es wird lediglich die Information benötigt, dass die Daten per SQL abgerufen werden können. Dieser Modellierungsansatz ist also vor allem

dann geeignet, wenn möglichst wenig Informationen geteilt werden sollen oder eine möglichst lose gekoppelte Anwendungstopologie gewünscht ist.

Aus Sicht des Entwicklers beziehungsweise Modellierers, können durch diesen Modellierungsansatz äußerst flexible und wiederverwendbare Anwendungstopologien erstellt werden, da er hierzu keine konkreten Informationen über die Zielumgebung benötigt. Weiterhin muss eine solche Anwendungstopologie ausschließlich bei grundlegenden Änderungen angepasst werden, beispielsweise falls die Daten nicht mehr per SQL abgerufen werden können. Auch wenn für diesen Modellierungsansatz nur einige grundlegenden Informationen ausgetauscht werden müssen, muss, damit die Anforderungen und Fähigkeiten übereinstimmen und in der Zielumgebung entsprechend zugeordnet werden können, die Definition der Anforderungen und Fähigkeiten auf einheitliche Weise erfolgen. Dementsprechend verteilt sich die Komplexität sowie das benötigte Fachwissen hier auf beide Seiten gleichermaßen.

4.2.4 Diskussion und Vergleich der Modellierungskonzepte

Um die Merkmale und Eigenheiten der drei vorgestellten Anwendungstopologien zu diskutieren, werden in diesem Abschnitt die verschiedenen Modellierungskonzepte hinsichtlich ihrer Vorteile sowie ihrer Auswirkungen auf das Deployment in Bezug auf Aspekte wie Zeitaufwand, Komplexität und die erforderlichen Fähigkeiten und Kenntnisse der für das Deployment verantwortlichen Personen verglichen.

Da bei Variablen Anwendungstopologien nicht der gesamte Anwendungsstack, sondern nur die auszuliefernde Komponente sowie deren Anforderungen modelliert werden müssen, ist die Variable Anwendungstopologie die schnellste Variante hinsichtlich des Zeitbedarfs für die Modellierung. Sowohl bei Vollständigen Anwendungstopologien als auch Konfigurierbaren Anwendungstopologien muss der gesamte Anwendungsstack modelliert werden, daher gibt es hier keinen großen Unterschied zwischen diesen beiden Alternativen hinsichtlich des Zeitbedarfs für die Modellierung. Allerdings enthält die Vollständige Anwendungstopologie alle erforderlichen Informa-

tionen für die Bereitstellung, sodass die modellierte Anwendung direkt in der Zielumgebung bereitgestellt werden kann, während bei der Konfigurierbaren Anwendungstopologie die erforderlichen und noch fehlenden Informationen, zum Beispiel IP-Adressen, Benutzernamen oder Passwörter, zunächst vom Entwickler oder Administrator eingegeben werden müssen. Im Falle der Variablen Anwendungstopologie müssen die Komponenten, die den spezifizierten Anforderungen entsprechen, von der Bereitstellungsumgebung ermittelt werden können, um die Topologie zu vervollständigen.

Was die Komplexität der verschiedenen vorgestellten Modellierungskonzepte betrifft, müssen bei Variablen Anwendungstopologien nur die Geschäftslogik beziehungsweise Kernfunktionalität implementierenden Komponenten sowie deren Anforderungen modelliert werden. Damit jedoch während der Bereitstellung die benötigten Komponenten ermittelt werden können, müssen die spezifizierten Anforderungen so definiert werden, dass sie mit den Fähigkeiten der in der Zielumgebung bereits vorhandenen Komponenten abgeglichen werden können. Um dieses Modellierungskonzept nutzen zu können, muss die Definition von Anforderungen und Fähigkeiten dementsprechend von allen beteiligten Parteien einheitlich gehandhabt werden. Zwar sind sowohl bei Vollständige Anwendungstopologien als auch bei Konfigurierbare Anwendungstopologien alle benötigten Komponenten bekannt, die Implementierung der Managementoperationen der Komponenten erfordert jedoch spezifisches Wissen über die im Modell enthaltenen Komponenten, unter anderem, wie sie installiert, konfiguriert und verwaltet werden können. Auch die Zusammenstellung und Verbindung der Komponenten selbst, um beispielsweise einen funktionierenden Analysestack zu erstellen, erfordert domänenspezifisches Wissen. Weiterhin werden Zielumgebung-spezifische Informationen, beispielsweise, dass die Daten per SQL abgefragt werden müssen, bei allen Modellierungsvarianten zur Entwicklung benötigt.

4.3 Paketierung von eigenständigen Deployment-Paketen

Deployment-Pakete bündeln die für die Bereitstellung und Verwaltung einer modellierten Anwendung erforderlichen Dateien in einem einheitlichen und portablen Format. Ein solches Deployment-Paket kann von einer kompatiblen Bereitstellungsumgebung interpretiert und verarbeitet werden, und wird dementsprechend zum automatisierten Deployment der modellierten Anwendung verwendet. In diesem Kapitel wird basierend auf den in Abschnitt 4.3.1 diskutierten Problemen sowie Anforderungen bezüglich der Erstellung von Deployment-Paketen, in Abschnitt 4.3.2 das Konzept und in Abschnitt 4.3.3 die Architektur eines Paketierungsframeworks vorgestellt, das es ermöglicht, uneigenständige Deployment-Pakete in eigenständige Deployment-Pakete umzuwandeln.

4.3.1 Problemstellung und Anforderungen an die Paketierung

In einem Deployment-Paket sind neben der Anwendungstopologie (vgl. Definition 4.1) der modellierten Anwendung auch benötigte Artefakte wie beispielsweise Managementartefakte (vgl. Definition 4.23) sowie Anwendungsartefakte (vgl. Definition 4.24) enthalten. Typischerweise werden bei der Bereitstellung von Anwendungen, unabhängig davon, ob diese manuell oder automatisiert durchgeführt wird, jedoch externe Abhängigkeiten benötigt, die vor ihrer Installation zuerst aufgelöst und heruntergeladen werden müssen [WBL14a; ZBH+20]. In Skripten oder Konfigurationsmanagementtools wird beispielsweise häufig das Paketverwaltungsprogramm *apt-get*[1] hierzu verwendet. Mit `apt-get install python` können beispielsweise die zur Installation einer Python-Laufzeitumgebung benötigten Dateien heruntergeladen sowie installiert werden. Ebenfalls wird das Kommandozeilenprogramm *wget*[2] typischerweise verwendet um externe Dateien vor der Installation in der Zielumgebung per HTTP, HTTPS, FTP oder FTPS Protokoll herunterzuladen. Da die externen Abhängigkeiten erst in der Zielumge-

[1] https://manpages.ubuntu.com/manpages/jammy/en/man8/apt-get.8.html
[2] https://manpages.ubuntu.com/manpages/jammy/en/man1/wget.1.html

bung zur Installation benötigt werden, können dadurch leichtgewichtige Deployment-Pakete erstellt werden. Solche Deployment-Pakete werden auch als uneigenständig beschrieben und wie folgt informell definiert:

Definition 4.36 (Uneigenständiges Deployment-Paket)
Ein Deployment-Paket wird als uneigenständig oder nichteigenständig bezeichnet, wenn die erfolgreiche Bereitstellung der modellierten Anwendung in der Zielumgebung von externen, also außerhalb des Deployment-Pakets befindlichen, Dateien abhängig ist. ■

Das Auflösen und Herunterladen externer Abhängigkeiten kann jedoch die Bereitstellung erheblich verlangsamen oder sogar verhindern, falls der Zugang zum Internet limitiert, instabil oder generell gesperrt ist. Weiterhin können Abhängigkeiten, die von einer externen Quelle bezogen werden, mit der Zeit veralten und von ihrer bisherigen Quelle gelöscht werden oder an eine andere Stelle verschoben werden, sodass auf diese nicht mehr zugegriffen werden kann [ZBG+18]. Dementsprechend können solche Umstände die erfolgreiche Ausführung von leichtgewichtigen Deployment-Paketen verhindern, bzw. müssen diese für eine erfolgreiche Ausführung zuerst wieder repariert und angepasst werden. In solchen Fällen werden daher eigenständige und in sich geschlossene Deployment-Pakete benötigt, die alles enthalten was für die Bereitstellung und Verwaltung der Anwendung erforderlich ist und keine Abhängigkeiten zu externen Dateien besitzen.

Definition 4.37 (Eigenständiges Deployment-Paket)
Ein Deployment-Paket wird als eigenständig bezeichnet, wenn die darin modellierte Anwendung in der Zielumgebung ohne externe Dateien erfolgreich bereitgestellt werden kann. Ein eigenständiges Deployment-Paket enthält somit alle benötigten Anwendungsartefakte und verweist in den ebenfalls enthaltenen Managementartefakten auf keine externen Quellen. ■

Da jedoch beide Varianten eines Deployment-Pakets ihre Vorteile sowie Nachteile haben, soll der Nutzer hinsichtlich seiner Möglichkeiten nicht eingeschränkt werden. Demzufolge sollen beide Ausprägungen eines Deployment-

Pakets von den einzelnen Forschungsbeiträgen dieser Arbeit (siehe Abbildung 1.1) sowie insbesondere der Shipping-Methode (siehe Abschnitt 3.2) unterstützt werden. Im folgenden Abschnitt wird daher ein Konzept zur Umwandlung von uneigenständigen Deployment-Paketen in eigenständige Deployment-Pakete vorgestellt. Dadurch kann die Modellierung der Anwendung sowie die grundsätzliche Erstellung von Deployment-Paketen einheitlich und somit unabhängig von ihrer späteren Ausprägung erfolgen.

4.3.2 Konzept des Paketierungsframeworks

Ziel des Ansatzes ist es, die automatisierte Umwandlung eines nichteigenständigen Deployment-Pakets in ein eigenständiges Deployment-Paket zu ermöglichen und damit die Erstellung von Deployment-Paketen auf einheitliche Weise zu unterstützen. Zur Realisierung der nötigen Transformation müssen alle Artefakte eines Deployment-Pakets betrachtet und analysiert werden, um definierte externe Abhängigkeiten innerhalb dieser Artefakte zu finden. Außerdem müssen anschließend diese externen Abhängigkeiten aufgelöst, heruntergeladen und dem jeweiligen Deployment-Paket hinzugefügt werden. Weiterhin müssen die vorhandenen Referenzen zu diesen externen Abhängigkeiten in den entsprechenden Artefakten angepasst werden, um die vorgenommenen Änderungen widerzuspiegeln. Hierfür müssen unter anderem Verweise auf externe Quellen, beispielsweise auf Internetadressen oder Repositorien, auf die entsprechend nun innerhalb des Deployment-Pakets vorhandenen Dateien abgeändert werden. Ebenso müssen gegebenenfalls bisherige Befehle, zum Beispiel innerhalb eines Skriptes zum Herunterladen und Installieren einer Komponente, entsprechend angepasst werden.

Beispielsweise muss anstelle des Befehls `apt-get install <package>`, der das angegebene Paket sowie dessen eigene Abhängigkeiten herunterlädt und installiert, der Befehl `dpkg -install <*.deb-files>`, zur Installation der während der Transformation aufgelösten, heruntergeladenen und dem Deployment-Paket hinzugefügten Installationsdateien, ausgeführt werden. Falls ein Dockerfile zur Installation einer Komponente verwendet wird,

muss zunächst das durch dieses Dockerfile beschriebene Docker-Image lokal gebaut werden. Danach muss das Image exportiert und dem Deployment-Paket hinzugefügt werden. Entsprechend müssen auch hier die Befehle, die während der Bereitstellung ausgeführt werden, angepasst werden.

Da die Artefakte eines Deployment-Pakets durch verschiedenste Technologien, beispielsweise Ansible Playbooks[1], Chef Cookbooks[2] oder Dockerfiles[3] implementiert sein können, ist das Paketierungsframework Plugin-basiert konzipiert. Ein Plugin beinhaltet dabei die Logik, um Artefakte eines konkreten Typs zu analysieren, externe Abhängigkeiten darin zu erkennen, diese aufzulösen, herunterzuladen sowie nötige Änderungen bezüglich den Ausführungsbefehlen umzusetzen. Somit können Plugins typspezifisch entwickelt und anschließend einfach in das Framework integriert werden.

Aufgrund der Transformation ermöglicht dieser Ansatz dem Modellierer der Anwendungstopologie beide Deployment-Paket Varianten auf eine einheitliche Art und Weise und ohne zusätzlichen Aufwand erstellen zu können. Dieses Konzept hat weiterhin den Vorteil, dass vorhandene Artefakte, unabhängig davon, ob sie externe Abhängigkeiten besitzen oder nicht, einfach wiederverwendet werden können. Ebenfalls kann je nach Anwendungsfall, entweder das leichtgewichtige, aber nichteigenständige Deployment-Paket oder das eigenständige Deployment-Paket erstellt und genutzt werden.

4.3.3 Architektur des Paketierungsframeworks

In diesem Abschnitt wird die Architektur eines Paketierungsframeworks zur automatisierten Umwandlung von nichteigenständigen Deployment-Paketen in eigenständige Deployment-Pakete vorgestellt. Die Architektur ist in Abbildung 4.5 dargestellt und unterstützt die in Abschnitt 3.2 vorgestellte Methode zum Shippen von Funktionen und Daten sowie realisiert insbesondere den Schritt 8 (siehe Abschnitt 3.2.8) dieser Methode. Die Hauptkomponenten des Paketierungsframeworks sind: (i) die Deployment-

[1] https://docs.ansible.com/ansible/latest/user_guide/playbooks.html
[2] https://docs.chef.io/cookbooks/
[3] https://docs.docker.com/engine/reference/builder/

Paket Handler Komponente, (ii) die Artefakte Handler Komponente sowie (iii) die Plugin Verwaltung Komponente. Diese Komponenten, der Aufbau der einzelnen typspezifischen Plugins sowie der grundsätzliche Ablauf zur Umwandlung von Deployment-Paketen wird im Folgenden erläutert.

Deployment-Pakete sind typischerweise Archive, beispielsweise ZIP-Archive, die die Struktur und Beschreibung einer Anwendung sowie zur Bereitstellung und Verwaltung benötigte ausführbare Artefakte enthalten. Ein Deployment-Paket mit externen Abhängigkeiten, das in ein eigenständiges Deployment-Paket umgewandelt werden soll, muss dementsprechend zunächst entpackt und für die Verarbeitung des Inhalts vorbereitet werden. Weiterhin müssen am Ende des gesamten Transformationsprozesses, wenn alle externen Abhängigkeiten aufgelöst sind, die angepassten Artefakte sowie die zusätzlich heruntergeladenen Dateien wieder entsprechend als Deployment-Paket paketiert werden. Daher ermöglicht der Deployment-Paket Handler das Entpacken und Verpacken von Deployment-Paketen, den Zugriff auf die enthaltenen Dateien sowie die Erhaltung der Konsistenz des Pakets bezüglich Ordnerstruktur und des generellen Aufbaus des Deployment-Pakets.

Um externe Abhängigkeiten, die die Bereitstellung einer Anwendung in einer Umgebung ohne Internetzugang verhindern, auflösen zu können, müssen alle in einem Deployment-Paket enthaltenen Artefakte auf entsprechende Verweise überprüft werden. Daher ermöglicht der Artefakte Handler das Durchsuchen eines Deployment-Pakets nach allen Arten von enthaltenen und ausführbaren Artefakten, wie beispielsweise Shellskripten, Konfigurationsmanagementtools oder Dockerfiles, die möglicherweise externe Abhängigkeiten spezifizieren. Hierzu nutzt der Artefakte Handler die Funktionalitäten des Deployment-Paket Handlers. Da das Auflösen der externen Abhängigkeiten aufgrund der Heterogenität nicht generalisiert werden kann, wird entsprechend dem Typ eines gefundenen Artefakts ein passendes Plugin zur weiteren Überprüfung und Bearbeitung dieses Artefaktes aufgerufen.

Da das Paketierungsframework weiterhin einfach erweiterbar sein soll, bezüglich der unterstützten Technologien und Tools, bietet es mit der Plugin Verwaltung einen Mechanismus für die Handhabung technologiespezifischer

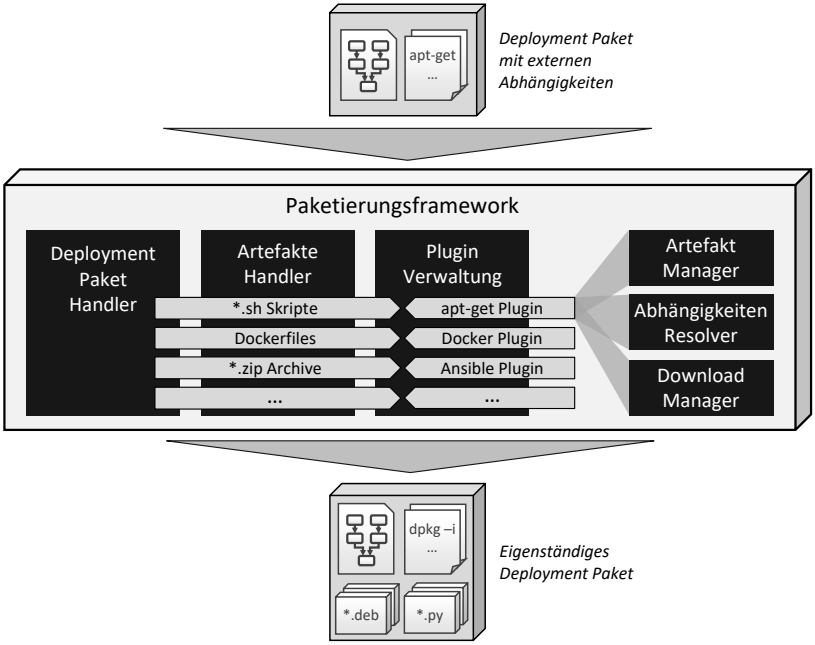


Abbildung 4.5: Architektur des Paketierungsframeworks

Plugins an, die beispielsweise Ansible Playbooks, Shellskripte oder Dockerfiles unterstützen. Hier werden die Plugins mit ihren entsprechend jeweils unterstützen Artefakttypen sowie Dateiendungen registriert. Ein Plugin zum Auflösen von externen Abhängigkeiten von `apt-get install` Befehlen unterstützt beispielsweise die Bearbeitung von *.sh Skripten, da diese Befehle in solchen Skripten genutzt werden. Ein Plugin zur Bearbeitung von Ansible Playbooks registriert sich beispielsweise auf als *.zip Archive gepackte Artefakte, da Ansible Playbooks typischerweise in dieser Form vorliegen. Plugins können sich dabei für ihre Verwendung von mehreren Artefakttypen sowie Dateiendungen registrieren. Ebenso können auch pro Artefakttyp sowie Dateiendung mehrere Plugins registriert sein. Dies ist nötig, da beispielsweise in einem Shellskript nicht zwingend nur *apt-get* Befehle vorkommen, sondern typischerweise auch weitere Befehle, wie zum Beispiel *wget* genutzt werden.

Jedes Plugin selbst enthält weiterhin typischerweise die drei Komponenten (i) Artefakt Manager, (ii) Abhängigkeiten Resolver sowie (iii) Download Manager. Je nach zu unterstützendem Typ beziehungsweise zu unterstützender Technologie werden gegebenenfalls nicht alle Komponenten benötigt und können dementsprechend weggelassen oder zusammengefasst werden. Im Falle von Docker beispielsweise, werden Abhängigkeiten beim Bauen des Docker-Images automatisch ermittelt und heruntergeladen, wodurch separate Komponenten hierfür entsprechend überflüssig sind. Der Artefakt Manager ermöglicht zum einen das Analysieren von Artefakten, um externe Abhängigkeiten entsprechend den unterstützten Typen des Plugins zu finden. Zum anderen ermöglicht der Artefakt Manager die Bearbeitung von Artefakten, um beispielsweise darin enthaltene Befehle anzupassen. Der Abhängigkeiten Resolver ist für die Auflösung der explizit angegebenen externen Abhängigkeiten sowie weiterer erforderlicher impliziter Abhängigkeiten zuständig. Der Befehl `apt-get install python` installiert zum Beispiel nicht nur Python selbst, sondern auch weitere Pakete, die für die Installation von Python benötigt werden. Der Abhängigkeiten Resolver analysiert hierzu jede gefundene Abhängigkeit, die in einem Artefakt explizit angegeben ist, und ermittelt gegebenenfalls den entsprechenden Abhängigkeitsbaum (engl. dependency tree) für jede dieser gefundenen Abhängigkeiten. Der Download Manager ist anschließend für das Herunterladen der ermittelten expliziten sowie impliziten Abhängigkeiten verantwortlich. Mithilfe der Funktionalitäten der Deployment-Paket Handler Komponente können die heruntergeladenen Dateien sowie die angepassten Artefakte wieder in die entsprechenden Ordner des Deployment-Pakets gelegt werden.

Die Architektur und prototypische Implementierung der Paketierungsframework Komponente wird in Kapitel 8 im Kontext des Gesamtsystems genauer betrachtet sowie validiert. Weiterhin wird die prototypische Implementierung evaluiert, indem unter anderem die jeweilige Bereitstellungszeit für beispielhafte nichteigenständige Deployment-Pakete mit ihren eigenständigen Pendants verglichen wird, sowie zusätzlich die jeweilige Größe der verschiedenen Paketvarianten miteinander verglichen wird.

4.4 Zusammenfassung und Diskussion

Da die in Abschnitt 3.2 vorgestellte Methode auf deklarativen Anwendungstopologien aufbaut, wurde in diesem Kapitel mit dem D^{3}M-Metamodell eine zur deklarativen Modellierung von verteilbaren Anwendungen nutzbare Modellierungssprache vorgestellt. Die Modellierungssprache basiert dabei auf bestehenden Modellierungssprachen wie TOSCA [OAS13b], EDMM [WBF+19] und DMMN [Bre16], und erweitert diese um weitere benötigte Elemente um nötige Informationen, wie beispielsweise die Schnittstellen der Anwendungskomponenten, zur Durchführung der in Kapitel 3 vorgestellten Methode modellieren zu können.

Auf Basis des D^{3}M-Metamodells wurden anschließend drei verschiedene Modellierungsansätze für das Deployment von Anwendungen, beziehungsweise einzelner Komponenten einer Anwendung vorgestellt, die zum einen die automatisierte Bereitstellung und Verwaltung dieser Anwendungen oder Komponenten in einer entfernten Infrastruktur ermöglichen sowie zum anderen die Verknüpfung dieser Anwendungen oder Komponenten mit in der Zielumgebung verfügbaren Datenquellen ermöglichen. Um die Auswahl eines geeigneten Modellierungsansatzes für ein konkretes Szenario zu ermöglichen, wurden zudem die jeweiligen Vor- und Nachteile der vorgestellten Modellierungsansätze beschrieben.

Weiterhin wurde in diesem Kapitel ein Konzept vorgestellt, um nichteigenständige Deployment-Pakete, also Deployment-Pakete mit externen Abhängigkeiten, in eigenständige Deployment-Pakete, also Deployment-Pakete ohne externe Abhängigkeiten, transformieren zu können. Durch den vorgestellten Ansatz kann die automatisierte Bereitstellung von Anwendungen in einer Umgebung ohne Internetzugang, wie beispielsweise in Produktionsumgebungen, die aus Gründen der Datensicherheit und des Datenschutzes zum Teil keine externen Internetverbindungen erlauben, ermöglicht werden. Darüber hinaus ermöglicht der Ansatz, der Paketierung aller nötigen Dateien, die Erhaltung aller zur Bereitstellung erforderlichen Softwarekomponenten und Abhängigkeiten, zum Beispiel für Forschungssoftware. Außerdem wur-

de die Architektur eines Paketierungsframeworks zur Realisierung dieser Transformation vorgestellt. Das Paketierungsframework durchsucht dabei Deployment-Pakete nach enthaltenen Artefakten, die externe Abhängigkeiten spezifizieren, löst diese Abhängigkeiten auf, lädt die entsprechenden Dateien herunter, passt die jeweiligen Artefakte an und paketiert schließlich ein somit eigenständiges Deployment-Paket.

Kapitel 5

Entscheidungsfindung zur Platzierung von Funktionen und Daten

Um Datenanalysen zu ermöglichen, die neue Erkenntnisse und Optimierungspotenziale beispielsweise in Fertigungsprozessen bieten, müssen typischerweise Daten aus unterschiedlichen Quellen und Standorten integriert, analysiert und verglichen werden. Aufgrund der immer weiter zunehmenden Menge an gespeicherten und zur Analyse zu übertragenden Daten wird das Netzwerk jedoch zunehmend zu einem Engpass und die Übertragung von Daten von verschiedenen Orten zu einem zentralisiertem Ort zu einem Problem (vgl. Abschnitt 2.2). Dementsprechend müssen die Komponenten einer Anwendung zur Analyse dieser Daten, abhängig von unter anderem den spezifischen Anforderungen sowie der konkreten Herkunft der zu analysierenden Daten, auf verschiedene Standorte, beispielsweise auf private und öffentliche sowie Fog- und Edge-Clouds (vgl. Abschnitt 2.1), verteilt werden.

In diesem Kapitel wird daher ein Konzept vorgestellt, das das Deployment verteilter Anwendungen hinsichtlich der Platzierung ihrer datenverarbeitenden Komponenten und Datensätzen automatisiert optimiert und dabei Faktoren, wie beispielsweise die Größe und Lokalität der einzelnen Datensätze sowie den Datenfluss der Anwendung berücksichtigt. Dieses Kapitel beschreibt somit das Konzept zur Realisierung von Schritt 6 der in Abschnitt 3.2 vorgestellten Shipping-Methode und stellt weiterhin Forschungsbeitrag 3 der vorliegenden Arbeit dar. Für einen Überblick über verwandte Arbeiten in diesem Bereich sei auf Abschnitt 2.3 verwiesen.

In Abschnitt 5.1 wird zunächst die Notwendigkeit dieses Forschungsbeitrags motiviert. Anschließend wird in Abschnitt 5.2 eine Methode zur Gruppierung und Platzierung der einzelnen Komponenten einer Anwendung vorgestellt. In Abschnitt 5.3 werden dann die einzelnen Algorithmen zur automatisierten Gruppierung und Platzierung aufgeführt und erläutert. Anschließend folgt in Abschnitt 5.4 eine Validierung des vorgestellten Konzepts sowie in Abschnitt 5.5 eine Diskussion und Zusammenfassung dieses Kapitels.

5.1 Motivation

Um die Motivation und Notwendigkeit dieses Forschungsbeitrags zu verdeutlichen, wird in diesem Abschnitt ein Szenario für die Platzierung der Komponenten einer datenverarbeitenden Cloud-Anwendung auf Grundlage ihres Datenflusses vorgestellt. In Abbildung 5.1 ist eine beispielhafte Cloud-Anwendung dargestellt, die aus datenverarbeitenden Komponenten besteht und zur Analyse verschiedener heterogener Datensätze, welche über unterschiedliche Datenbanken, Provider sowie Lokalitäten verteilt sind, eingesetzt werden soll. Neben den einzelnen Komponenten zur Verarbeitung der Daten und den verschiedenen auszuwertenden Datensätze selbst gibt die Abbildung auch den Datenfluss der Gesamtanwendung wieder.

Auf der linken Seite der Abbildung ist ein Data Scientist abgebildet, der für die Durchführung der Analyse zuständig ist. Dieser hat 5 Terabyte an Daten gesammelt (*Daten 1*), die in einer *PostgreSQL*-Datenbank gespeichert

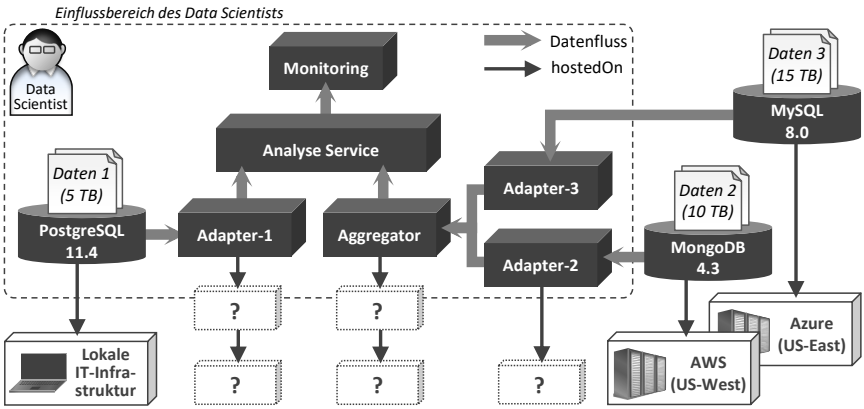


Abbildung 5.1: Szenario einer datenverarbeitenden verteilten Anwendung

sind und auf einer *lokalen IT-Infrastruktur* gehostet wird, beispielsweise in einer privaten Cloud. Auf der rechten Seite der Abbildung sind zwei weitere Datensätze dargestellt: (i) *Daten 2*, gespeichert in einer *MongoDB*-Datenbank, und *Daten 3*, gespeichert in einer *MySQL*-Datenbank. Die beiden Datenbanken, die die zu analysierenden Datensätze enthalten, werden bei den beiden Public-Cloud-Anbietern *AWS* und *Azure* gehostet. Während der Datensatz *Daten 2* ein Volumen von 10 Terabyte hat und in der Region US-West gespeichert ist, hat der Datensatz *Daten 3* ein Volumen von 15 Terabyte und ist in der Region US-Ost gespeichert. Weiterhin werden beide Datenbanken von Dritten verwaltet, die nur Lesezugriff darauf gewähren.

Diese beiden Datenbanken liegen somit außerhalb des Einflussbereichs des Data Scientists, welcher in der Abbildung durch den gestrichelten Rahmen dargestellt ist. Die datenverarbeitenden Komponenten der dargestellten Anwendung, also die drei Adapter zum Abrufen der Daten, der *Aggregator* zum Aggregieren der Datensätze *Daten 2* und *Daten 3*, der *Analyse Service* zum Analysieren der Daten und der *Monitoring* Dienst zum Überwachen der Ergebnisse, werden dagegen vom Data Scientist entwickelt und verwaltet, und befinden sich daher innerhalb seines Einflussbereichs.

Das Ziel des Data Scientists ist es, die beiden in den USA gespeicherten Datensätze zuerst zu aggregieren, dann gemeinsam zu analysieren und anschließend mit seinem lokal verfügbaren Datensatz zu vergleichen. Hierzu müssen die verschiedenen Datensätze, beziehungsweise die resultierenden Zwischenergebnisse aus der Bearbeitung einer Komponente, an einen gemeinsamen Ort gebracht werden. Da die beiden in den USA verfügbaren Datensätze jedoch zusammen 25 Terabyte groß sind, wäre ein manuelles Herunterladen der Daten aus den USA zur Verarbeitung und Analyse der Daten vor Ort sehr zeitaufwändig. Die Bereitstellung der gesamten Anwendung in einer lokalen Umgebung und das Abrufen der Daten aus den entfernten Datenbanken würde aufgrund der Größe der Datensätze ebenfalls zu einer hohen Übertragungszeit führen. Ebenso wäre die Bereitstellung aller erforderlichen Komponenten und das Hochladen von 5 Terabyte an Daten in der Nähe der Daten in den USA zeitaufwändig und wenig effizient. In solchen Szenarien (vgl. Abschnitt 2.2), in denen große Datenmengen von verschiedenen Standorten zu einem entfernten Standort übertragen werden müssen, wird das Internet aufgrund der technischen Umsetzung vor allem mit zunehmender geografischer Entfernung und zunehmender Anzahl an Hops zu einem Engpass, der sich negativ auf die Leistungsfähigkeit einer verteilten Anwendung auswirkt [APHS21; Sch18; SDH+11; TSZS06; ZBK+20].

Abhängig vom Datenfluss, den Größen der Datensätze sowie den Eigenschaften der einzelnen Komponenten, die die Daten verarbeiten, zum Beispiel, ob diese die Datenmenge reduzieren oder vergrößern, kann durch eine sinnvolle Verteilung der Komponenten, die Menge an Daten die über große Entfernungen versendet werden muss verringert und damit die Leistung der Anwendung verbessert werden. Datenreduzierende Verarbeitungsschritte sind beispielsweise die Aggregierung von Zeitreihen oder die Auswahl von Teilmengen, indem zum Beispiel irrelevante oder redundante Daten aus dem originalen Datensatz entfernt werden. Die Datenmenge vergrößernde Verarbeitungsschritte sind beispielsweise das Interpolieren von Zeitreihen oder das Entpacken von komprimierten Datensätzen. Im optimalen Fall wären alle Komponenten einer Anwendung sowie alle Datensätze in der

gleichen IT-Infrastruktur vorhanden, um die Übertragungswege so klein wie möglich zu halten. Dadurch würde man von einem erhöhten Datendurchsatz sowie einer erhöhten Geschwindigkeit im lokal begrenzten Netzwerk, zum Beispiel innerhalb eines Rechenzentrums, profitieren. Dies ist jedoch wie in dem in Abbildung 5.1 beispielhaft dargestelltem Szenario aufgrund von heterogenen und verteilten Datensätzen oft nicht möglich, so wie beispielsweise im Kontext von Industry 4.0 [KWXD17; OJ15; ZBF+17a], wo Daten in verschiedenen Fabriken generiert aber zusammen analysiert werden.

Ziel des Ansatzes ist es daher, datenverarbeitende Komponenten und bereitzustellende Datensätze in Abhängigkeit von unter anderem den Eigenschaften jeder einzelnen Komponente sowie dem Datenfluss der Gesamtanwendung geeignet zu platzieren um damit die Menge der zu übertragenden Daten über das Internet und über große Entfernungen hinweg zu reduzieren. Falls möglich, sollten Datensätze und Komponenten dementsprechend so platziert werden, dass datenreduzierende Komponenten nah an den zu verarbeitenden Daten, beispielsweise innerhalb des gleichen Datencenters, bereitgestellt werden. Demgegenüber sollte eine Komponente, welche die Datenmenge vergrößert, möglichst nah an der Komponente, welche den nächsten Arbeitsschritt realisiert, platziert werden.

5.2 Platzierungs-Methode

In diesem Abschnitt wird die Methode zur Platzierung der Komponenten datenverarbeitender Anwendungen vorgestellt. Ziel ist es, die Komponenten auf der Grundlage des Datenflusses der Gesamtanwendung und der Eigenschaften der einzelnen Komponenten so zu verteilen, dass die Datenmenge, die über weite geografische Entfernungen und eine Vielzahl an Hops über das Internet oder das Netzwerk von einem Ort zum anderen verschoben werden muss, reduziert wird und somit die Leistung der Anwendung verbessert wird [CZ14; KPL+09; Sch18]. Ein Überblick über die Methode ist in Abbildung 5.2 dargestellt. Details zu den einzelnen Schritten der Methode werden in den nachfolgenden Abschnitten erläutert und diskutiert.

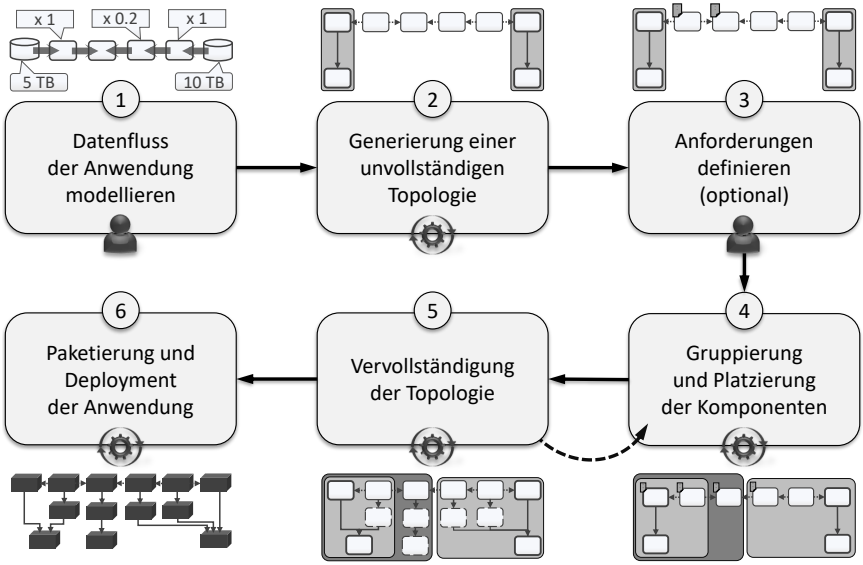


Abbildung 5.2: Übersicht der Methode zur Gruppierung und Platzierung der Komponenten von datenverarbeitenden Anwendungen

5.2.1 Datenfluss der Anwendung modellieren

Im ersten Schritt der Methode muss ein Datenflussmodell erstellt werden, das (i) die Datenquellen, (ii) die Komponenten der datenverarbeitenden Anwendung, (iii) den Datenfluss zwischen den Datenquellen und den Komponenten, (iv) die Größe der Datensätze und (v) den Datenfaktor (DF) der Komponenten darstellt. Der Datenfaktor beschreibt, ob und inwieweit eine datenverarbeitende Komponente (i) die Datenmenge bei der Verarbeitung reduziert (DF $<$ 1), (ii) sie vergrößert (DF $>$ 1) oder (iii) sich hinsichtlich der Datenmenge neutral verhält (DF $=$ 1). Der Datenfaktor ist somit erforderlich, um eine Komponente, die die Größe der zu übertragenden Daten verringert, möglichst nahe an den zu verarbeitenden Daten platzieren zu können. Andererseits sollte eine Komponente, die die Datengröße vergrößert, so nahe wie möglich an ihrem Zielverarbeitungsort platziert werden. Im

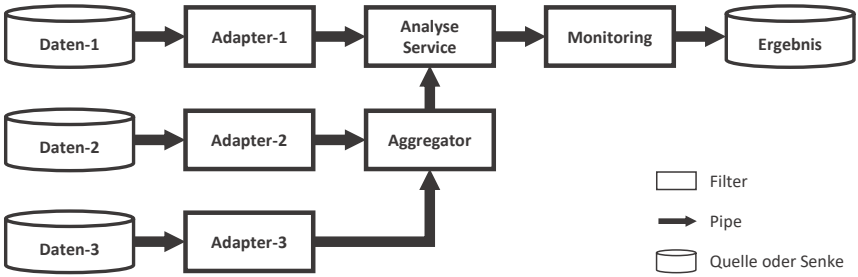


Abbildung 5.3: Modellierung des Datenflusses des zuvor in Abschnitt 5.1 vorgestellten Szenarios mittels Pipes und Filter

Falle einer bezüglich der resultierenden Datenmenge neutralen Komponente kann diese Komponente beliebig entweder nahe an der Datenquelle oder Datensenke platziert werden. Für die Modellierung des Datenflusses wird ein an Pipes und Filter [Meu95] orientierter Ansatz verwendet. In Abbildung 5.3 ist beispielhaft das zuvor in Abschnitt 5.1 zur Motivation dieses Forschungsbeitrags dargestellte Szenario mittels Pipes und Filter modelliert.

Grundsätzlich ist ein auf Pipes und Filter basierendes Modell ein gerichteter Graph $G = (V, E)$, der aus Knoten sowie Kanten besteht. Die Knoten V werden dabei als Filter und die Kanten E als Pipes bezeichnet. Verarbeitende Filter haben Eingabe- und Ausgabedaten und stellen Verarbeitungsschritte, also unter anderem Komponenten, die zum Beispiel Daten aggregieren oder interpolieren dar. Datenquellen und Datensenken werden ebenfalls als Knoten dargestellt, da sie entweder die in den Datenfluss eingehenden Daten (Datenquelle) oder die am Ende des Datenfluss resultierenden Daten (Datensenke) repräsentieren. Die Pipes verbinden die Filter und beschreiben weiterhin die Richtung des Datenflusses. Durch die Verwendung von Pipes und Filter sowie der zusätzlichen Möglichkeit zur Annotation der Datenvolumina der Datenquellen sowie den Datenfaktoren der einzelnen Filter, kann der Datenfluss innerhalb einer Anwendung detailliert modelliert werden. Dies ermöglicht, eine Anwendung so zu verteilen, dass Komponenten, die viele Daten austauschen, möglichst nahe beieinander deployt werden.

```
<DataFlow id="ExampleDataFlow" ...>
  <Filters>
    <Filter id="Daten-3">
      <Properties>
        <DataSize>15</DataSize>
        ...
      </Properties>
    </Filter>
    <Filter id="Adapter-3">
      <Properties>
        <DataFactor>0.5</DataFactor>
        ...
      </Properties>
    </Filter>
    ...
  </Filters>
  <Pipes>
    <Pipe transferType="pull">
      <Source>Daten-3</Source>
      <Target>Adapter-3</Target>
    </Pipe>
    ...
  </Pipes>
</DataFlow>
```

Listing 5.1: Simplifizierte Definition eines Datenflusses basierend auf dem zuvor vorgestellten Szenario

Listing 5.1 stellt dem Szenario dieses Kapitels folgend, ein vereinfachtes Beispiel zur Definition eines Datenflusses basierend auf Pipes und Filter dar. Das verwendete Modell ist XML-basiert und unterstützt die beiden Hauptelemente Pipes und Filter. Filter haben eine *id* und können zudem Eigenschaften (*Properties*) enthalten, um beispielsweise das Datenvolumen (*DataSize*) im Falle einer Datenbank oder den Datenfaktor (*DataFactor*) im Falle einer Verarbeitungskomponente zu definieren. Es können auch weitere Eigenschaften angegeben werden, die dann in das generierte D^3M übernommen werden. Die Generierung einer Topologie basierend auf einem solchen Datenflussmodell wird im folgenden Abschnitt beschrieben.

5.2.2 Unvollständige Topologie generieren

Im zweiten Schritt der Methode wird eine unvollständige Anwendungstopologie, also ein deklaratives Bereitstellungsmodell mit offenen Anforderungen, auf Basis des Datenflussmodells aus dem vorherigen Schritt erstellt. Basierend auf den IDs der Komponenten des Datenflussmodells kann entweder (i) eine bereits vorhandene Komponente in einem Repository gefunden und genutzt werden oder (ii) eine entsprechende Komponente muss erstellt und der Topologie hinzugefügt werden. Im zuvor vorgestellten Szenario könnten beispielsweise Verarbeitungskomponenten wie der *Analyse Service* oder *Adapter-1* auf konkrete ausführbare Implementierungen abgebildet werden. Der Analyse Service könnte beispielsweise als Java Anwendung implementiert sein, die auf einem Apache Flink Framework betrieben wird, und der Adapter als einfaches Python Skript, das die Daten aus der Datenquelle *Daten-1* lesen kann. Der Datenfluss selbst wird transformiert, indem für jede Pipe im Datenflussmodell, eine entsprechende Relation zur Topologie hinzugefügt wird. Diese Relationen haben den Typ *connectsTo*, der eine Verbindung zwischen zwei Komponenten oder einer Komponente und einer Datenquelle oder Senke definiert. Die Richtung der Relation hängt dabei davon ab, ob eine Komponente Daten von einer anderen Komponente anfordert, beispielsweise von einer Datenbank, oder die Daten aktiv zu einer anderen Komponente zur weiteren Verarbeitung geschoben werden. Diese Unterscheidung wird auch *push* und *pull* genannt [AFZ97; BMR+96; Kos00] und kann im Datenflussmodell für die jeweiligen Pipes annotiert werden (vgl. Zeile 18). Zum Beispiel initiieren die Adapterkomponenten im Szenario die Kommunikation mit den Datenbanken und rufen die Daten ab. Daher werden in diesem Fall die Adapter der jeweilige Ausgangspunkt und die Datenbank das jeweilige Ziel der einzelnen connectsTo-Relationen sein.

Nachdem alle Komponenten und Relationen zur Topologie hinzugefügt wurden, wird anschließend jede vorhandene Datenquelle automatisch mit einer initialen Lokalitätsgruppe annotiert: Eine *Lokalitätsgruppe* gibt an, dass die am Ende der Methodendurchführung so gruppierten Komponenten und Datenquellen möglichst nahe beieinander, bestenfalls im selben Datencenter,

betrieben werden sollen. Für bereits in einer Cloud-Umgebung verfügbare Datenquellen wird weiterhin deren Provider sowie Standort definiert, beziehungsweise in der Topologie mittels entsprechenden Knoten modelliert.

5.2.3 Anforderungen definieren

Im optionalen dritten Schritt, können Anforderungen an die Umgebungen, in den die Komponenten betrieben werden sollen, festgelegt werden. Beispielsweise Anforderungen bezüglich der erforderlichen Rechenleistung oder Speicherkapazität. Weiterhin können auch Einschränkungen oder Vorgaben hinsichtlich der Auswahl möglicher Hosting-Standorte als solche Anforderungen definiert werden. Zum Beispiel, dass Daten und die Komponenten, die diese Daten verarbeiten, aufgrund von Datenschutzrichtlinien oder anderen gesetzlichen Bestimmungen nur in einem bestimmten Land oder in der privaten Cloud eines Unternehmens gehostet werden dürfen [ZBKL18]. Im gezeigten Szenario könnten zum Beispiel die datenverarbeitenden Komponenten *Analyse Service* und *Aggregator* hohe Rechenleistung in ihrer jeweiligen Hosting-Umgebung benötigen. Außerdem könnte die Komponente *Adapter-1*, zum Lesen der Daten aus der Datenquelle *Daten-1*, die Anforderung haben, dass sie beispielsweise aufgrund von Datenschutzrichtlinien oder Gesetzen nur in einer bestimmten privaten Cloud bereitgestellt werden darf. Diese Anforderungen werden benötigt, um geeignete Hosting-Umgebungen für den Betrieb der jeweiligen Komponenten auswählen zu können.

5.2.4 Gruppierung und Platzierung der Komponenten

Im vierten Schritt wird die Platzierung der datenverarbeitenden Komponenten im Verhältnis zu den Datenquellen bestimmt. Als Input werden hierfür das im ersten Schritt erstellte Datenflussmodell sowie die unvollständige Topologie zusammen mit den spezifizierten Anforderungen und den initialen Lokalitätsgruppen aus den Schritten zwei und drei benötigt. Auf der Grundlage des Datenflusses, den Datenfaktoren sowie den spezifizierten Anforderungen der Komponenten, werden die Verarbeitungskomponenten

in ihre jeweils geeigneten Lokalitätsgruppen eingeteilt. Im motivierenden Szenario sollte zum Beispiel die Komponente *Aggregator* in der Nähe der beiden Komponenten *Adapter-2* und *Adapter-3* platziert werden, da dieser die zu übertragende Datenmenge reduziert. Nachdem alle Komponenten jeweils einer Lokalitätsgruppe zugeordnet sind, werden die einzelnen Gruppen jeweils einer konkreten Infrastruktur- oder Plattform-Komponente, wie beispielsweise einem Cloud-Anbieter, Hypervisor oder IoT-Gerät, die alle Anforderungen der sich in der Gruppe befindlichen Komponenten erfüllt, zugewiesen. In Abschnitt 5.3 werden die Algorithmen zur automatischen Bestimmung der Lokalitätsgruppen und der entsprechenden Platzierung der Komponenten im Detail beschrieben sowie diskutiert.

5.2.5 Vervollständigung der Topologie

In Schritt fünf wird die bisher unvollständige Topologie auf der Grundlage der in Schritt vier vorgenommenen Zuweisungen bezüglich den Zielumgebungen der Komponenten vervollständigt. Dies kann entweder manuell oder automatisiert mithilfe eines entsprechenden Ansatzes zur Vervollständigung von Topologien geschehen, wie er unter anderem in verschiedenen Arbeiten [HBBL14; PBL+17; SBKL17] vorgestellt wird. Bei diesen Ansätzen wird in der Regel rekursiv über alle Komponenten mit offenen Anforderungen iteriert und nach passenden Komponenten gesucht, die diese Anforderungen erfüllen. Nachdem die passenden Komponenten der unvollständigen Topologie hinzugefügt wurden und damit alle Anforderungen erfüllt sind, ist die resultierende Topologie vollständig und damit ausführbar. Im vorgestellten Szenario könnte die Komponente *Adapter-2* beispielsweise in Java implementiert sein und daher für den Betrieb eine Java Laufzeitumgebung benötigen, die wiederum eine virtuelle Maschine voraussetzt. Dementsprechend könnte eine solche Java Komponente durch einen Stack ersetzt werden, der die Java Komponente selbst sowie eine Java Laufzeitumgebung und eine virtuelle Maschine enthält, die wiederum von einem Cloud-Anbieter gehostet wird. Falls die Gruppierung aus Schritt vier keine erfolgreiche Vervollständigung ermöglicht, also nicht alle Anforderungen erfüllt werden können,

wird die problematische Komponente, beispielsweise ein Cloud-Anbieter, der bestimmte Anforderungen bezüglich Rechenleistung oder Datenschutz nicht erfüllt, von der Verwendung in der entsprechenden Lokalitätsgruppe ausgenommen und die Gruppierung samt anschließender Vervollständigung erneut durchgeführt. Falls dieser Schritt fehlerhaft abbricht, beispielsweise weil sich modellierte Anforderungen gegenseitig widersprechen, muss entweder (i) der Modellierungsfehler behoben und der Schritt erneut durchgeführt werden oder (ii) die finale Anwendungstopologie manuell erstellt werden.

5.2.6 Paketierung und Deployment der Anwendung

In Schritt sechs der Methode wird die Anwendung paketiert und kann anschließend mithilfe eines deklarativen Bereitstellungssystems automatisiert bereitgestellt werden. Die konkrete Durchführung dieses Schrittes, beziehungsweise die Auswahl eines geeigneten Bereitstellungssystems hängt von der verwendeten Modellierungssprache zur Modellierung der Topologie ab. Für die vorgestellte Methode ist grundsätzlich jedes Bereitstellungssystems nutzbar, das Topologien unterstützt und somit die deklarative Modellierung von Komponenten und deren Beziehungen ermöglicht. Mit dem D^3M-Metamodell wurde eine solche Sprache zur deklarativen Modellierung (Abschnitt 4.1) und Paketierung (Abschnitt 4.3) von verteilbaren Anwendungen definiert. Weiterhin wird in Kapitel 8 ein entsprechend dazu passender Prototyp eines solchen Bereitstellungssystems vorgestellt und diskutiert.

5.3 Algorithmen zur Bestimmung der Platzierung

Für die Automatisierung von Schritt vier der im vorherigen Abschnitt betrachteten Methode werden nachfolgend fünf Algorithmen vorgestellt. Zunächst wird ein High-Level-Algorithmus zur Koordination der weiteren Algorithmen präsentiert. Anschließend werden Algorithmen vorgestellt, welche einer Komponente, unter anderem auf der Grundlage des Datenflusses der Anwendung, zuerst einer Gruppe und anschließend einem bestimmten Standort

zuweisen. Weiterhin wird ein Algorithmus zur Auswahl einer konkreten Deployment-Plattform für die Komponenten an ihrem Standort präsentiert.

Algorithmus 5.1 bestimmeStandortUndPlatzierung(dfm, top)

```
1: // dfm => Datenflussmodel aus Schritt 1 der Platzierungs-Methode
2: // top => Unvollständige Topologie aus Schritt 3 dieser Methode
3: while (zuweisungMoeglich(top)) do
4:     standortZuweisen(dfm, top)
5:     anbieterZuweisen(top)
6:     try
7:         return vervollstaendigeTopologie(top)
8:     catch (vervollstaendigungNichtMoeglichFehler)
9:         loescheZuweisungen(top)
10:        setzeAnbieterAufBlacklist(top,fehler)
11:    end try
12: end while
13: throw platzierungNichtMoeglichFehler
```

Der High-Level-Algorithmus zur Bestimmung der Platzierung von Komponenten datenverarbeitender Anwendungen ist in Algorithmus 5.1 dargestellt. Er erhält als Eingabe das Datenflussmodell *dfm* aus Schritt eins der Platzierungs-Methode sowie die unvollständige Topologie *top*, die in Schritt zwei der Platzierungs-Methode generiert und in Schritt drei mit Anforderungen erweitert wurde. Der Algorithmus durchläuft eine Schleife, bis eine vollständige Topologie erstellt ist oder keine weitere Zuweisung mehr möglich ist (Zeile 3). Hierzu werden den Komponenten die Standorte zugewiesen (Zeile 4), an denen sie sich zur Laufzeit befinden sollen. Diese Standortszuweisung von Komponenten wird in Algorithmus 5.2 detailliert beschrieben. Anschließend werden den Komponenten die Cloud-Anbieter, Hypervisor oder Devices zugewiesen, die die Infrastruktur für die ausgewählten Standorte bereitstellen bzw. darstellen, basierend auf Anforderungen wie beispielsweise dem Datenschutz (Zeile 5). Die Zuordnung hierzu wird in Algorithmus 5.5 beschrieben. Anschließend kann die unvollständige Topologie anhand der zugewiesenen Standorten und Anbietern vervollständigt werden (Zeile 7). Das grundsätzliche Vorgehen hierfür wurde zuvor in Abschnitt 5.2.5 be-

schrieben. Der genutzte Ansatz zur Vervollständigung der Topologien basiert dabei auf Hirmer et al. [HBBL14] und wurde so erweitert, dass im Fehlerfall, falls keine Vervollständigung der Topologie möglich ist, eine entsprechende Fehlermeldung zurückgegeben wird, auf die reagiert werden kann.

Falls die Vervollständigung erfolgreich war, wird der Algorithmus beendet und die resultierende Topologie zurückgegeben (Zeile 7), die anschließend in Schritt sechs der Platzierungs-Methode deployt werden kann. Falls die Zuweisung mindestens einer Komponente aufgrund beispielsweise nicht erfüllter Anforderungen nicht möglich ist, wirft der Algorithmus einen Fehler, der die jeweiligen Komponenten und Anbieter beinhaltet, die zu dem Fehler geführt haben (Zeile 8). In diesem Fall werden die zuvor getätigten Zuweisungen gelöscht und die problematischen Anbieter zur Blacklist der jeweiligen Komponenten hinzugefügt (Zeilen 9 und 10). Falls aufgrund der dadurch von der Zuweisung ausgeschlossenen Anbietern keine weiteren Zuweisungen möglich sind (Zeile 3), gibt der Algorithmus eine Fehlermeldung aus (Zeile 13) und die Topologie muss manuell vom Benutzer angepasst werden. Dieser Fall kann jedoch nur dann eintreten, wenn keine der verfügbaren Cloud-Anbieter die Anforderungen einer Komponente erfüllen kann und somit die entsprechende Topologie grundsätzlich nicht deploybar ist.

Algorithmus 5.2 wird verwendet, um die Komponenten auf der Grundlage der verfügbaren Datenfaktoren, des Datenflusses der Gesamtanwendung, sowie der Datenvolumina der Datenquellen den Standorten zuzuweisen. Der Algorithmus erhält hierfür das Datenflussmodell und die unvollständige Topologie als Eingabe. Zunächst wird geprüft, ob keine beziehungsweise maximal eine Komponente bereits einen Standort zugewiesen hat (Zeile 3).

Dazu werden die beiden Methoden *gibKomponenten*, die die Menge der Komponenten in einer Topologie zurückgibt, und *hatStandort*, die prüft, ob einer Komponente bereits ein Standort zugewiesen ist, verwendet. Bei Komponenten mit einem bereits zugewiesenen Standort könnte es sich beispielsweise um Forschungsdaten handeln, die an einem bestimmten Ort gespeichert sind (vgl. Abbildung 5.1), oder um Komponenten, die aufgrund gesetzlicher Vorschriften an einem bestimmten Ort platziert werden müssen

Algorithmus 5.2 standortZuweisen(dfm, top)

```
1: // dfm => Datenflussmodel aus Schritt 1 der Platzierungs-Methode
2: // top => Unvollständige Topologie aus Schritt 3 dieser Methode
3: if (|{k | k ∈ gibKomponenten(top) : hatStandort(k)}| <= 1) then
4:     return weiseAllenKomponentenDenSelbenStandortZu(top)
5: end if
6: while (∃k ∈ gibKomponenten(top) : ¬hatStandort(k)) do
7:     komponenteZuweisen(dfm, k)
8: end while
```

(vgl. Abschnitt 5.1). Für den Fall, dass nur eine oder gar keine Komponente bereits einem Standort zugewiesen ist, werden alle Komponenten demselben Standort zugewiesen, da somit der Datentransfer zwischen verschiedenen Standorten vermieden werden kann (Zeile 4). Andernfalls wird über die noch nicht zugewiesenen Komponenten iteriert (Zeile 6) und diese jeweils mithilfe des Algorithmus 5.3 einem Standort zugewiesen (Zeile 7).

Algorithmus 5.3 erhält das Datenflussmodell sowie die zur Zuweisung ausgewählte Komponente k als Eingabe. Um die Komponenten-Gruppe bestimmen zu können, von der die meisten Daten zur Komponente k übertragen werden, darf die Komponente k keine eingehende Pipe haben, deren Quelle noch nicht zur einer Komponenten-Gruppe zugewiesen wurde. Daher wird über alle Pipes iteriert, die die Komponente k als Ziel haben (Zeile 6). Falls einer Quelle dieser Pipes noch keiner Komponenten-Gruppe und dementsprechend keinem Standort zugewiesen wurde, bricht der Algorithmus ohne Zuweisung ab (Zeilen 7 bis 9). In diesem Fall wird der Algorithmus in der nächsten Iteration entsprechend mit einer anderen Komponente aufgerufen. Da jedoch per Annahme festgelegt ist, dass Topologien sowie Datenflussmodelle keine Schleifen enthalten dürfen, gibt es immer mindestens eine Komponente die diese Bedingung erfüllt. Falls die Quelle einer Pipe einem Standort zugewiesen wurde, wird der Name des Standortes abgerufen und geprüft, ob in *standorteSet* bereits ein Tupel für diesen Standort hinzugefügt wurde (Zeilen 10 und 11). Falls dies zutrifft, wird anschließend das Datenvolumen, das über die aktuelle Pipe *pipe* übertragen werden muss, mit Algorithmus 5.4

Algorithmus 5.3 komponenteZuweisen(dfm, k)

```
1: // dfm => Datenflussmodel aus Schritt 1 der Platzierungs-Methode
2: // k => Komponente, der ein Standort zugewiesen werden soll
3: standorteSet := {} // Enthält Zuordnung zwischen Standorten zur
4:                    // Gruppierung von Komponenten und dem für
5:                    // jeden Standort berechnetem Datenvolumen
6: for all (pipe ∈ dfm : gibZiel(pipe) = k) do
7:     if (¬hatStandort(gibQuelle(pipe))) then
8:         return
9:     end if
10:    standort := gibStandort(gibQuelle(pipe))
11:    if (∃s ∈ standorteSet : gibStandort(s) = standort) then
12:        datenvolumen := berechneDatenvolumen(pipe)
13:        neu := (standort, gibDatenvolumen(s) + datenvolumen)
14:        standorteSet ← standorteSet \ {s} ∪ {neu}
15:    else
16:        neu := (standort, berechneDatenvolumen(pipe))
17:        standorteSet ← standorteSet ∪ {neu}
18:    end if
19: end for
20: for all (s ∈ standorteSet : max(gibDatenvolumen(s))) do
21:    // gibt es für den Standort einen Anbieter, der die Anforderungen
22:    // der Komponente erfüllt und nicht auf der Blacklist steht
23:    if (anbieterVorhanden(k, gibStandort(s))) then
24:        setzeStandort(k, gibStandort(s))
25:        return
26:    end if
27:    standorteSet ← standorteSet \ {s}
28: end for
```

berechnet (Zeile 12). Danach wird dieses Datenvolumen zum bisherigen, für diesen Standort gespeichertem, Datenvolumen hinzuaddiert und ein neues Tupel für diesen Standort erstellt (Zeile 13). Schließlich wird in *standorteSet* das alte Tupel für diesen Standort mit diesem neuen Tupel ersetzt (Zeile 14). Falls *standorteSet* noch kein Tupel für den Standort enthält, der dem zugewiesenem Standort der Quelle der aktuellen Pipe *pipe* entspricht (Zeile 15), wird ein neues Tupel für diesen Standort erstellt, welches das entsprechende

Datenvolumen der aktuellen Pipe beinhaltet (Zeile 16). Dieses neue Tupel wird anschließend *standorteSet* hinzugefügt (Zeile 17).

Somit enthält *standorteSet* am Ende der Schleife die Datenvolumen, die an die Komponente k übertragen werden müssen, gruppiert nach ihrem jeweiligen Standort, von dem die Daten stammen. Anschließend wird k zu dem Standort mit dem zu diesem Zeitpunkt maximalen Datenvolumen in *standorteSet* zugewiesen (Zeilen 20 bis 24). Dies verringert die Menge an Daten, die von verschiedenen Standorten zu k übertragen werden muss. Da die Zuweisung schrittweise erfolgt und dabei jeweils das aktuelle Optimum betrachtet wird, handelt es sich hierbei um einen Greedy-Algorithmus, der keine optimale Lösung garantiert. Falls diese Zuweisung nicht möglich ist, weil am entsprechenden Standort kein Anbieter verfügbar ist, der nicht auf der Blacklist der Komponente k steht (Zeile 23), wird dieser Standort aus *standorteSet* entfernt (Zeile 27) und im nächsten Schleifendurchlauf der Standort mit dem nächst größeren Datenvolumen ausgewählt (Zeile 20).

Bezüglich der Standortzuweisung wird im Rahmen dieser Arbeit angenommen, dass die Übertragung von Daten innerhalb eines Standortes, beispielsweise innerhalb desselben Rechenzentrums, schneller und zuverlässiger ist als die Übertragung zwischen verschiedenen Standorten [CGR+17; GHJ+09]. Weiterhin wird angenommen, dass eine zunehmende geografische Entfernung sowie eine zunehmende Anzahl an Hops zwischen verschiedenen Standorten ebenfalls einen zunehmenden negativen Effekt auf die Übertragung von Daten haben [APHS21; Sch18; SDH+11; TSZS06]. Für eine detailliertere Einbeziehung der Übertragungsgeschwindigkeit oder der Übertragungsqualität zwischen verschiedenen Standorten zur Gruppierung und Platzierung der Komponenten können die Algorithmen um entsprechende Ansätze [BDMA13; Gar17; JD02; PDMC03; SKK03] erweitert werden.

Zur Berechnung des Datenvolumens, das über eine bestimmte Pipe übertragen werden muss, verwendet Algorithmus 5.4 den Datenfaktor sowie das Datenvolumen der jeweils transitiv verbundenen Komponenten beziehungsweise Datenquellen. Falls der Ursprungs-Filter der gegebenen Pipe selbst keine eingehenden Pipes hat (Zeile 3), wird angenommen, dass er

Algorithmus 5.4 berechneDatenvolumen(pipe)

```
1: // pipe => Pipe, deren zu übertragendes Datenvolumen
2:            // berechnet werden soll
3: if (hatEingehendePipe(gibQuelle(pipe))) then
4:     datenvolumen := 0
5:     for all (p ∈ gibEingehendePipes(gibQuelle(pipe))) do
6:         datenvolumen ← datenvolumen + berechneDatenvolumen(p)
7:     end for
8:     return datenvolumen * gibDatenfaktor(gibQuelle(pipe))
9: end if
10: return gibDatenvolumen(gibQuelle(pipe))
```

eine Datenquelle ist. Dementsprechend kann das für diese Datenquelle annotierte Datenvolumen direkt zurückgegeben werden (Zeile 10). Andernfalls iteriert der Algorithmus über alle eingehenden Pipes (Zeile 5) und ruft sich rekursiv selbst auf, um das Datenvolumen zu ermitteln, das über die Pipe übertragen werden muss (Zeile 6). Anschließend wird das Ergebnis noch mit dem Datenfaktor des Ursprungs-Filters multipliziert, um das Datenvolumen zu berechnen, das über die Pipe *pipe* übertragen werden muss (Zeile 8).

Nach der Zuweisung von Komponenten zu Standorten müssen sie einem konkreten Cloud-Anbieter, Hypervisor oder einem vorhandenen Infrastruktur-Knoten zugewiesen werden, der diesen Standort unterstützt bzw. dort verfügbar ist, um die Vervollständigung der Topologie und damit die automatisierte Bereitstellung der Anwendung zu ermöglichen. Diese Zuweisung wird von Algorithmus 5.5 durchgeführt, der die Topologie *top* mit den zugewiesenen Standorten als Eingabe erhält. Zunächst wird ermittelt, welche der verfügbaren Cloud-Anbieter, Hypervisor oder Infrastruktur-Knoten überhaupt die jeweiligen Anforderungen der Komponenten unterstützt. Dazu iteriert der Algorithmus über jede Komponente in der Topologie (Zeile 4) und fragt mittels der Funktion *gibMoeglicheAnbieter* alle Cloud-Anbieter, Hypervisor oder vorhandene Infrastruktur-Knoten ab, die diese Komponente unterstützen (Zeile 5). Zur Verwaltung dieser Deployment-Möglichkeiten wird ein Repository verwendet, in dem alle Cloud-Anbieter, Hypervisor und vorhan-

Algorithmus 5.5 anbieterZuweisen(top)

```
1: // top => Topologie mit zugewiesenen Standorten
2: anbieterSet := {} // Enthält Zuordnung zwischen Anbietern und
3:                   // den jeweils darauf deploybaren Komponenten
4: for all (k ∈ gibKomponenten(top)) do
5:     for all (anbieter ∈ gibMoeglicheAnbieter(k)) do
6:         if (∃a ∈ anbieterSet : gibAnbieter(a) = anbieter) then
7:             neu := (anbieter, gibKomponenten(a) ∪ {k})
8:             anbieterSet ← anbieterSet \ {a} ∪ {neu}
9:         else
10:            anbieterSet ← anbieterSet ∪ {(anbieter, {k})}
11:        end if
12:    end for
13: end for
14: while (∃k_1 ∈ gibKomponenten(top) : ¬hatAnbieter(k_1)) do
15:    naechsterAnbieter := anbieter ∈ {a ∈ anbieterSet :
       max(|{k_2 | k_2 ∈ gibKomponenten(a) : ¬hatAnbieter(k_2)}|)}
16:    for all (k ∈ gibKomponenten(naechsterAnbieter) : ¬hatAnbieter(k)) do
17:        setzeAnbieter(gibAnbieter(naechsterAnbieter), k)
18:    end for
19: end while
```

dene Infrastruktur-Knoten, wie beispielsweise virtuelle Maschinen oder IoT-Devices, organisiert sind. Die Funktion *gibMoeglicheAnbieter* überprüft hierzu für jede dieser Deployment-Möglichkeiten, die im zugewiesenen Standort der Komponente verfügbar ist, ob die definierten Anforderungen der Komponente erfüllt werden. Hierfür wird die von Saatkamp et al. [SBK+18; SBKL17] eingeführte Funktionalität zum Abgleich von Anforderungen und Fähigkeiten wiederverwendet. Diese Funktion kann jedoch um eine beliebige Logik zur Auswahl von Anbietern für derzeit nicht berücksichtigte Anforderungen erweitert werden, zum Beispiel die erwarteten Kosten der benötigten Ressourcen oder Anforderungen bezüglich der Zuverlässigkeit [RHH11].

Wurde der Anbieter bereits der Variable *anbieterSet* hinzugefügt, wird die jeweilige Komponente in die Menge der unterstützten Komponenten für diesen Anbieter aufgenommen (Zeilen 6 bis 8). Andernfalls wird ein neues

Tupel für diesen Anbieter erstellt und nur die aktuelle Komponente wird der Menge der unterstützten Komponenten hinzugefügt (Zeilen 10). Anschließend werden die Komponenten einem Anbieter zugewiesen, bis alle Komponenten der Topologie eine gültige Zuweisung haben (Zeilen 14 bis 19). Dazu wird jeweils der nächste Anbieter ermittelt, der die meisten noch nicht zugeordneten Komponenten unterstützt (Zeile 15). Anschließend werden diese Komponenten dann dem ausgewählten Anbieter zugewiesen (Zeile 16 bis 18). Es werden also möglichst viele Komponenten demselben Anbieter zugeordnet, um von einer schnelleren internen Anbindung zu profitieren. Nachdem alle Komponenten einem Anbieter zugewiesen sind, kann die bis dahin noch unvollständige Topologie entsprechend den Ergebnissen dieses Algorithmus vervollständigt werden (vgl. Zeile 7 in Algorithmus 5.1).

5.4 Validierung und Evaluierung

Dieser Abschnitt validiert und evaluiert das in diesem Kapitel vorgestellte Konzept sowie insbesondere die dazugehörigen Algorithmen zur Verteilung von Anwendungskomponenten auf Basis des Datenflusses einer Anwendung. In Abschnitt 5.4.1 werden hierzu zunächst verschiedene beispielhafte Datenflussmodelle vorgestellt und die aus den Algorithmen resultierenden Verteilungen mit anderen Verteilungsmöglichkeiten verglichen. Anschließend wird die Ausführungszeit des Ansatzes in Abschnitt 5.4.2 evaluiert.

5.4.1 Beispiele und Validierung der Verteilung

Das vorgestellte Konzept sowie die in Abschnitt 5.3 vorgestellten Algorithmen zur Gruppierung und Platzierung von Komponenten auf Basis des Datenflusses einer Anwendung werden zum einen durch die in Kapitel 8 beschriebenen Prototypen validiert. Zum anderen werden im Folgenden verschiedene beispielhafte Datenflussmodelle vorgestellt und die, durch die Verwendung verschiedener Verteilungsmöglichkeiten resultierenden, Verteilungen der jeweils enthaltenen Komponenten aufgezeigt und miteinander verglichen.

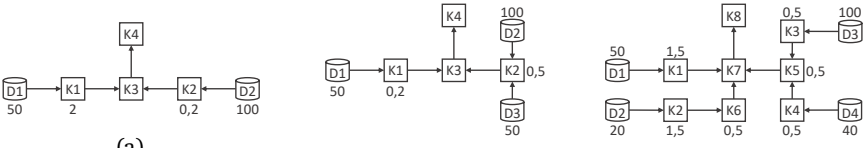
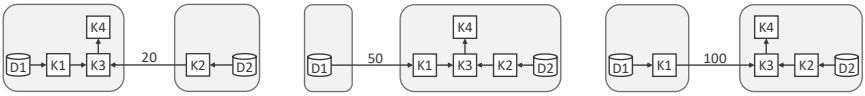


Abbildung 5.4: Verschiedene Datenflussdiagramme zur Veranschaulichung und Validierung der Verteilung der einzelnen Komponenten

Die drei hierzu genutzten Datenflussmodelle sind in Abbildung 5.4 dargestellt. Die Datenquellen sind pro Datenflussmodell jeweils mit $D1, ..., Dn$ bezeichnet, wobei n die Menge an enthaltenen Datenquellen angibt. Die zu verteilenden Komponenten sind entsprechend jeweils mit $K1, ..., Km$ bezeichnet, wobei m die Menge an enthaltenen Komponenten angibt. Die Datenflussmodelle enthalten weiterhin die Datenmengen der einzelnen Datenquellen. Diese sind jeweils unter beziehungsweise über der jeweiligen Datenquelle angegeben. Da insbesondere das Verhältnis der jeweils zu übertragenden Datenmenge der verschiedenen Verteilungsmöglichkeiten miteinander verglichen wird, kann hier zur Vereinfachung auf die Angabe einer konkreten Dateneinheit verzichtet werden. Die Datenfaktoren der einzelnen Komponenten werden ebenfalls jeweils unter, über oder neben der jeweiligen Komponente angeben. Falls ein Datenfaktor nicht explizit angegeben ist, wird dieser als neutral (Datenfaktor = 1) angenommen.

Da verglichen werden soll, wie groß die Datenmenge ist, die in den jeweiligen resultierenden Verteilungen zwischen verschiedenen Standorten übertragen werden muss, wird angenommen, dass die einzelnen Datenquellen bereits an einem, jeweils separaten, Standort betrieben werden und dementsprechend diesem bereits zugewiesen sind. Da weiterhin, falls keine oder nur eine Datenquelle oder Komponente bereits an einem Standort platziert ist, alle weiteren Komponenten dem bereits genutzten Standort zugeteilt werden würden (vgl. Algorithmus 5.2), wird dieser Fall hier nicht genauer betrachtet. Um außerdem ausschließlich die reine Verteilung der Komponenten auf Basis des Datenflusses zu betrachten, wird die im Konzept dieses Kapitels



(a) Datenfluss-Verteilung (Übertragung: 20)

(b) Monopol-Verteilung (Übertragung: 50)

(c) Gleich-Verteilung (Übertragung: 100)

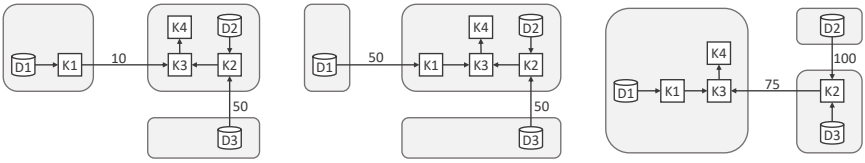
Abbildung 5.5: Resultierende Verteilungen der Komponenten des in Abbildung 5.4a dargestellten Datenflussdiagramms sowie die zwischen den beiden Standorten zu übertragende Datenmenge

vorgestellte, optionale Möglichkeit zur Definition von Anforderungen hier nicht genutzt (siehe hierfür Kapitel 7). Aus gleichem Grund wird weiterhin auch die Erstellung beziehungsweise Vervollständigung einer Topologie sowie die hierzu notwendige Zuweisung von Komponenten, an eine konkrete und am jeweils zugewiesenen Standort verfügbare Infrastruktur- oder Plattformkomponente, im Folgenden nicht näher betrachtet.

In Abbildung 5.5 werden die drei unterschiedlichen resultierenden Verteilungen der Komponenten des in Abbildung 5.4a vorgestellten Datenflussmodells einer Anwendung dargestellt. Die grau hinterlegten Umrandungen markieren dabei jeweils einen Standort und geben damit an, welche Komponenten diesem jeweils zugeteilt wurden. Weiterhin ist für jede betrachtete Verteilungsmöglichkeit, die jeweils zwischen den verschiedenen Standorten zu übertragende Datenmenge angegeben.

In Abbildung 5.5a wird die resultierende Verteilung dargestellt, welche mit den in Abschnitt 5.3 vorgestellten Algorithmen erzielt wird. Da diese Verteilung auf Basis des Datenflusses geschieht, wird diese Art der Verteilung in den Abbildungen als *Datenfluss-Verteilung* bezeichnet. Die zwischen den beiden Standorten zu übertragende Datenmenge ist hier 20.

In Abbildung 5.5b wird die resultierende Verteilung dargestellt, welche erreicht wird, wenn alle Komponenten dem Standort zugeteilt werden, der bereits die größte Datenquelle beinhaltet. Der Hintergrundgedanke bei dieser Art der Verteilung ist, dass an dem Standort mit der größten Datenquelle, auch potenziell am meisten Daten übertragen werden müssen, weswegen



(a) Datenfluss-Verteilung (Übertragung: 60)

(b) Monopol-Verteilung (Übertragung: 100)

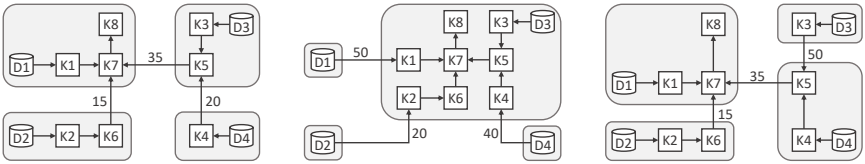
(c) Gleich-Verteilung (Übertragung: 175)

Abbildung 5.6: Resultierende Verteilungen der Komponenten des in Abbildung 5.4b dargestellten Datenflussdiagramms sowie die zwischen den Standorten zu übertragenden Datenmengen

die weiteren Komponenten auch möglichst hier platziert werden sollten. Diese Art der Verteilung beschreibt damit ein naives, aber grundsätzlich nachvollziehbares Vorgehen, insbesondere wenn die Verteilung manuell durchgeführt wird. In den Abbildungen wird diese Art der Verteilung als *Monopol-Verteilung* bezeichnet. Die zwischen den beiden Standorten zu übertragende Datenmenge ergibt hier 50.

In Abbildung 5.5c wird die resultierende Verteilung dargestellt, wenn naiv alle Komponenten gleichmäßig an die verschiedenen Standorte verteilt werden sollen, um somit eine möglicherweise durchschnittliche Übertragungsmenge von Daten zu erreichen. Für das hier betrachtete Datenflussmodell bedeutet das, dass $K1$ dem Standort von $D1$ und $K2$ dem Standort von $D2$ zugewiesen wird. Da $K3$ die nächstfolgende Komponente für beide Standorte darstellt, wird diese einem der möglichen Standorte beliebig zugeteilt (hier $D2$). Diese Art der Verteilung wird in den Abbildungen als *Gleich-Verteilung* bezeichnet und hat für das betrachtete Datenflussmodell eine, zwischen den Standorten, zu übertragende Datenmenge von 100 zur Folge.

Abbildung 5.6 stellt die verschiedenen Verteilungen der Komponenten für das in Abbildung 5.4b abgebildete Datenflussmodell dar. Für die in Abbildung 5.6a abgebildete Datenfluss-Verteilung ergibt sich hier eine, zwischen den verschiedenen Standorten, zu übertragende Gesamtdatenmenge von 60. Für die in Abbildung 5.6b abgebildete Monopol-Verteilung ergibt sich



(a) Datenfluss-Verteilung (Übertragung: 70)

(b) Monopol-Verteilung (Übertragung: 110)

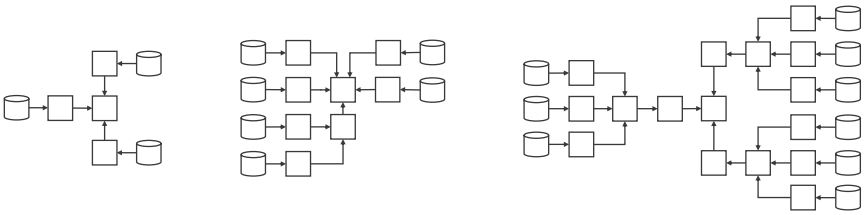
(c) Gleich-Verteilung (Übertragung: 100)

Abbildung 5.7: Resultierende Verteilungen der Komponenten des in Abbildung 5.4c dargestellten Datenflussdiagramms sowie die zwischen den Standorten zu übertragenden Datenmengen

eine Datenmenge von 100 und für die in Abbildung 5.6c abgebildete Gleich-Verteilung eine zu übertragende Datenmenge von 175.

In Abbildung 5.7 werden weiterhin die verschiedenen Verteilungen der Komponenten für das in Abbildung 5.4c abgebildete Datenflussmodell dargestellt. Hier ergibt sich für die in Abbildung 5.7a abgebildete Datenfluss-Verteilung eine zu übertragende Gesamtdatenmenge von 70. Für die in Abbildung 5.7b abgebildete Monopol-Verteilung ergibt sich hier eine Datenmenge von 110. Die in Abbildung 5.7c abgebildete Gleich-Verteilung ergibt eine, zwischen den verschiedenen Standorten, zu übertragende Datenmenge von 100.

Die konkreten Ergebnisse sowie dementsprechend die entstehenden Differenzen zwischen den drei Verteilungsstrategien bezüglich der, zwischen den verschiedenen Standorten, zu übertragenden Datenmenge sind von verschiedenen Faktoren abhängig, insbesondere von der Menge an enthaltenen und bereits platzierten Datenquellen, der jeweils darin enthaltenen Datenmenge sowie den Datenfaktoren der einzelnen Komponenten. Daher lässt sich keine quantitative Aussage darüber treffen, wie viel besser die Datenfluss-Verteilung im Vergleich mit den beiden anderen naiven, aber einfacher manuell anzuwendenden Verteilungsstrategien ist. Auf qualitativer Ebene lässt sich jedoch festhalten, dass die auf dem Datenfluss basierende Verteilung der Komponenten im Vergleich bessere Ergebnisse liefert.



(a) (b) (c)

Abbildung 5.8: Zur Evaluierung genutzte Datenflussmodelle

5.4.2 Evaluierung

Die in Abschnitt 5.2 vorgestellte Methode ermöglicht eine automatisierte Verteilung der Komponenten auf Basis des Datenflusses der Anwendung sowie die Vervollständigung der Topologie. Mithilfe des in Kapitel 8 vorgestellten Prototyps wird in diesem Abschnitt die Gesamtdauer zur Verteilung sowie zur Vervollständigung der Topologie gemessen. Hierfür werden drei unterschiedlich große Datenflussmodelle in jeweils drei verschiedenen Szenarien, also insgesamt neun verschiedene Fälle, betrachtet.

Das Datenflussmodell der ersten beispielhaften Anwendung enthält 3 Datenquellen sowie 4 zu verteilende Komponenten. Das zweite Datenflussmodell enthält 6 Datenquellen sowie 8 Komponenten und das dritte Datenflussmodell enthält 9 Datenquellen sowie 16 Komponenten. Die Datenflussmodelle sind schematisch in Abbildung 5.8 abgebildet. Da hier die benötigte Gesamtlaufzeit zur Verteilung und Vervollständigung und nicht die konkrete Verteilung der einzelnen Komponenten betrachtet werden soll, wird hier entsprechend auf die Angabe von Datenmengen sowie Datenfaktoren verzichtet. Da die für die Platzierung der Komponenten benötigte Zeit auch insbesondere von der Anzahl der Möglichkeiten abhängt, wo die Komponenten platziert werden können, werden drei verschiedene Szenarien betrachtet. Ein Szenario mit 5 Infrastruktur- oder Plattformkomponenten, wie beispielsweise Cloud-Anbieter oder Hyperscaler, die jeweils 2 Standorte unterstützen. Ein zweites Szenario mit 10 Infrastruktur- oder Plattformkomponenten,

Tabelle 5.1: Benötigte Zeit zur Gruppierung und Platzierung der Komponenten sowie Vervollständigung der Topologien

	Datenfluss-modell (a) (3D & 4K)*	**Datenfluss-modell (b) (6D & 8K)***	**Datenfluss-modell (c) (9D & 16K)***
5 Anbieter mit je 2 Standorten	494 ms	513 ms	524 ms
10 Anbieter mit je 10 Standorten	1043 ms	1080 ms	1148 ms
100 Anbieter mit je 10 Standorten	6117 ms	7052 ms	7169 ms

* D = Datenquellen; K = Zu platzierende Komponenten

die jeweils 10 Standorte unterstützen, sowie ein drittes Szenario mit 100 Infrastruktur- oder Plattformkomponenten, die jeweils 10 Standorte unterstützen. Im ersten Szenario gibt es somit 10 potenzielle Platzierungsmöglichkeiten für jede Komponente, im zweiten Szenario gibt es 100 potenzielle Platzierungsmöglichkeiten und im dritten Szenario gibt es 1000 potenzielle Platzierungsmöglichkeiten für jede Komponente der Anwendungen.

Die Ergebnisse der Messungen[1] sind in Tabelle 5.1 dargestellt. Es wurde der Median auf der Grundlage von jeweils 10 Messungen pro Fall berechnet. Die Tabelle zeigt, dass vor allem die Anzahl der möglichen Infrastruktur- oder Plattformkomponenten und Standorte einen großen Einfluss auf die benötigte Durchführungszeit hat. Der Ansatz ist jedoch insbesondere für den Einsatz bei datenintensiven Anwendungen, zum Beispiel mit mehreren Terabyte an zu versendenden Daten, ausgelegt. Im Vergleich zur angenommenen Gesamtlaufzeit einer solchen Anwendung, ist die initial benötigte Zeit zur Bestimmung der Verteilung der Komponenten und Vervollständigung der Topologie daher als vernachlässigbar zu betrachten. Die Ergebnisse zeigen weiterhin, dass die gemessenen Zeiten durch die automatisierte Verteilung sowie Vervollständigung, im Vergleich zu einer möglichen manuellen Durchführung, eine deutliche Zeitersparnis versprechen.

[1] Messumgebung: Windows 10 64 Bit, i7-4710MQ @ 3.5 GHz CPU und 16 GB RAM

5.5 Zusammenfassung und Diskussion

In diesem Kapitel wurde ein praktischer Ansatz für die automatisierte Verteilung von Komponenten einer datenverarbeitenden Anwendung vorgestellt, unter anderem auf der Grundlage des Datenflusses der Anwendung sowie weiteren Faktoren, wie beispielsweise Anforderungen der einzelnen Komponenten bezüglich des Deployments. Da der Ansatz nicht nur im Gesamtrahmen dieser Arbeit und insbesondere der in Abschnitt 3.2 vorgestellten Methode anwendbar ist, sondern auch alleinstehend angewandt werden kann, wurde zunächst eine Methode zur systematischen Modellierung, Gruppierung und Platzierung der Komponenten einer solchen datenverarbeitenden Anwendung vorgestellt. Anschließend wurden die Algorithmen zur automatisierten Umsetzung dieser Methode im Detail betrachtet.

Der vorgestellte Ansatz hat das Ziel, die Effizienz und Leistung verteilter Anwendungen zu verbessern, indem datenverarbeitende Komponenten so nah wie möglich an den zu verarbeitenden Daten platziert werden und dadurch die Übertragung von großen Datenmengen zwischen verschiedenen Standorten verhindert werden kann. Der Ansatz wird durch eine prototypische Implementierung sowie einem Vergleich, bezüglich der zwischen verschiedenen Standorten zu übertragenden Datenmenge, mit zwei weiteren Verteilungsmöglichkeiten validiert. Weiterhin wurde der Prototyp (siehe Kapitel 8) in verschiedenen Szenarien evaluiert und die Auswirkung der Größe der Datenflussmodelle sowie der Anzahl an verfügbaren Infrastruktur- und Plattformkomponenten auf die Laufzeit des Prototyps untersucht.

In der vorgestellten Methode (siehe Abschnitt 5.2) wird grundsätzlich davon ausgegangen, dass mit der Modellierung eines Datenflussmodells begonnen wird und basierend darauf dann zunächst eine (unvollständige) Topologie generiert wird. Dies ist insbesondere bei neu zu modellierenden sowie entwickelnden Anwendungen der zu empfehlende Ansatz. Allerdings kann die in diesem Kapitel beschriebene Methode sowie der grundlegende Ansatz darüber hinaus auch sowohl bei einer direkten Modellierung der Topologie als auch bei einer bereits bestehenden Topologie angewandt werden. In die-

sen Fällen können die benötigten Informationen bezüglich des Datenflusses der Anwendung, beispielsweise die Datenmengen und Datenfaktoren, auch manuell in den Topologien annotiert werden. Weiterhin müssen bei einer bereits bestehenden und vollständigen Topologie gegebenenfalls zusätzlich die in der Topologie enthaltenen Stacks gesplittet werden, um die jeweiligen Komponenten entsprechend frei verteilen zu können. Hierzu können Methoden und Algorithmen eingesetzt werden, wie sie beispielsweise von Saatkamp et al. [SBKL17; SBKL19] beschrieben werden.

Kapitel 6

Verteilbare und kommunizierende Anwendungskomponenten

Im vorherigen Kapitel wurde ein Konzept zur Verteilung von Komponenten auf Basis des Datenflusses ihrer Gesamtanwendung vorgestellt. Um dieses Konzept allerdings anwenden zu können, müssen die Anwendungen dementsprechend komponentenbasiert aufgebaut sein. In diesem Kapitel wird daher ein Programmiermodell sowie eine darauf aufbauende Entwicklungsmethode vorgestellt, um den Entwickler bei der Implementierung verteilbarer und interagierender Komponenten zur Realisierung von automatisch bereitstellbaren, verteilten Anwendungen zu unterstützen.

Die Grundidee dieses Ansatzes ist dabei die Abstraktion der Kommunikation und des Endpunktaustausches zwischen den interagierenden Komponenten sowie die Generierung von benötigten Code-Fragmenten auf Basis der in einem D^3M angegebenen Informationen, wie beispielsweise Identifikatoren

und Schnittstellenbeschreibungen der Komponenten. Durch die Verwendung eines D^{3}Ms wird dadurch nicht nur die Planung und Realisierung einer verteilten Anwendung erleichtert, sondern weiterhin das automatisierte Deployment der Anwendung ermöglicht. Die in diesem Kapitel vorgestellten Ansätze tragen somit zur Realisierung des dritten, vierten und neunten Schrittes der in Abschnitt 3.2 vorgestellten Shipping-Methode bei. Weiterhin stellt dieses Kapitel Forschungsbeitrag 4 der vorliegenden Arbeit dar.

Dieses Kapitel ist wie folgt gegliedert: Abschnitt 6.1 diskutiert zunächst die Motivation für diesen Forschungsbeitrag. In Abschnitt 6.2 wird die Idee und das Grundkonzept dieses Forschungsbeitrags beschrieben. Anschließend wird in Abschnitt 6.3 das D^{3}M-basierte Programmiermodell zur Abstraktion der Kommunikation zwischen Komponenten sowie dem Endpunktaustausch während der Bereitstellung vorgestellt. In Abschnitt 6.4 wird die Verwendung des Programmiermodells im Gesamtkontext der Entwicklung einer verteilten Anwendung vorgestellt und gezeigt, wie auf Basis einer D^{3}M-basierten Topologie, hierbei Code-Generierung zur Unterstützung des Entwicklers eingesetzt werden kann. In Abschnitt 6.5 wird die Arbeitsweise des Code-Generators sowie die Verwendung von Plugins zur Code-Generierung erläutert. Abschließend fasst Abschnitt 6.6 dieses Kapitel zusammen und ordnet den Forschungsbeitrag in den Gesamtkontext dieser Arbeit ein.

6.1 Motivation

Im Vergleich zu beispielsweise einem Monolith, der vergleichsweise einfach bereitgestellt und betrieben werden kann, ist der Aufwand und die Komplexität bei verteilten Anwendungen bezüglich des Deployments und Managements der einzelnen Komponenten deutlich höher [DGL+17; KMM18; VGC+15]. So müssen die Komponenten solcher Anwendungen in der Lage sein, miteinander zu kommunizieren. Die größten Herausforderungen bei komplexen heterogenen Anwendungen sind daher aus Entwicklersicht, neben dem Deployment, die Orchestrierung, Konfiguration sowie Verbindung (*Binding*) der verschiedenen Komponenten [BBK+13a; BSG+15; CZL+18].

Die konkrete Durchführung dieser Aufgaben hängt unter anderem von den Technologien ab, die zur Realisierung der Anwendung und ihrer Komponenten verwendet werden [BBK+13a]. Abhängig von diesen Technologien, beispielsweise genutzte Middleware, Programmiersprachen und Bereitstellungstechnologien, müssen während der Bereitstellung der Gesamtanwendung verschiedene Informationen ausgetauscht werden, um die Kommunikation zwischen den Komponenten zu ermöglichen [ZBL17; ZBL18]. Solche Endpunktinformationen, die für die Kommunikation zwischen den Komponenten erforderlich sind, sind beispielsweise URLs von Diensten, Servicenamen, IP-Adressen und gegebenenfalls erforderliche Anmeldeinformationen.

Das Problem der automatischen Verbindung von Anwendungskomponenten kann durch den Einsatz einer einheitlichen Technologie, wie beispielsweise Docker Compose[1], Docker Swarm[2] oder Kubernetes[3], gelöst werden. Typischerweise bieten solche Technologien integrierte Funktionalitäten zur Orchestrierung, die bei der Implementierung der Komponenten berücksichtigt werden müssen. Die Umsetzung geschieht zum Beispiel durch die Weitergabe von Umgebungsvariablen an die einzelnen Container oder durch die gemeinsame Nutzung von Konfigurationsdateien, die zum Aufbau der Verbindungen zwischen den Komponenten verwendet werden [BGO+16].

In komplexen verteilten Cloud-Anwendungen, die aus mehreren heterogenen Komponenten bestehen, müssen jedoch typischerweise mehrere Technologien kombiniert werden, insbesondere wenn auch physische Geräte in IoT-Szenarien beteiligt sind [BBK+13a]. Dementsprechend erfordern solche Szenarien die Kombination aus unterschiedlichen Protokollen und Mechanismen für den Austausch der Endpunktinformationen sowie zum Aufruf der einzelnen Komponenten. Dies führt zu individuell geschriebenem Code, beispielsweise Konfigurationsskripten, was die Portabilität der Anwendungen einschränkt und weiterhin den Aufwand sowie die Komplexität bei der Implementierung der Komponenten erhöht [WBB+14a; WBB+14b].

[1] https://docs.docker.com/compose/
[2] https://docs.docker.com/engine/swarm/
[3] http://kubernetes.io/

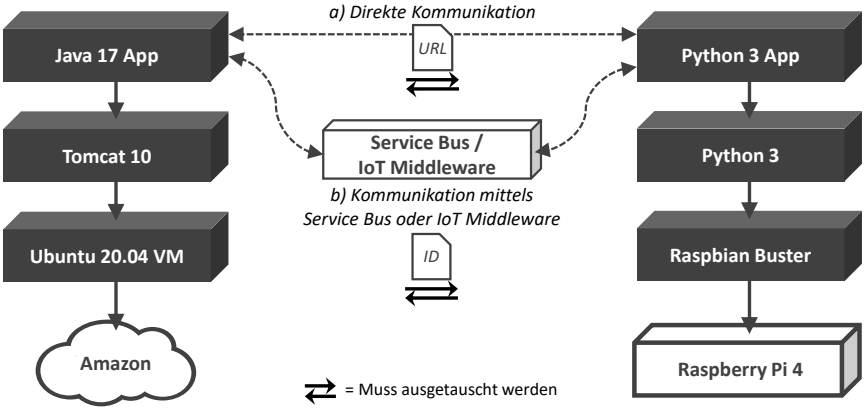


Abbildung 6.1: Möglichkeiten zur Orchestrierung eines beispielhaften Cloud und IoT Szenarios

Eine weitere Möglichkeit ist die Verwendung eines Enterprise Service Bus (ESB) [Cha04] oder, im Falle von Cyber-physischen Szenarien, einer IoT-Middleware [GBF+16] zur Abstraktion der Kommunikation. Hierbei muss die aufrufende Komponente nur den Endpunkt des Service Bus beziehungsweise der Middleware und nicht den konkreten Endpunkt der anderen Komponente kennen. Weiterhin kann der Service Bus weitere Funktionalitäten wie die Wahl des benötigten Kommunikationsprotokolls oder die Umwandlung in ein gefordertes Datenformat übernehmen. Um eine konkrete Komponente aufrufen zu können, müssen allerdings auch bei der Nutzung eines Service Bus Informationen an ihn übermittel werden um die gewünschte Komponente identifizieren und auswählen zu können. Im Falle einer IoT-Middleware, wie zum Beispiel einem Nachrichtenbroker, muss die ID eines Topics dem Sender sowie dem Empfänger im Voraus bekannt sein [Nai17].

Der Austausch solcher IDs ist jedoch technisch ähnlich wie der Austausch von Endpunkten der aufzurufenden Komponenten, beispielsweise URLs unter denen die Komponenten erreichbar sind. In Abbildung 6.1 wird die Problematik des notwendigen Austausches von Endpunktinformationen zur Verbindung von Komponenten anhand eines Szenarios veranschaulicht. In

dem dargestellten Szenario soll die rechts abgebildete Komponente *Python 3 App*, die auf einem Raspberry Pi 4[1] betrieben wird, Daten an die in der Cloud auf Amazon EC2[2] betriebene Komponente *Java 17 App* senden, die für die Auswertung und Anzeige dieser Daten verantwortlich ist.

Die Abbildung veranschaulicht zwei Möglichkeiten die Komponenten zu verbinden: (i) per direkter Kommunikation und (ii) per Kommunikation über einen Service Bus oder einer anderen (IoT-)Middleware. Beide Varianten erfordern allerdings den Austausch von Endpunktinformationen: Entweder muss die Komponente *Python 3 App* (i) im Falle einer direkten Kommunikation, den Endpunkt der Komponente *Java 17 App*, zum Beispiel eine URL, kennen oder (ii) im Falle der Nutzung eines Service Bus beziehungsweise einer IoT-Middleware, eine ID zur Identifikation der Komponente *Java 17 App* beziehungsweise eines Topics kennen. Bei Verwendung des Service Bus muss die Komponente *Java 17 App* sich außerdem zusätzlich zuerst am Service Bus registrieren, um sich ihm bekannt zu machen. Weiterhin muss sie sich, im Falle der Verwendung einer IoT-Middleware, auf das entsprechend Topic abonnieren, um die dort veröffentlichten Daten zu erhalten.

Da diese Problematik erheblich von den verwendeten (i) Programmiersprachen, (ii) Middleware-Technologien und (iii) Bereitstellungstechnologien abhängt, führt dies dazu, dass für jede Komponente individueller Code geschrieben werden muss, um den Austausch der erforderlichen Endpunktinformationen zu realisieren. Dies ist einerseits mit zusätzlichem Aufwand, neben der Implementierung der eigentlichen Geschäftslogik, verbunden und erhöht zudem bei manueller Durchführung die Fehleranfälligkeit. Um diese Probleme zu adressieren, stellt dieses Kapitel ein Programmiermodell zur Abstraktion der Kommunikation zwischen heterogenen Komponenten vor. Orientiert an Ferguson et al. [FS05] beschreibt ein Programmiermodell im Rahmen dieser Arbeit dabei die grundlegenden Eigenschaften, Konzepte, Abstraktionen und Artefakte, die von einem Entwickler zur Lösung eines konkreten Problems angewandt bzw. verwendet werden können.

[1] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[2] https://aws.amazon.com/de/ec2/

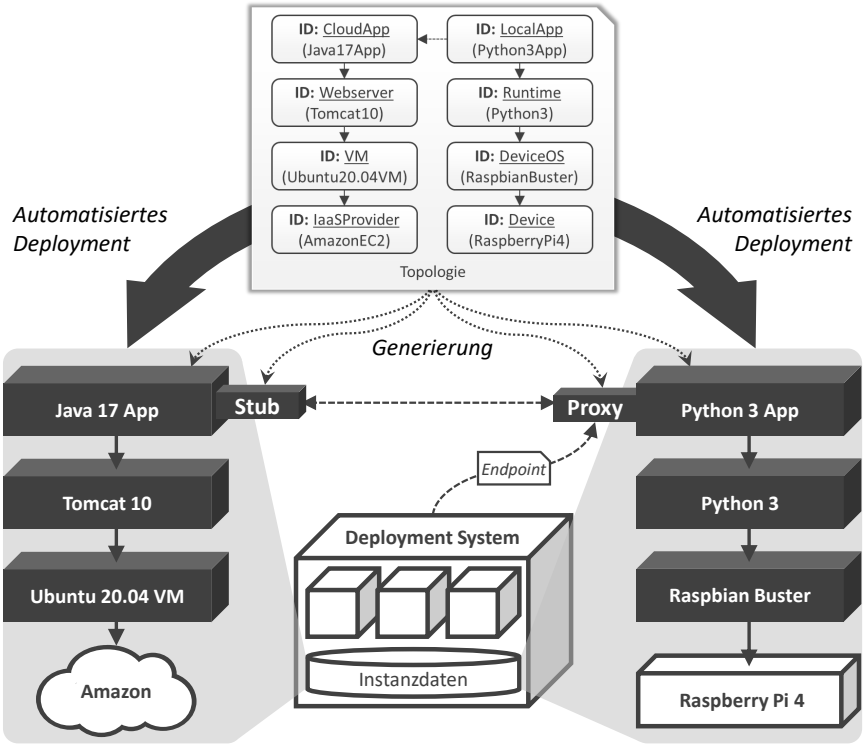


Abbildung 6.2: Grundkonzept zur Umsetzung des Programmiermodells

6.2 Idee und Grundkonzept

Die Hauptziele des in diesem Kapitel vorgestellten Konzepts sind es, (i) den Austausch von benötigten Endpunktinformationen während des Deployments der Komponenten sowie (ii) die Kommunikation zwischen den verschiedenen Komponenten zur Laufzeit vollständig zu abstrahieren und damit (iii) den Entwickler bei der Entwicklung und Implementierung der entsprechenden Komponenten einer verteilten Anwendung zu unterstützen. Ein wichtiger Bestandteil zur Realisierung dieses Beitrags ist dabei insbesondere die Generierung von zur Kommunikation benötigten Code-Fragmenten.

So kann durch die Nutzung von Techniken aus dem Bereich der modellgetriebenen Softwareentwicklung (engl.: model-driven software development, MDSD), die Entwicklungszeit einer Anwendung, unter anderem durch die Generierung von Code, beschleunigt werden [Sel03; SVC06]. Um eine nutzbare Modellierung von komplexen verteilten Cloud- oder IoT-Anwendungen zu erhalten, müssen hierzu jedoch einige Aspekte beschrieben werden. So müssen beispielsweise die einzelnen Komponenten der Anwendung sowie deren Beziehungen zueinander beschrieben werden. Zur Code-Generierung werden darüber hinaus Beschreibungen der formalen Schnittstellen der Komponenten benötigt. Weiterhin müssen die benötigte Infrastruktur zum Betrieb dieser Komponenten sowie Eigenschaften modelliert werden. Zur Modellierung dieser Aspekte können entweder mehrere unterschiedliche Modellierungssprachen verwendet werden, was zur Durchführung jedoch Fachwissen über jede dieser Modellierungssprachen voraussetzt, oder das in dieser Arbeit vorgestellte D^3M-Metamodell genutzt werden. Dieses ermöglicht die Modellierung aller dieser Aspekte innerhalb eines Modells. Dadurch kann der Entwickler beim Entwicklungs- und Bereitstellungsprozess, vom Beginn der Modellierung der Anwendung über die Implementierung der Komponenten bis hin zur Bereitstellung der Anwendung, unter anderem durch die Generierung von Code, einer automatisierten Bereitstellung sowie Möglichkeiten zur Verwaltung der Anwendung unterstützt werden.

Die Grundidee sowie eine vereinfachte Architekturübersicht eines beispielhaften Deployment Systems zur Realisierung dieses Konzepts ist in Abbildung 6.2 dargestellt. Ausgangspunkt ist eine modellierte Topologie der verteilten Anwendung. Basierend auf dieser und insbesondere den darin beschriebenen Komponenten sowie deren Schnittstellen können verschiedene Code-Fragmente generiert werden, die den Entwickler bei der Realisierung der Anwendung, beziehungsweise der Implementierung der einzelnen Komponenten unterstützen [ZBL17; ZBL18]. So kann unter anderem jeweils die Grundstruktur einer Komponente, sogenannte Skeletons, basierend auf der Topologie generiert werden. Diese kann anschließend vom Entwickler mit der beabsichtigten Geschäftslogik ausimplementiert werden.

Weiterhin können auf Basis der Schnittstellenbeschreibungen der Komponenten Proxies sowie Stubs zur Vereinfachung und Abstraktion der Kommunikation zwischen den verschiedenen Komponenten generiert werden. Aus Sicht der aufrufenden Komponente macht es dadurch keinen Unterschied, ob die aufzurufende Komponente lokal verfügbar ist oder an einem entfernten Standort betrieben wird, da der entsprechende Komponentenaufruf vom Proxy der aufrufenden Komponente an den Stub der aufzurufende Komponente weitergeleitet wird. Dementsprechend wird durch diesen Ansatz eine Verteilungstransparenz sowie Ortstransparenz erreicht. Dieses Vorgehen entspricht weiterhin dem Proxy-Entwurfsmuster [GHJV94] und reduziert den Aufwand des Entwicklers, da das manuelle Implementieren der Kommunikationsfunktionalität der Komponenten dadurch entfällt.

Ähnliche Ansätze zur Realisierung einer verteilten Kommunikation, unter anderem durch die Generierung von Code auf Basis von Schnittstellenbeschreibungen, wurden bereits in anderen Technologien, wie beispielsweise CORBA[1], DCOM[2], RMI[3] sowie gRPC[4] eingesetzt. Der Vorteil des in diesem Kapitel beschriebenen Ansatzes ist es jedoch, dass unter anderem die Modellierung der Struktur der Gesamtwendung sowie die Beschreibung der Schnittstellen der einzelnen Komponenten der Anwendung innerhalb eines einheitlichen Modells vorgenommen werden kann. Dadurch kann, wie in Abbildung 6.2 dargestellt, nicht nur die Kommunikation zwischen den Komponenten, durch die Generierung von entsprechendem Code, abstrahiert werden, sondern auch das Binding-Problem, durch automatisiertes Deployment mit einem entsprechendem Deployment System, behoben werden.

Hierfür muss das genutzte Deployment System eine Komponente zur Erfassung und Speicherung von Instanzdaten besitzen. Diese Instanzdaten beinhalten Informationen über die bereitgestellten Anwendungen, beispielsweise unter welchem Endpunkt eine Komponente einer Anwendung erreich-

[1] https://www.omg.org/spec/CORBA/3.4/About-CORBA
[2] https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom
[3] https://www.oracle.com/java/technologies/javase/remote-method-invocation-home.html
[4] https://grpc.io/docs/platforms/web/basics

bar ist. Da das Deployment System somit die Struktur einer Anwendung, insbesondere welche Komponenten miteinander verbunden sein sollen, sowie die zur Verbindung dieser Komponenten benötigten Endpunktinformationen kennt, kann das Binding der Komponenten im direkten Anschluss an das Deployment der Anwendung ebenfalls durch das Deployment System automatisiert durchgeführt werden.

6.3 Programmiermodell zur Abstraktion der Kommunikation

In Abbildung 6.3 wird das Programmiermodell zur Abstraktion der Kommunikation zwischen verteilten Komponenten einer Anwendung veranschaulicht. In der oberen Hälfte der Abbildung wird die modellierte Topologie der in Abschnitt 6.1 vorgestellten Anwendung dargestellt. Auf der linken Seite der Topologie ist die Komponente vom Typ *Java 17 App* mit der ID *CloudApp* sowie die für ihren Betrieb weiteren notwendigen Komponenten, wie beispielsweise ein Tomcat Webserver sowie eine virtuelle Maschine, abgebildet. Weiterhin wird die Schnittstellenbeschreibung der Komponente *CloudApp* wiedergegeben. So besitzt die Komponente in diesem Beispiel eine Schnittelle *StorageInterface*, welche eine Operation *store* beinhaltet. Als Eingabeparameter ist für diese Operation der Parameter *val* spezifiziert.

Die rechte Seite der Topologie zeigt die Komponente vom Typ *Python 3 App* mit der ID *LocalApp* sowie die weiteren für den Betrieb benötigten Komponenten, wie beispielsweise eine Python 3 Laufzeitumgebung. Im Gegensatz zur Komponente *CloudApp*, welche in der Cloud auf Amazon EC2 betrieben werden soll, ist für die Komponente *LocalApp* der Betrieb auf einem physischen IoT-Gerät, einem Raspberry Pi, vorgesehen. Die Hauptfunktion der Komponente *LocalApp* besteht darin, gesammelte Daten, beispielsweise Sensordaten wie Temperatur oder Luftdruck, an die Komponente *CloudApp* zur weiteren Verarbeitung zu senden, indem sie deren Operation *store* nutzt.

Die untere Hälfte der Abbildung veranschaulicht das physische Deployment der modellierten Anwendung. Die linke Seite deutet weiterhin eine vereinfachte und beispielhaft in Pseudocode implementierte Umsetzung der

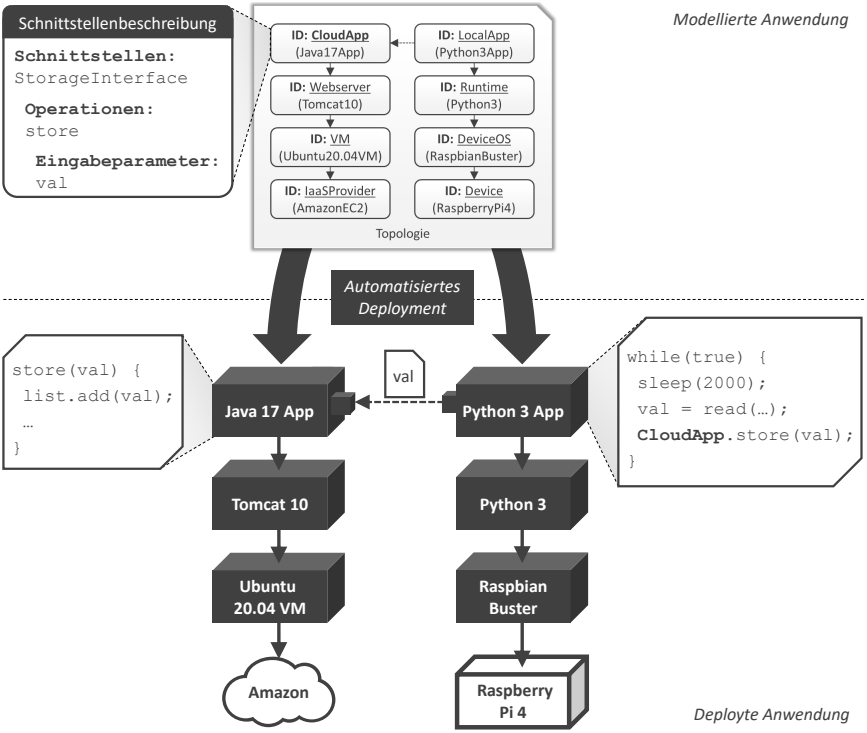


Abbildung 6.3: Programmiermodell zur Abstraktion des Austausches von Endpunktinformationen sowie der Kommunikation zwischen verteilten Komponenten einer Anwendung

Operation *store* an. Auf der rechten Seite ist ebenfalls ein in Pseudocode veranschaulichter Ausschnitt der Komponente *LocalApp* skizziert.

Das Ziel des Programmiermodells besteht darin, den Aufruf von Operationen, die von anderen Komponenten angeboten werden, ausschließlich auf der Grundlage der in der Topologie modellierten Informationen zu ermöglichen, sowie weiterhin durch das Proxy-Entwurfsmuster die Operation aufrufen zu können, als wäre sie lokal verfügbar. Um einen Operationsaufruf zu realisieren, wird daher die ID der aufzurufenden Komponente als Objekt im Code verwendet. Weiterhin kann die gewünschte Operation, wie in der

objektorientierten Programmierung üblich, aufgerufen werden. Der in der Abbildung dargestellte Code der Komponente *LocalApp* enthält daher den Befehl *CloudApp.store(val)* um die Operation *store*, der in der Topologie modellierten Komponente *CloudApp*, entsprechend mit dem Eingabeparameter *val* aufrufen zu können. Dies wird durch den generierten Proxy, welcher die Schnittstelle der Komponente *CloudApp* lokal anbietet, ermöglicht. Der Proxy enthält die Funktionalität, um die übergebenen Daten anschließend an den ebenfalls generierten Stub der Komponente *CloudApp* zu schicken. Der Stub übergibt die erhaltenen Daten dann wiederum, indem er die in der Komponente *CloudApp* implementierte Operation *store* aufruft.

Obwohl die Komponente *CloudApp* in der Cloud betrieben wird und die Komponente *LocalApp* auf einem physischen Gerät betrieben wird, kann dadurch die Operation *store* innerhalb der Komponente *LocalApp* verwendet werden, als ob es sich um eine lokal verfügbare Methode handeln würde. Da weiterhin alle IDs der Komponenten in der Topologie angegeben sind und, wie die Endpunkte der deployten Komponenten, dem genutzten Deployment System dementsprechend bekannt sind, entfällt der ansonsten benötigte Austausch von IDs oder anderen Endpunktinformationen durch hierfür individuell zu erstellenden Code. Somit wird insgesamt das Binding sowie die Kommunikation zwischen den Komponenten durch die Nutzung des Programmiermodells abstrahiert und der Entwickler dadurch bei der Implementierung der Anwendung unterstützt.

6.4 Entwicklung verteilbarer und kommunizierender Komponenten

In diesem Abschnitt wird die Methode zur systematischen Entwicklung von D^3M-basierten Anwendungen und deren Komponenten unter der Verwendung des zuvor erläuterten Programmiermodells vorgestellt. Weiterhin wird die Nutzung von Codegenerierung an konkreten Beispielen veranschaulicht und die Vorteile dieses Ansatzes aufgezeigt, um die Entwicklung sowie das Deployment einer solchen verteilten Anwendung zu vereinfachen.

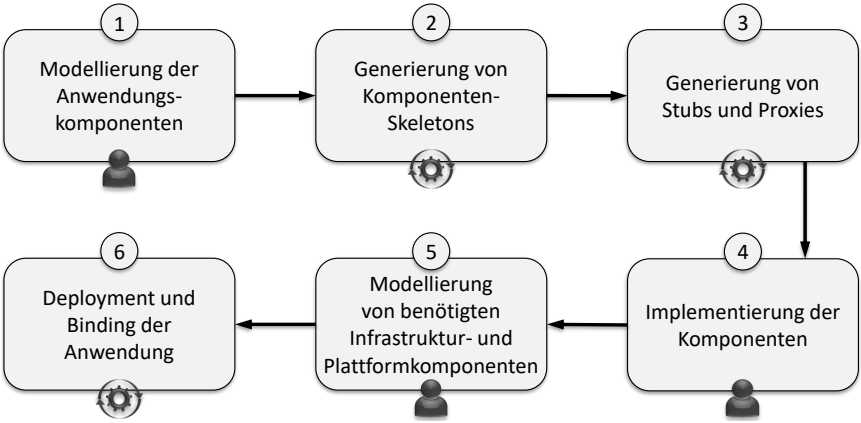


Abbildung 6.4: Methode zur systematischen Entwicklung von verteilten Anwendungen unter der Verwendung des Programmiermodells

6.4.1 Methode zur systematischen Entwicklung verteilbarer und kommunizierender Komponenten

Wie in Abschnitt 4.1 vorgestellt wurde, ermöglicht das D^3M-Metamodell unter anderem die Modellierung der Struktur einer Anwendung, bestehend aus den enthaltenen Komponenten und deren Beziehungen zueinander, sowie weiterhin die Beschreibungen der Managementschnittstellen als auch, im Gegensatz zu TOSCA [OAS13a; OAS13b], EDMM [WBF+19] oder DMMN [Bre16], zusätzlich die Beschreibung der Anwendungsschnittstellen dieser Komponenten. Managementschnittstellen enthalten Managementoperationen, die für die Bereitstellung und Verwaltung einer Komponente verantwortlich sind, zum Beispiel für die Installation sowie Konfiguration eines Webservers auf einer virtuellen Maschine. Dagegen enthalten die Anwendungsschnittstellen Anwendungsoperationen, welche die implementierte und nach außen hin anzubietende Geschäftslogik einer Komponente darstellen. Daher können Anwendungsoperationen erst dann genutzt werden, nachdem die Anwendung, mithilfe der entsprechenden Managementoperationen, erfolgreich bereitgestellt und in Betrieb genommen wurde.

Um den Prozess von der Modellierung, über die Entwicklung der einzelnen Komponenten, bis hin zum Deployment und der Inbetriebnahme einer solchen verteilten Anwendung systematisch durchführen zu können sowie insbesondere die Implementierung der Komponenten zu vereinfachen, wird in den nachfolgenden Abschnitten eine entsprechende Methode präsentiert, die zur Umsetzung das zuvor vorgestellte Programmiermodell verwendet.

Die Methode zur Entwicklung verteilbarer und kommunizierender Komponenten ist in Abbildung 6.4 dargestellt und besteht aus sechs Schritten. Im ersten Schritt der Methode werden die Komponenten der Anwendung sowie deren Anwendungsschnittstellen und Operationen durch den Modellierer mittels einer geeigneten Modellierungssprache, beispielsweise dem D^3M-Metamodell, modelliert. Im zweiten Schritt der Methode werden die Code-Skeletons der Komponenten auf Basis der zuvor modellierten Schnittstellenbeschreibungen generiert. Im dritten Schritt der Methode werden weiterhin die benötigten Stubs und Proxies für die entsprechenden Komponenten generiert. Im vierten Schritt der Methode werden die zuvor generierten Code-Skeletons vom Entwickler ausimplementiert. Im fünften Schritt der Methode werden die für den Betrieb der Anwendung erforderlichen Infrastruktur- und Plattformkomponenten modelliert um eine vollständige und ausführbare Topologie zu erstellen. Im sechsten und letzten Schritt der Methode wird die modellierte und implementierte Anwendung automatisiert mithilfe eines entsprechenden Deployment Systems bereitgestellt und die Komponenten zur Ermöglichung der Kommunikation miteinander verbunden. Im Folgenden werden die einzelnen Schritte der Methode erläutert.

6.4.2 Modellierung der Anwendungskomponenten und deren Schnittstellen

Im ersten Schritt modelliert der Modellierer die Komponenten der Anwendung sowie die Anwendungsschnittstellen mittels einer geeigneten Modellierungssprache. Bereits vorhandene, modellierte Komponenten und Topologien können wiederverwendet werden. Bei der Modellierung der Komponenten gibt es für diese dabei grundsätzlich jeweils drei Ausgangssituationen:

```
<NodeType name="xs:NCName">
  <ApplicationInterfaces>
    <Interface name="xs:NCName">
      <Operation name="xs:NCName">
        <documentation/> ?
        <InputParameters>
          <InputParameter name="xs:string" type="
              xs:string" required="yes|no"?/> +
        </InputParameters> ?
        <OutputParameters>
          <OutputParameter name="xs:string" type="
              xs:string" required="yes|no"?/> +
        </OutputParameters> ?
      </Operation> +
    </Interface> +
  </ApplicationInterfaces> ?
</NodeType>
```

Listing 6.1: Auszug aus dem Schema zur Beschreibung von Anwendungsschnittstellen einer Komponente

(i) Es kann eine bereits vorhandene, fertig ausmodellierte und implementierte Komponente wiederverwendet werden. (ii) Die Komponente ist bereits teilweise modelliert, insbesondere die zum Deployment benötigten Managementoperationen, aber es fehlen beispielsweise noch die Anwendungsschnittstellen. (iii) Die Komponente muss von Grund auf neu modelliert werden. Als vollständig modellierte Komponente gilt, falls Komponententyp, Eigenschaften, gegebenenfalls Fähigkeiten und Anforderungen, Managementschnittstellen sowie Anwendungsschnittstellen der Komponente modelliert sind. Ein Auszug aus dem Schema zur Beschreibung von Anwendungsschnittstellen ist in Listing 6.1 dargestellt. Die Modellierung von Fehlern wurde im Rahmen dieser Arbeit nicht genauer betrachtet. Hierfür müsste das Schema entsprechend angepasst werden. In Listing 6.2 wird weiterhin anhand des zuvor in Abbildung 6.3 vorgestellten Szenarios ein konkretes Beispiel dargestellt, wie die von einer Komponente angebotene Anwendungsschnittstelle definiert werden kann. So zeigt das Listing die Schnittstelle *StorageInter-*

```
<NodeType name="Java_17_App">
  <ApplicationInterfaces>
    <Interface name="StorageInterface">
      <Operation name="store">
        <documentation>
          Stores the passed value
        </documentation>
        <InputParameters>
          <InputParameter name="val" type="xs:int"/>
        </InputParameters>
      </Operation>
    </Interface>
  </ApplicationInterfaces>
</NodeType>
```

Listing 6.2: Definition der in Abbildung 6.3 beispielhaft dargestellten Anwendungsschnittstelle der Komponente *Java 17 App*

face, deren Operation *store*, einen Beschreibungstext der Operation sowie den Eingabeparameter *val*. Ausgabeparameter können analog spezifiziert werden. Basierend auf dieser Modellierung können im nächsten Schritt Code-Skeletons für diese Komponente generiert werden sowie im übernächsten Schritt zusätzlich Stubs und Proxies generiert werden.

6.4.3 Generierung von Skeletons zur Vereinfachung der Implementierung der Komponenten

Der zweite Schritt der Methode ist die Generierung von Code-Skeletons auf der Grundlage der zuvor modellierten Topologie. Da das D^3M-Metamodell sowie das Programmiermodell eine abstrakte und technologieunabhängige Modellierung der Anwendungskomponenten ermöglicht, müssen Code-Skeletons für beliebige Programmiersprache generiert werden können. Für den Entwickler ergeben sich somit keine Einschränkungen hinsichtlich der zur Implementierung nutzbaren Programmiersprachen. Vielmehr kann die Entwicklungszeit mittels Code-Generierung verkürzt werden.

Algorithmus 6.1 generiereSkeleton(top)

```
1: // top => Topologie, für deren Komponenten Skeletons generiert
2:                                                    werden sollen
3: for all (k ∈ komponenten(top)) do
4:     if (skeletonSollGeneriertWerden(k)) then
5:         for all (as ∈ anwendungsschnittstellen(k)) do
6:             asName := gibNameAs(as)
7:             erstelleDatei(asName)
8:             for all (ao ∈ anwendungsoperationen(as)) do
9:                 aoName := gibNameAo(ao)
10:                doku := gibDokumentation(ao)
11:                eingabePs := gibEingabeParameter(ao)
12:                ausgabePs := gibAusgabeParameter(ao)
13:                erstelleSignatur(aoName, doku, eingabePs, ausgabePs)
14:            end for
15:        end for
16:    end if
17: end for
```

Algorithmus 6.1 stellt den grundsätzlichen Algorithmus zur Generierung eines solchen Code-Skeletons dar. Dem Algorithmus übergeben wird eine Topologie mit entsprechend modellierten Komponenten, deren Skeletons generiert werden sollen. Anschließend wird über diese Menge an Komponenten iteriert (Zeile 3). Falls Skeletons für die jeweilige Komponente generiert werden sollen (Zeile 4), wird anschließend über die definierten Anwendungsschnittstellen dieser Komponente iteriert (Zeile 5). Da bei bereits implementierten Komponenten keine Code-Skeletons generiert werden müssen, kann im Vorhinein festgelegt werden, ob Code-Skeletons für eine Komponente generiert werden sollen oder nicht. Für jede definierte Anwendungsschnittstelle wird dann eine entsprechende Datei mit dem Namen der Anwendungsschnittstelle erstellt (Zeilen 6 und 7). Anschließend wird über alle Anwendungsoperationen dieser Anwendungsschnittstelle iteriert (Zeile 8) und jeweils die Methodensignatur dazu, auf Basis der in der Topologie definierten Informationen, generiert (Zeilen 9 bis 13). In Listing 6.3 wird die beispielhafte Struktur eines solchen generierten Code-Skeletons dargestellt.

```
1  class [interfaceName] {
2
3    /**
4    * [documentation]
5    */
6    static [operationName]([InputParameter1],[
         InputParameter2],...) {
7      // TODO generated skeleton
8      return [OutputParameter]
9    }
10 }
```

Listing 6.3: Beispielhafte Struktur eines generierten Code-Skeletons für eine Anwendungsschnittstelle (der konkrete Aufbau sowie Syntax sind von der gewählten Programmiersprache abhängig)

Um verschiedene Programmiersprachen unterstützen zu können, müssen die Methoden *erstelleDatei* (Zeile 7) sowie *erstelleSignatur* (Zeile 13) in Algorithmus 6.1 je nach gewünschter Programmiersprache unterschiedlich implementiert sein. Um dies zu realisieren, kann unter anderem ein Plugin-System zur Umsetzung des Code-Generators genutzt werden, wie es auch beispielhaft in Abschnitt 6.5 vorgestellt wird. Listing 6.4 zeigt ein konkretes Beispiel für ein in Java generiertes Code-Skeleton, basierend auf der in Listing 6.2 definierten Anwendungsschnittstelle.

```
1  class StorageInterface {
2
3    /**
4    * Stores the passed value
5    */
6    static void store(int val) {
7      // TODO generated skeleton
8    }
9  }
```

Listing 6.4: Generiertes Code-Skeleton, der in Listing 6.2 definierten Anwendungsschnittstelle

6.4.4 Generierung von Stubs und Proxies zur Abstraktion der Kommunikation zwischen den Komponenten

Der dritte Schritt der Methode ist die Generierung von Stubs und Proxies. Der Ablauf hierfür ist grundsätzlich mit dem in Abschnitt 6.4.3 vorgestellten Ablauf vergleichbar: Es müssen die Anwendungsschnittstellen sowie darin definierten Anwendungsoperationen der einzelnen Komponenten durchgegangen werden und hierfür jeweils entsprechender Code generiert werden. Der Unterschied zu den zuvor generierten Code-Skeletons ist allerdings, dass in dem zu generierenden Code für Stubs und Proxies Logik zum Versenden sowie Empfangen von Nachrichten enthalten ist. Aufgrund der Kommunikation mittels Nachrichten können Stubs und Proxies somit auch in voneinander unterschiedlichen Programmiersprachen generiert werden. Für das in Abbildung 6.2 dargestellte Szenario, müsste beispielsweise der Proxy für die Komponente *Python 3 App* in Python und der Stub für die Komponente *Java 17 App* in Java, also jeweils entsprechend der zur Implementierung der jeweiligen Komponente genutzten Programmiersprache, generiert werden.

Konzeptionell könnten die in Abschnitt 6.4.3 generierten Skeletons auch direkt entsprechenden Code mit der Funktionalität zum Senden und Empfangen von Nachrichten enthalten, sodass Stubs und Proxies nicht noch zusätzlich separat generiert werden müssen. Da im Falle von (hinsichtlich der Geschäftslogik) bereits fertig implementierten Komponenten allerdings kein Skeleton generiert werden muss, ist die Möglichkeit zur separaten Generierung von Skeletons, Stubs und Proxies jedoch die flexiblere Herangehensweise. Dementsprechend kann auch solchen Komponenten die Funktionalität zur Kommunikation mit anderen Komponenten einfach mittels der Generierung von Stubs und Proxies ermöglicht werden.

Der beschriebene Ansatz eignet sich vor allem für eine nachrichtenbasierte Kommunikation, beziehungsweise eine RPC-artige Kommunikation zwischen verschiedenen Komponenten. Zur Verarbeitung von Datenströmen ist dieser Ansatz allerdings nur bedingt nutzbar und es müssen hierfür daher individuelle Lösungen realisiert werden. Ebenso ist bei direkten Zugriffen auf

Komponenten, beispielsweise auf Datenbanken, die typischerweise bereits Möglichkeiten für einen entfernten Zugriff anbieten, die Wiederverwendung von existierenden und dazu spezialisierten Artefakten eine besser geeignete Lösung [SBLW17]. Die im Konzept vorgesehene Funktionalität zur Übermittlung von Endpunkten im Anschluss an das Deployment (vgl. Abschnitt 6.2) kann jedoch unabhängig davon eingesetzt werden und dementsprechend zumindest in dieser Hinsicht das Implementieren von individuellem Code zur Konfiguration einer Komponente unnötig machen.

6.4.5 Implementierung der Komponenten

Im vierten Schritt der Methode müssen die zuvor generierten Code-Skeletons mit der Geschäftslogik der jeweiligen Komponente gefüllt werden. Da hierzu Geschäftswissen benötigt wird, muss dieser Schritt manuell vom Entwickler der Anwendung durchgeführt werden. Durch die generierten Code-Skeletons ist der Code jedoch bereits vorstrukturiert und es muss nur noch die reine Geschäftslogik implementiert werden. Da darüber hinaus auch Stubs und Proxies in Schritt drei generiert werden können, die die Kommunikation zwischen verschiedenen Komponenten abstrahieren, kann sich der Entwickler vollständig auf die Implementierung der Geschäftslogik konzentrieren ohne sich zusätzlich noch um die Realisierung von Kommunikationsmöglichkeiten kümmern zu müssen. Dieser Schritt muss entsprechend für jede Komponente, für die ein Code-Skeleton generiert wurde und dementsprechend implementiert werden muss, wiederholt werden.

6.4.6 Modellierung von zum Betrieb der Anwendung benötigten Infrastruktur- und Plattformkomponenten

Im fünften Schritt der Methode müssen die weiteren zum Betrieb der Anwendung nötigen Komponenten sowie deren Beziehungen zueinander modelliert werden, insbesondere Infrastruktur- und Plattformkomponenten. Hierzu können bereits vorhandene Modellelemente sowie Artefakte wiederverwendet werden. Darüber hinaus gibt es verschiedene Arbeiten [EBLW17; WBKL16;

WBL14b] die diesen Arbeitsschritt erleichtern, beispielsweise indem Modellelemente und Artefakte aus anderen Technologien generiert werden können, wie zum Beispiel DevOps-Artefakte aus Chef Cookbooks[1] oder Juju Charms[2]. Dadurch können in anderen Technologien bewährte Lösungen wiederverwendet werden, was die Modellierung und Umsetzung der benötigten Plattform- und Infrastrukturkomponenten zusätzlich vereinfacht. Weiterhin gibt es Arbeiten [HBBL14; KBL17; SBK+18; ZBF+17b], welche unvollständige Topologien automatisiert vervollständigen können, unter anderem, indem geeignete Komponenten mit passenden Fähigkeiten gesucht werden. Beispielweise könnte für eine als WAR bereitzustellende Java-Anwendung, ein entsprechender Webserver, zum Beispiel ein Apache Tomcat, ausgewählt werden. Neben der vollautomatisierten Vervollständigung ist auch eine halbautomatisierte Vervollständigung möglich. Wenn mehrere Komponenten die benötigten Anforderungen einer Komponente befriedigen, kann der Modellierer hier entscheiden, welche davon in der Topologie verwendet werden soll. Anstatt einzelner Komponenten können gegebenenfalls auch Topologiefragmente zur Vervollständigung der Topologie genutzt werden.

6.4.7 Deployment und Binding der Anwendung

Der sechste und letzte Schritt der Methode ist das Deployment der modellierten Anwendung mithilfe eines kompatiblen Deployment Systems. In diesem Schritt werden daher zunächst die modellierten Infrastruktur- und Plattformkomponenten, wie beispielsweise virtuelle Maschinen oder Webserver, installiert und konfiguriert sowie anschließend darauf die Anwendungskomponenten installiert und gegebenenfalls miteinander verbunden. Aufgrund der modellierten Topologie der Anwendung, muss dieser Schritt nur manuell initiiert werden, läuft anschließend aber automatisiert ab. In Abschnitt 6.2 wurde die Grobarchitektur eines solchen Deployment Systems vorgestellt und erläutert, wie mithilfe der Topologie sowie den gespeicherten Instanzdaten einer Anwendung, das Binding der Komponenten automatisiert durchge-

[1] https://www.chef.io/
[2] https://jaas.ai/

führt werden kann. Dementsprechend müssen bei Nutzung der vorgestellten Methode sowie des Programmiermodells keine Komponenten manuell konfiguriert oder verbunden werden, um miteinander kommunizieren zu können. In Abschnitt 8.2 wird weiterhin eine prototypische Implementierung eines hierfür geeigneten Deployment Systems vorgestellt.

6.5 Arbeitsweise des Code-Generators und dessen Plugins

Da alle erforderlichen Informationen über die Anwendungsoperationen, wie zum Beispiel die vorgesehenen Parameter, in der Topologie enthalten sind, ermöglicht der vorgestellte Ansatz, neben der Generierung eines Skeletons, die Generierung von Client-Proxies und Server-Stubs, um die Implementierung der Kommunikationsfunktionalitäten zwischen verschiedenen Komponenten zu erleichtern. Die Grobarchitektur eines solchen Code-Generators ist in Abbildung 6.5 abgebildet. Als Eingabe erhält der Generator die Topologien mit modellierten Komponenten sowie deren definierten Schnittstellenbe-

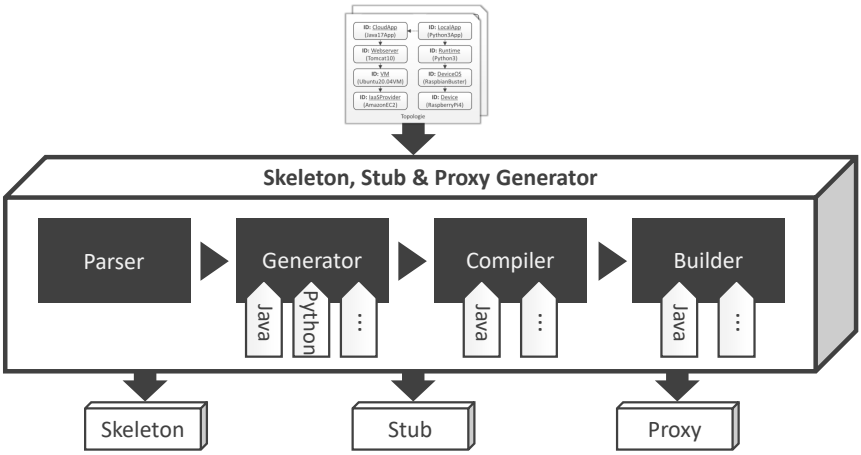


Abbildung 6.5: Grobarchitektur des Generators zur Generierung von Skeletons, Stubs und Proxies

schreibungen. Anschließend analysiert die Komponente *Parser* die Topologie und sucht nach entsprechenden Komponenten mit definierten Anwendungsschnittstellen und Anwendungsoperationen. Abhängig von der Auswahl des Nutzers, generiert die Komponente *Generator* danach Proxies, Stubs und Code-Skeletons für diese Komponenten. Falls Java als Programmiersprache gewählt wurde, wird hier beispielsweise für jede definierte Anwendungsschnittstelle eine entsprechende *.java-Klasse mit den enthaltenen Operationen generiert (vgl. Abschnitt 6.4). Nach der Generierung des Codes muss der generierte Code je nach Programmiersprache, beispielsweise im Falle von Java, kompiliert werden. Dies geschieht in der Komponente *Compiler*. Im letzten Schritt wird das zu einer Komponente gehörige Artefakt, erneut abhängig von der Programmiersprache, von der Komponente *Builder* gebaut. Im Falle von Java wird beispielsweise eine JAR-Datei auf der Grundlage der zuvor in der Komponente *Compiler* kompilierten *.class-Dateien erstellt. Eine solche mithilfe des Generators generierte JAR enthält dann neben dem generierten Code ebenso die zum Betrieb notwendigen Bibliotheken.

Um unterschiedliche Programmiersprachen unterstützen zu können sowie die Menge an unterstützten Programmiersprachen einfach erweitern zu können, wird die eigentliche Generierung des Codes durch entsprechende Plugins ausgeführt. In Abschnitt 6.4.3 wurde der grundsätzliche Algorithmus zu Generierung von Code-Skeletons sowie die Integrationspunkte für die Plugins der unterschiedlichen Programmiersprachen vorgestellt. Da das weitere Vorgehen ebenso abhängig von der gewählten Programmiersprache ist, sind die beiden nachfolgenden Komponenten *Compiler* und *Builder* des Generators ebenfalls Plugin-basiert. Aufgrund der unterschiedlichen Eigenschaften verschiedener Programmiersprachen wird allerdings nicht für jede Programmiersprache jeweils ein entsprechendes Plugin für alle Komponenten benötigt. Im Gegensatz zu Java muss Python beispielsweise nicht vom Entwickler kompiliert werden[1], weshalb für Python keine Plugins für die *Compiler* sowie *Builder* Komponenten benötigt werden.

[1] Die Kompilierung erfolgt bei Python automatisch zur Laufzeit.

6.6 Zusammenfassung und Diskussion

In diesem Kapitel wurde ein Programmiermodell vorgestellt, das die Kommunikation sowie das Binding von Komponenten automatisch bereitgestellter Anwendungen in einer abstrahierten Weise ermöglicht. Weiterhin wird zudem durch die Möglichkeit der Generierung von Code-Skeletons auf Basis der in einer Topologie beschriebenen Anwendungsschnittstellen, die Implementierung dieser Komponenten zusätzlich vereinfacht. Durch diesen Ansatz soll der Entwickler dabei unterstützt werden, seine Anwendung anstatt eines Monoliths, komponentenbasiert zu entwickeln und dadurch von den Vorteilen der in Kapitel 5 vorgestellten, vom Datenfluss einer Anwendung abhängigen, Verteilung der Komponenten profitieren zu können. Da der in diesem Kapitel vorgestellte Ansatz nicht nur im Rahmen der Gesamtmethode dieser Arbeit (siehe Kapitel 3) genutzt werden kann, sondern ebenso auch allein stehend verwendet werden kann, wurde außerdem eine generische Methode zur systematischen Modellierung und Entwicklung komponentenbasierter Anwendungen vorgestellt, welche das Programmiermodell verwendet.

Der grundsätzliche Ansatz, Stubs und Proxies aus einer Schnittstellendefinition zu generieren, um den Aufruf einer entfernten Methode als lokalen Aufruf zu ermöglichen, ist vergleichbar mit anderen Ansätzen wie beispielsweise Java-RMI [Ora10] und CORBA [OMG12]. Da im vorgestellten Konzept jedoch Webservice-Technologien wie zum Beispiel HTTP und XML verwendet werden, ist dieser Ansatz vor allem mit Hinblick auf Firewalls bei der verteilten Kommunikation zwischen den Komponenten unabhängiger einsetzbar. Weiterhin wird durch die Integration des Konzepts in eine Bereitstellungsumgebung, die Registrierung, das Lookup sowie das Binding der Komponenten ebenfalls automatisch von diesem mitübernommen. Um aus Nutzersicht einen normalen lokalen Methodenaufruf abzubilden, ist die interne Kommunikation zwischen den generierten Proxies und Stubs im Rahmen dieser Arbeit dem asynchronen Request-Reply Pattern [Mic22] folgend mittels Polling realisiert. Technisch könnten die vorgestellten Konzepte jedoch auch beispielsweise mittels Warteschlangen und Callbacks implementiert werden.

Zur unabhängigen Beschreibung von Schnittstellen existieren ebenfalls verschiedene Möglichkeiten, wie unter anderem WSDL[1], WADL[2], IDL[3] und OpenAPI[4], welche ebenfalls die Generierung von entsprechenden Code-Fragmenten ermöglichen. Konzeptionell könnte daher anstatt dem in Abschnitt 6.4 vorgestellten Schema zur Schnittstellenbeschreibung auch eine dieser Beschreibungssprachen integriert beziehungsweise referenziert werden. Da das in dieser Arbeit genutzte D^{3}M-Metamodell (siehe Kapitel 4) jedoch auf TOSCA basiert, wird die Beschreibung von Anwendungsschnittstellen analog und damit einheitlich zu dem in TOSCA definiertem Schema zur Beschreibung von Managementschnittstellen und Managementoperationen (siehe [OAS13a; OAS13b]) realisiert.

Weitere Möglichkeit zur Abstraktion der Kommunikation zwischen Komponenten einer verteilten Anwendung, insbesondere im Bereich von Container-basierten Anwendungen und Service Meshes, ist die Umsetzung des Sidecar-Patterns oder des Ambassador-Patterns [BO16; MZ19]. Ein Sidecar wird an eine Komponente angeschlossen, um dieser zusätzliche Funktionalitäten bieten zu können, wie beispielsweise Kommunikation, Monitoring und Logging. Ein Sidecar ist allerdings ein eigenständiger Dienst, der parallel auf derselben Maschine wie die eigentliche Komponente betrieben wird. Ein Sidecar kann daher auf die verfügbaren Ressourcen einer Komponente zugreifen, um diese beispielsweise zu überwachen. Beim Ambassador-Pattern wird ebenfalls ein zusätzlicher Container verwendet, der jedoch ausschließlich für die Kommunikation mit dem Hauptcontainer sowie weiteren Containern verantwortlich ist. Er übernimmt damit eine vergleichbare Funktionalität wie die vorgestellten Stubs und Proxies. Aufgrund der unmittelbaren Nähe zueinander hat weiterhin die Kommunikation zwischen einer Komponente und ihrem Sidecar oder Ambassador nur einen geringen Einfluss auf die Latenz. Ein weiterer Vorteil ist, dass diese Container unabhängig der für die Implementierung einer Komponente genutzten Programmiersprache entwickelt werden

[1] https://www.w3.org/TR/wsdl20
[2] https://www.w3.org/Submission/wadl
[3] https://www.omg.org/spec/IDL/
[4] https://spec.openapis.org/oas/v3.1.0

können, da diese typischerweise mittels Sprach- und Laufzeit-agnostischen Kommunikationsprotokollen verbunden sind. Dementsprechend muss nicht für jede Programmiersprache ein neues Sidecar oder ein neuer Ambassador entwickelt werden. Allerdings muss der Kommunikationsmechanismus zwischen den Komponenten und ihren Sidecars beziehungsweise Ambassadors entsprechend manuell realisiert werden. Gegebenenfalls wäre eine Kombination des Sidecar- oder Ambassadors-Ansatzes mit dem in diesem Kapitel vorgestellten Ansatz möglich um die Generierung dieser Artefakte bei Container-basierten Anwendungen zu ermöglichen. Dies wurde im Rahmen dieser Arbeit allerdings nicht weitergehend untersucht.

Kapitel 7

Sicherheitsrichtlinien bei der Shipping-Automatisierung

Die Analyse und Auswertung von gesammelten Daten ist für Unternehmen eine wichtige Möglichkeit, um zum Beispiel Erkenntnisse über potenzielle Wertsteigerungen zu erlangen oder neue Geschäftsmodelle zu entwickeln [CZ14; GH15; KAEM13; VOE11]. Im Kontext von Industrie 4.0 ist zum Beispiel die Erfassung von Sensordaten und der Einsatz von Analysesoftware unter anderem ein wichtiges Mittel um Erkenntnisse über die Produktion zu erhalten, beispielsweise um vorausschauende Wartung (engl.: Predictive Maintenance) zu ermöglichen [KWXD17; OJ15; ZBF+17a; ZBF+17b].

Allerdings gibt es abhängig von verschiedenen Faktoren, wie zum Beispiel der Art des Unternehmens oder dem Standort eines Unternehmens, eine Vielzahl an Verordnungen und Gesetzen, die verschiedenste Anforderungen und Richtlinien aufstellen, welche von Unternehmen beim Betrieb von Anwendun-

gen sowie der Verarbeitung von Daten eingehalten werden müssen [Mow09; SC13; ZBKL18]. Verstöße dagegen, beispielsweise gegen die EU-Datenschutz-Grundverordnung[1], können zu hohen Geldstrafen führen [Ame21]. Weiterhin kann es je nach Unternehmen weitere individuelle Richtlinien geben, beispielsweise zur Sicherheit der eigenen IT-Landschaft oder zum Schutz von Geschäfts- und Betriebsgeheimnissen [ZBF+17b; ZBKL18]. Damit soll zum Beispiel der Einsatz von veralteter und nicht sicherer Software unterbunden werden, um potenzielle Sicherheitsbedrohungen für die unternehmensweite IT-Infrastruktur zu verhindern.

Bei komplexen und heterogenen, verteilten Anwendungen ist die manuelle Überprüfung aller Regeln und Vorschriften allerdings zeitaufwendig und fehleranfällig. Weiterhin erfordert die Überprüfung ein anderes Fachwissen als die Erstellung von Topologien oder anderen Deployment-Modellen, die typischerweise zur automatisierten Bereitstellung von Anwendungen genutzt werden (vgl. Abschnitt 2.4). So besteht insbesondere bei einer automatisierten und dadurch simplen und schnellen Art der Bereitstellung die Gefahr, eine Anwendung versehentlich an einem falschen Ort, beispielsweise außerhalb der eigenen unternehmensweiten IT-Infrastruktur, zu deployen.

Daher wird in diesem Kapitel sowohl eine Möglichkeit zur einfachen Definition von wiederverwendbaren Anforderungen und Richtlinien sowie ein automatischer Kontrollmechanismus vor dem Deployment vorgestellt. Dieses Kapitel stellt somit Forschungsbeitrag 5 der vorliegenden Arbeit dar und beschreibt weiterhin das Konzept zur Realisierung von Schritt 5 der in Abschnitt 3.2 vorgestellten Shipping-Methode.

In Abschnitt 7.1 wird zunächst eine Übersicht über die beteiligten Rollen sowie die grundlegende Verwendungsweise solcher, im Folgenden *Deployment-Regeln* genannten, Sicherheitsrichtlinien gegeben. Anschließend wird in Abschnitt 7.2 der grundlegende Aufbau von Deployment-Regeln vorgestellt, konkrete Beispiele erläutert und deren Einsatzmöglichkeiten diskutiert. In Abschnitt 7.3 folgt eine Zusammenfassung dieses Kapitels sowie eine Diskussion der vorgestellten Konzepte im Gesamtkontext dieser Arbeit.

[1] `https://eur-lex.europa.eu/eli/reg/2016/679/oj`

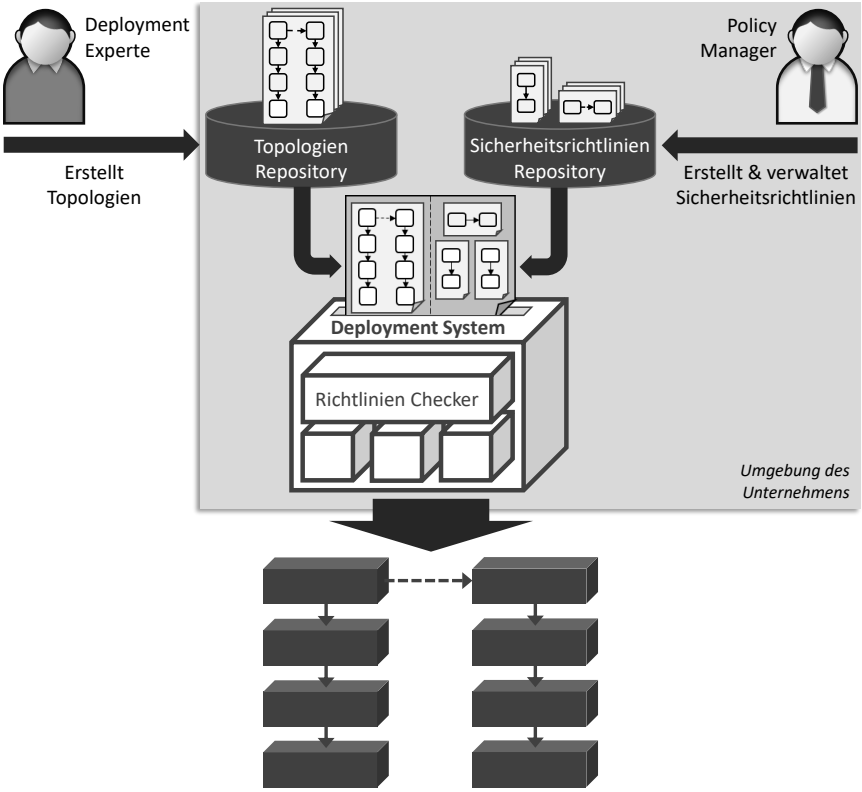


Abbildung 7.1: Konzeptionelle Übersicht der Nutzung von Deployment-Regeln zur Einhaltung von Sicherheitsrichtlinien

7.1 Idee und Grundkonzept

Das Hauptziel des Ansatzes ist es, die Erstellung generischer und wiederverwendbarer Regeln zu ermöglichen, um die Erfüllung bestimmter Anforderungen und Einschränkungen hinsichtlich des Deployments einer Anwendung automatisiert überprüfen und sicherzustellen zu können. Zum Beispiel Anforderungen, die den physischen Ort einschränken, an dem eine Anwendung bereitgestellt werden darf, oder Anforderungen, die festlegen, dass nur

bestimmte Betriebssysteme, beziehungsweise bestimmte Versionen eines Betriebssystems verwendet werden dürfen oder verboten sind. Darüber hinaus sollen die Regeln getrennt von den Topologien spezifiziert werden können, damit sie flexibel an bestehende Topologien angehängt werden können, ohne dass diese bearbeitet werden müssen. Dadurch ist kein besonderes Fachwissen über die Topologien, die jeweils enthaltenen Komponenten oder die verwendeten Bereitstellungstechnologien erforderlich, um die Topologien mit den Sicherheitsrichtlinien des Unternehmens in Einklang zu bringen. Für die Definition der Deployment-Regeln müssen lediglich die Anforderungen und Einschränkungen bekannt sein, die bei der Bereitstellung von Anwendungen grundsätzlich berücksichtigt werden sollen.

Eine Übersicht des Konzepts sowie beteiligte Rollen, Artefakte und Komponenten sind in Abbildung 7.1 dargestellt. Auf der linken Seite der Abbildung ist ein, möglicherweise externer (vgl. Abschnitt 4.2), Deployment Experte dargestellt. Diese Person ist für die Erstellung der Topologie und gegebenenfalls für die Implementierung der Anwendung sowie der dazu benötigten Artefakte verantwortlich. Topologien, unabhängig davon, ob sie intern oder extern erstellt wurden, können in der lokalen IT-Umgebung des Unternehmens in einem *Topologien Repository* gespeichert werden. Auf der rechten Seite der Abbildung ist ein Policy Manager des Unternehmens abgebildet, der für die Erstellung und Pflege der Deployment-Regeln gemäß den Richtlinien und Anforderungen des Unternehmens verantwortlich ist. Um entsprechende Regeln vorgeben zu können, sollte dieser entweder selbst Deployment-Expertise mitbringen oder mit einem Deployment-Experten zusammenarbeiten. Die erstellten Deployment-Regeln können ebenso in einem lokalen *Sicherheitsrichtlinien Repository* verwaltet werden.

Um die Durchsetzung der Sicherheitsrichtlinien eines Unternehmens beim Deployment einer Anwendung zu gewährleisten, müssen Topologien und Deployment-Regeln zusammengebracht werden. Dabei können entweder (i) alle Topologien automatisiert mit allen Deployment-Regeln zusammengeführt werden oder (ii) Topologien individuell mit den jeweils relevanten Deployment-Regeln zusammengeführt werden. Da Topologien flexibel

erstellt werden können (vgl. Abschnitt 4.2) und die Eigenschaften einer Komponente, wie beispielsweise der Endpunkt einer Datenbank oder die Region einer zu erstellenden virtuellen Maschine, erst vor dem Deployment einer Anwendung spezifiziert oder noch abgeändert werden können, kann die Überprüfung der Deployment-Regeln entsprechend erst unmittelbar vor dem Deployment einer Anwendung durchgeführt werden. Dementsprechend muss das verwendete Deployment System eine Komponente beinhalten, die vor der Bereitstellung der Anwendung, aber nach der letzten Möglichkeit für Anpassungen und Eingaben durch den Nutzer, die Einhaltung der Sicherheitsrichtlinien überprüft. Daher enthält das in der Abbildung dargestellte Deployment System eine Komponente *Richtlinien Checker*, welche diese Funktion übernimmt. Falls die Deployment-Regeln erfüllt sind, kann die Anwendung wie in ihrer Topologie definiert, automatisiert bereitgestellt werden. Andernfalls, falls gegen spezifizierte Deployment-Regeln verstoßen wird, wird eine entsprechende Fehlermeldung mit Verweis auf diese ausgegeben und das Deployment abgebrochen.

7.2 Deployment-Regeln zur Einhaltung von Sicherheitsrichtlinien

Zur Modellierung und Einhaltung von Sicherheitsrichtlinien werden in diesem Abschnitt Deployment-Regeln vorgestellt. Wie im vorherigen Abschnitt beschrieben, können diese einer Topologie zugeordnet werden, damit beim Deployment der Anwendung die Erfüllung der Regeln und damit die Einhaltung der vorgesehenen Sicherheitsrichtlinien überwacht wird. Dadurch, dass die Deployment-Regeln einer bestehenden Topologie zugeordnet werden können, können sie unabhängig von konkreten Topologien modelliert werden und es müssen weiterhin keine Topologien zur Einhaltung von Sicherheitsrichtlinien manuell angepasst werden. Dies ermöglicht zudem eine Trennung der Verantwortlichkeiten, mit der Modellierung von Topologien auf der einen Seite und der Modellierung von Deployment-Regeln auf der anderen Seite. Ein weiteres Hauptziel bei der Konzeption der Deployment-Regeln ist, die

Erstellung von möglichst generischen und modularen Deployment-Regeln zu ermöglichen, um somit die Wiederverwendbarkeit der Regeln zu erhöhen. Dementsprechend sind Deployment-Regeln wie folgt definiert:

Definition 7.1 (Deployment-Regel)
Eine Deployment-Regel ist ein gerichteter Graph und beschreibt Richtlinien bezüglich des Deployments einer Anwendung. Sei DR die Menge aller Deployment-Regeln, dann ist ein $dr \in DR$ folgendermaßen definiert:

$$dr = (SK_{dr}, ZK_{dr}, R_{dr}, KT_{dr}, RT_{dr}, E_{dr}, Art_{dr}, \\ typ_{dr}, supertyp_{dr}, eigenschaften_{dr},)$$

Hierbei bezeichnet

- SK_{dr} die Startkomponente in dr und
- ZK_{dr} die Zielkomponente in dr, wobei SK_{dr} sowie ZK_{dr} analog zu einem D^{3}M-Komponentenelement definiert sind (vgl. Definition 4.2).
- R_{dr} die Relation zwischen SK_{dr} und ZK_{dr} in dr und ist analog zu einem D^{3}M-Relationselement definiert (vgl. Definition 4.3).
- KT_{dr} die Menge aller Komponententypelemente in dr, welche analog zu den D^{3}M-Komponententypelementen definiert sind (vgl. Definition 4.4).
- RT_{dr} die Menge aller Relationstypelemente in dr, welche analog zu den D^{3}M-Relationstypelementen definiert sind (vgl. Definition 4.5).
- E_{dr} die Menge aller Eigenschaftselemente in dr, welche analog zu den D^{3}M-Eigenschaftselementen definiert sind (vgl. Definition 4.12).
- $Art_{dr} \in \{whitelisted, blacklisted\}$ die Art der Deployment-Regel (siehe Definition 7.2 und Definition 7.3).

Weiterhin bezeichnet

- typ_{dr} die Abbildung, die jedem Komponentenelement und Relationselement in dr, jeweils dessen Komponententypelement bzw. Relationstypelement zuordnet (vgl. Definition 4.25),

- $supertyp_{dr}$ die Abbildung, die jedem Komponententypelement und Relationstypelement in *dr*, falls vorhanden, jeweils dessen Supertypelement zuordnet (vgl. Definition 4.26),
- $eigenschaften_{dr}$ die Abbildung, die jedem Komponentenelement und Komponententypelement in *dr*, jeweils dessen Eigenschaftselemente zuordnet (vgl. Definition 4.29). ■

Die Elemente einer Deployment-Regel, wie beispielsweise die Start- und Zielkomponenten, sind grundsätzlich analog zu den in Abschnitt 4.1 vorgestellten Elementen eines D^{3}Ms definiert. Dadurch können sie auf eine einheitliche Weise und mit demselben Werkzeug wie ein D^{3}M modelliert werden. So haben die Komponenten eine ID, einen Typ und können Eigenschaften definieren. Eine Deployment-Regel schreibt einen bestimmten Aufbau einer Topologie und gegebenenfalls bestimmte Eigenschaften einer Komponente vor, beziehungsweise verbietet diese. Daher lassen sich Deployment-Regeln in Whitelisting Regeln sowie Blacklisting Regeln unterteilen.

Definition 7.2 (Whitelisting Regel - informell)
Whitelisting Regeln sind Deployment-Regeln, die eine bestimmte Struktur sowie bestimmte Eigenschaften einer Topologie erzwingen. ■

Definition 7.3 (Blacklisting Regel - informell)
Blacklisting Regeln sind Deployment-Regeln, die eine bestimmte Struktur sowie bestimmte Eigenschaften einer Topologie verbieten. ■

Um Deployment-Regeln individuell einem D^{3}M zuordnen zu können sowie zusätzlich Alternativgruppen (siehe Definition 7.4) für Whitelisting Regeln definieren zu können, kann das in Listing 7.1 skizzierte Format verwendet werden. Damit kann beispielsweise festgelegt werden, dass eine Komponente sowohl in der Region *US-West* als auch in der Region *EU-Central* bereitgestellt werden darf (Zeile 5-9). Ohne diese Möglichkeit der Alternativgruppen würden sich diese beiden Regeln ansonsten gegenseitig ausschließen und eine erfolgreiche Bereitstellung der Anwendung somit verhindern.

```
<DeploymentRulesMapping>
  <Rule>Verbiete_Ubuntu14</Rule>
  <Rule>Erlaube_Ubuntu22</Rule>
  ...
  <RuleGroup groupID="Region">
    <Rule>Erlaube_US-West</Rule>
    <Rule>Erlaube_EU-Central</Rule>
    ...
  </RuleGroup>
  ...
</DeploymentRulesMapping>
```

Listing 7.1: Zuweisung von Deployment-Regeln und Definition von Alternativgruppen für Whitelisting Regeln

Definition 7.4 (Whitelisting Alternativgruppe - informell)
Whitelisting Regeln können in einer Alternativgruppe gruppiert werden, um mehrere Deployment-Regeln als Alternativen zu kennzeichnen. Eine Alternativgruppe gilt dann als erfüllt, wenn mindestens eine Deployment-Regel dieser Gruppe erfüllt ist. ■

Bevor im Anschluss verschiedene Beispiele zur Modellierung von Deployment-Regeln dargestellt und diskutiert werden, werden zunächst die Algorithmen zur Überprüfung der Deployment-Regeln vorgestellt. Algorithmus 7.1 ist der High-Level-Algorithmus zur Koordination der Prüfung und bekommt als Eingabe die Topologie *top* sowie die der Topologie zugewiesenen Deployment-Regeln und Alternativgruppen *regelMapping* übergeben. Basierend darauf werden zunächst alle einzeln zugewiesenen Deployment-Regeln mittels Algorithmus 7.2 überprüft (Zeile 4-9). Falls eine Deployment-Regel nicht erfüllt wird, wird diese zu *fehlRegeln* hinzugefügt (Zeile 7). Anschließend werden die Alternativgruppen überprüft (Zeile 10-21). Hierzu wird für jede Deployment-Regel innerhalb einer Alternativgruppe überprüft, ob diese erfüllt wird oder nicht (Zeile 12-17). Wird eine Deployment-Regel nicht erfüllt, wird diese zu *fehlRegelnTemp* hinzugefügt (Zeile 15). Falls keine Deployment-Regel erfüllt wurde und somit die Alternativgruppe nicht erfüllt

Algorithmus 7.1 überprüfeRegeln(top, regelMapping)

```
1:  // top => Topologie, die geprüft werden soll
2:  // regelMapping => Zuweisung von Regeln und Alternativgruppen
3:  fehlRegeln := {} // Fehlgeschlagene Regeln
4:  for all (einzelRegel ∈ regelMapping) do
5:      regel := gibRegel(einzelRegel)
6:      if (¬überprüfeRegel(top, regel)) then
7:          fehlRegeln := fehlRegeln ∪ {regel}
8:      end if
9:  end for
10: for all (gruppenRegeln ∈ regelMapping) do
11:     fehlRegelnTemp := {}
12:     for all (einzelRegel ∈ gruppenRegeln) do
13:         regel := gibRegel(einzelRegel)
14:         if (¬überprüfeRegel(top, regel)) then
15:             fehlRegelnTemp := fehlRegelnTemp ∪ {regel}
16:         end if
17:     end for
18:     if (|fehlRegelnTemp| = |gruppenRegeln|) then
19:         fehlRegeln := fehlRegeln ∪ fehlRegelnTemp
20:     end if
21: end for
22: return fehlRegeln
```

wurde (Zeile 18), werden alle Deployment-Regeln der Alternativgruppe *fehlRegeln* hinzugefügt (Zeile 19). Nachdem alle Deployment-Regeln sowie Alternativgruppen überprüft wurden, gibt der Algorithmus mit *fehlRegeln* die fehlgeschlagenen Deployment-Regeln zurück (Zeile 22). Falls fehlRegeln nicht leer ist, wird die Bereitstellung abgebrochen und die fehlgeschlagenen Deployment-Regeln in der Benutzeroberfläche angezeigt.

Algorithmus 7.2 bestimmt zum einen, ob eine Regel auf die Topologie angewendet werden kann und zum anderen, welche Komponente der Topologie mit der Startkomponente einer Deployment-Regel übereinstimmt. Hierzu bekommt der Algorithmus die Topologie *top* sowie eine einzelne zu prüfende Deployment-Regel *regel* als Eingabe übergeben. Zunächst wird überprüft,

Algorithmus 7.2 überprüfeRegel(top, regel)

```
1:  // top => Topologie, die geprüft werden soll
2:  // regel => Zu überprüfende Regel
3:  if (∃k ∈ gibKomponenten(top) : name(k) = name(gibSK(regel))) then
4:      return regelErfüllt(top, regel, k)
5:  end if
6:  if (name(gibSK(regel)) = ∗) then
7:      for all (k ∈ gibKomponenten(top)) do
8:          if (typ(gibSK(regel)) ∈ {typ(k)} ∪ supertypen(typ(k))) then
9:              return regelErfüllt(top, regel, k)
10:         end if
11:     end for
12: end if
13: return true
```

ob es in der Topologie eine Komponente gibt, die den gleichen Namen wie die Startkomponente der Deployment-Regel hat (Zeile 3). Falls dies der Fall ist, wird die Topologie, die Deployment-Regel sowie die entsprechende Komponente an Algorithmus 7.3 zur weiteren Prüfung übergeben (Zeile 4). Andernfalls wird geprüft, ob der Name der Startkomponente per Wildcard markiert ist (Zeile 6). In solchen Fällen wird nicht auf die Gleichheit der Namen abgeglichen, sondern ob der Typ der Startkomponente innerhalb der Typhierarchie von Komponenten der Topologie liegt (Zeile 7-8). Durch die Möglichkeit der Verwendung von Wildcards sowie der Typisierung von Komponenten können somit generische Deployment-Regeln erstellt werden. Anschließend werden wieder die Topologie, die Deployment-Regel sowie die entsprechende Komponente an Algorithmus 7.3 zur weiteren Prüfung übergeben (Zeile 9). Falls keine entsprechende Übereinstimmung mit der Startkomponente gefunden werden kann, wird die Deployment-Regel als nicht anwendbar und daher als nicht fehlgeschlagen angesehen (Zeile 13).

Algorithmus 7.3 überprüft, ob eine anwendbare Deployment-Regel erfüllt wird oder nicht. Hierzu bekommt der Algorithmus die Topologie *top*, die Deployment-Regel *regel* sowie die mit der Startkomponente der Deployment-Regel gematchte Komponente *komponente* als Eingabe übergeben. Basierend

Algorithmus 7.3 regelErfüllt(top, regel, komponente)

```
1:  // top => Topologie, die geprüft werden soll
2:  // regel => Zu überprüfende Regel
3:  // komponente => Mit SK der Regel gematchte Komponente aus top
4:  komponenten := transitiveHülle(top, komponente, relation(regel))
5:  if (∃k ∈ komponenten : name(k) = name(gibZK(regel))) then
6:      return prüfeEigenschaften(gibZK(regel), k)
7:  end if
8:  if (name(gibZK(regel)) = ∗) then
9:      for all (k ∈ komponenten) do
10:         if (typ(gibZK(regel)) ∈ {typ(k)} ∪ supertypen(typ(k))) then
11:             return prüfeEigenschaften(gibZK(regel), k)
12:         end if
13:     end for
14: end if
15: return art(regel) = blacklisted
```

auf der in der Deployment-Regel definierten Relation, wird zunächst die transitive Hülle ausgehend der Komponente *komponente* berechnet (Zeile 4). Die Variable *komponenten* enthält somit alle Komponenten der Topologie, die ausgehend von Komponente *komponente* durch eine Relation mit demselben Relationstyp wie die Relation der Deployment-Regel mit ihr verbunden sind. Die in einer Deployment-Regel definierte Relation wird somit beim Abgleich mit einem D^3M als transitive Relation betrachtet, wodurch ermöglicht wird, dass nur die für eine Sicherheitsrichtlinie relevanten Komponenten in einer Deployment-Regel modelliert werden müssen (vgl. Abbildung 7.2). So können Komponenten, die sich in einer Topologie zwischen Start- und Zielkomponente der Deployment-Regel befinden würden, aber für die zu modellierende Sicherheitsrichtlinie irrelevante sind, in der Deployment-Regel weggelassen werden. Dadurch wird die Modellierung einer Deployment-Regel vereinfacht sowie die Wiederverwendbarkeit dieser erhöht.

In Zeile 5 wird anschließend eine Komponente innerhalb der transitiven Hülle gesucht, die mit dem Namen der Zielkomponente der Deployment-Regel übereinstimmt. Falls eine solche Komponente gefunden wurde, wird

mittels Algorithmus 7.4 überprüft, ob alle Eigenschaften der Zielkomponente mit dieser gefundenen Komponente übereinstimmen und die Deployment-Regel somit erfüllt ist oder nicht (Zeile 6). Wenn keine Komponente der transitiven Hülle mit dem Namen der Zielkomponente übereinstimmt, wird wiederum die Verwendung der Wildcard überprüft (Zeile 8). Ist die Zielkomponente mittels der Wildcard markiert, wird erneut abgeglichen, ob der Typ der Zielkomponente innerhalb der Typhierarchie von Komponenten der transitiven Hülle liegt (Zeile 9-10). Ist dies der Fall, werden anschließend wieder die Eigenschaften mittels Algorithmus 7.4 abgeglichen und somit die Erfüllung der Deployment-Regel überprüft (Zeile 11). Falls keine Wildcard verwendet wurde und keine Komponente der transitiven Hülle mit dem Namen der Zielkomponente übereinstimmt, wird eine Blacklisting Regel als erfüllt angesehen (Zeile 15). Eine Whitelisting Regel gilt entsprechend als fehlgeschlagen, da die vorgegebene Struktur nicht eingehalten wird.

Algorithmus 7.4 prüfeEigenschaften(zk, komponente)

1: // *zk* => Zielkomponente einer Deployment-Regel
2: // *komponente* => Zu überprüfende Komponente
3: **if** $(\forall e_i \in eigenschaften(zk), \exists e_j \in eigenschaften(komponente) : (name(e_i) = name(e_j)) \wedge (wert(e_i) = wert(e_j)))$ **then**
4: **return** $art(regel) = whitelisted$
5: **end if**
6: **return** $art(regel) = blacklisted$

Algorithmus 7.4 bekommt die Zielkomponente *zk* einer Deployment-Regel sowie die zu überprüfende Komponente *komponente* als Eingabe übergeben. Anschließend werden die Eigenschaften der Zielkomponente mit den Eigenschaften der Komponente *komponente* abgeglichen (Zeile 3). Stimmen die Eigenschaften überein oder hat die Zielkomponente keine Eigenschaften definiert und es handelt sich um eine Whitelisting Regel, wird die Deployment-Regel als erfüllt angesehen (Zeile 4). Falls es sich um eine Blacklisting Regel handelt, wird die Deployment-Regel entsprechend als fehlgeschlagen angesehen. Stimmen die Eigenschaften dagegen nicht überein und es handelt sich um eine Blacklisting Regel, wird die Deployment-Regel als erfüllt angese-

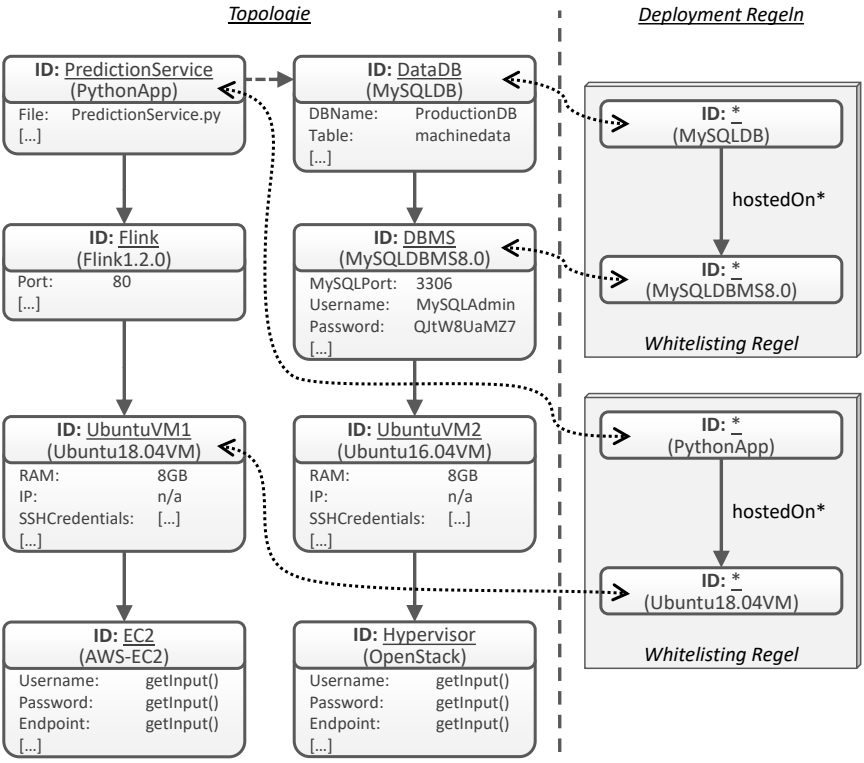
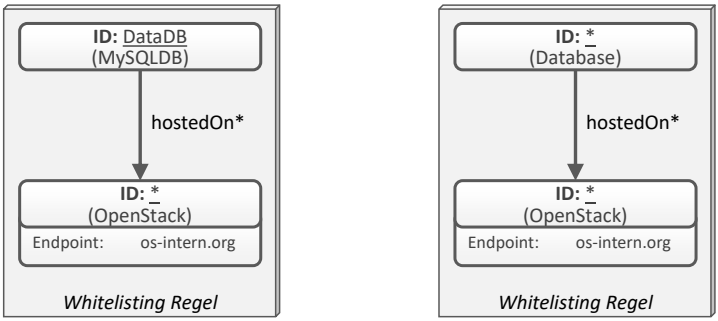


Abbildung 7.2: Beispiel einer Anwendungstopologie mit zugewiesenen Deployment-Regeln

hen (Zeile 6). Eine Whitelisting Regel wird entsprechend als fehlgeschlagen angesehen, da nicht alle vorgegebenen Eigenschaften eingehalten werden.

In Abbildung 7.2 ist auf der linken Seite die Topologie einer Anwendung zu sehen. Die Anwendung besteht aus einer Komponente *PredictionService*, die sich zur Datenbank *DataDB* verbinden soll. Weiterhin sind zum Betrieb benötigte Komponenten, wie beispielsweise virtuelle Maschinen und weitere Infrastrukturkomponenten, sowie Eigenschaften der unterschiedlichen Komponenten abgebildet. Die mittels *getInput()* besetzten Eigenschaften der beiden Komponenten *EC2* und *Hypervisor* sind noch nicht definiert und müs-



(a) (b)

Abbildung 7.3: Vorgabe von Eigenschaften und Verwendung der Typisierung von Komponenten in Deployment-Regeln

sen vor dem Deployment der Anwendung vom Nutzer angegeben werden (vgl. Abschnitt 4.2). Weiterhin zeigt die Abbildung auf der rechten Seite zwei Deployment-Regeln. Die obere Regel gibt vor, dass eine Komponente vom Typ *MySQLDB*, wie die in der Topologie befindliche Komponente *DataDB*, auf einer Komponente vom Typ *MySQLDBMS8.0* betrieben werden darf. Die untere Regel beschreibt analog dazu, dass eine Komponente vom Typ *PythonApp*, wie die in der Topologie befindliche Komponente *PredictionService*, auf einer virtuellen Maschine mit Ubuntu 18.04 betrieben werden darf.

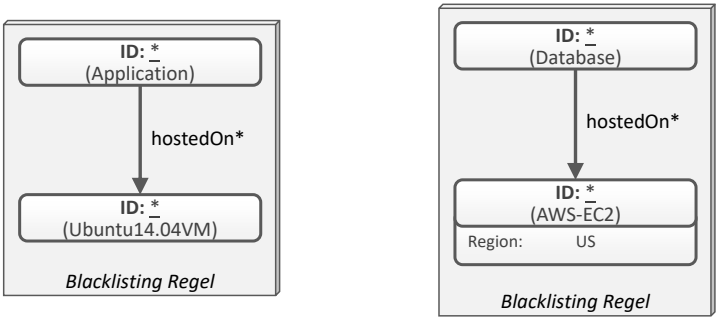
Deployment-Regeln können also unter anderem dafür verwendet werden, um das Deployment nur unter Verwendung von speziellen Komponenten, wie beispielsweise aktuellen und daher voraussichtlich sichereren Softwareversionen, zu erlauben. In beiden Deployment-Regeln sind die Komponenten mittels einer Wildcard markiert. Dadurch werden ausschließlich die Typen der Komponenten berücksichtigt, um zu entscheiden, ob eine Regel erfüllt ist oder nicht. Somit können Regeln einfacher wiederverwendet werden, da sie nicht an konkrete Komponenten einer Topologie gebunden sind.

In Abbildung 7.3 sind weitere Modellierungsmöglichkeiten zur Definition von Deployment-Regeln abgebildet. Abbildung 7.3a zeigt zum einen, dass auch konkrete Komponenten einer Topologie in einer Deployment-Regel

adressiert werden können und zum anderen, wie Eigenschaften innerhalb einer Deployment-Regel genutzt werden können, um weitere unternehmensspezifische Richtlinien umzusetzen. So wird bei dieser Deployment-Regel festgelegt, dass die konkrete Datenbank *DataDB* vom Typ *MySQLDB* nur auf einer *OpenStack* Instanz mit dem Endpunkt *os-intern.org* deployt werden darf. Eine solche Regel kann beispielsweise dafür genutzt werden, um sicherzustellen, dass eine Datenbank mit sensiblen Daten nur innerhalb der unternehmenseigenen IT-Infrastruktur betrieben wird. Weiterhin kann somit ausgeschlossenen werden, dass bei der Initialisierung des Deployments und der dabei gegebenenfalls nötigen Eingabe von fehlenden Eigenschaften, eine falsche Eingabe zur Verletzung von Sicherheitsrichtlinien führt.

In Abbildung 7.3b wird die Typisierung von Komponenten, zur einfacheren Definition von allgemeineren Deployment-Regeln verwendet. Falls beispielsweise die Vorgaben der zuvor vorgestellten Deployment-Regel für alle Datenbanken und nicht nur für eine konkrete Datenbank oder einen Typ von Datenbanken, wie zum Beispiel MySQL Datenbanken, gelten soll, kann deren Supertyp zur Definition verwendet werden. Im konkreten Fall wird daher anstelle des Typs *MySQLDB*, deren Supertyp *Database* genutzt sowie weiterhin die Komponente per Wildcard entsprechend markiert.

Bisher wurden ausschließlich Whitelisting Regeln als Deployment-Regeln vorgestellt. Nachfolgend werden daher auch Beispiele für Blacklisting Regeln vorgestellt und diskutiert. In Abbildung 7.4 sind zwei Deployment-Regeln dieser Art abgebildet. Abbildung 7.4a stellt eine Deployment-Regel dar, welche es verbietet, Anwendungen auf einer virtuellen Maschine zu deployen, die mit Ubuntu 14.04 betrieben wird. Hierzu wird erneut die Möglichkeit der Verwendung der Typisierung von Komponenten genutzt. *Application* kann als eine Art Haupttyp angesehen werden, von dem die anderen Typen für Anwendungskomponenten, wie beispielsweise die in Abbildung 7.2 dargestellte *PythonApp*, erben. Neben dem Erlauben von bestimmten Strukturen mittels Whitelisting Regeln lassen sich mithilfe von Blacklisting Regeln also auch bestimmte Strukturen verbieten. Entsprechend kann dadurch beispielsweise die Verwendung von veralteten Softwarekomponenten unterbunden werden.



(a) (b)

Abbildung 7.4: Beispiele für die Definition von Blacklisting Regeln

Abbildung 7.4b zeigt eine weitere Blacklisting Regel, welche sowohl die Typisierung von Komponenten sowie die Verwendung von Eigenschaften nutzt. Die dargestellte Regel verhindert, beispielsweise aus unternehmensinternen Sicherheitsrichtlinien oder gesetzgeberischen Vorgaben, dass eine Datenbank nicht auf einer *AWS-EC2* Instanz in der Region *US* deployt werden darf. Zur Umsetzung einer solchen Sicherheitsrichtlinie, welche typischerweise nicht mit nur einer Regel definiert werden kann, kann es daher je nach Menge der erlaubten beziehungsweise verbotenen Regionen Sinn ergeben, entweder alle verbotenen Regionen mittels Blacklisting Regeln zu unterbinden oder alle erlaubten Regionen mittels Whitelisting Regeln zu ermöglichen.

Um die verschiedenen Modellierungskonzepte zur Definition von Deployment-Regeln aufzuzeigen, wurde ein fortlaufendes Beispiel genutzt sowie die transitive Relation *hostedOn** durchgehend verwendet. Analog können jedoch auch beliebige andere Relationen, beispielsweise *dependsOn* oder *connectsTo* Relationen, entsprechend in einer Deployment-Regel verwendet werden.

7.3 Zusammenfassung und Diskussion

In diesem Kapitel wurde ein Ansatz zur Definition sowie automatisierten Überprüfung von Sicherheitsrichtlinien, in Form von wiederverwendbaren Deployment-Regeln, vorgestellt. Der Ansatz ermöglicht es, die Deployment-Regeln unabhängig von konkreten Topologien zu modellieren sowie einzusetzen, ohne dass eine Topologie hierzu angepasst werden muss. Durch die Verwendung transitiver Relationen müssen nur die relevanten Komponenten innerhalb einer Regel angegeben werden, was zu einer erhöhten Wiederverwendbarkeit der Deployment-Regeln führt. Weiterhin können Modellierungskonzepte wie Wildcards und Typisierung von Komponenten genutzt werden, um möglichst flexibel einsetzbare Regeln zu definieren. Durch die Unterscheidung in Whitelisting und Blacklisting Regeln, wird zudem eine schnellere Modellierung von Sicherheitsrichtlinien ermöglicht, in Fällen in denen mehrere Varianten erlaubt oder verboten werden sollen.

In der Literatur finden sich einige Arbeiten [MA21], die eine ähnliche Zielsetzung verfolgen. Im Folgenden werden daher ein paar dieser Arbeiten vorgestellt und von dem in diesem Kapitel erläuterten Konzept abgegrenzt.

Képes et al. [KBF+17] präsentieren einen Ansatz zur Durchsetzung nichtfunktionaler Sicherheitsanforderungen während der Bereitstellungsphase von Anwendungen. Dazu gehören zum Beispiel Zugriffsbeschränkungen oder Anforderungen an Passwörter. Diese Sicherheitsanforderungen werden in Form von Policies spezifiziert, die innerhalb einer TOSCA Topologie an eine Komponente angehängt werden. Außerdem werden Provisionierungspläne generiert, welche die Provisionierung der Anwendung unter Einhaltung der definierten Policies ermöglichen. Ein ähnlicher Ansatz zur Definition von nicht-funktionalen Sicherheitsanforderungen wird von Blehm et al. [BKK+14] vorgestellt. Auch sie definieren Sicherheitsanforderungen mittels Policies, die innerhalb einer TOSCA Topologie angegeben werden. Zusätzlich nutzen sie richtlinienspezifische Dienste, die sicherstellen, dass diese Sicherheitsanforderungen eingehalten werden. Im Gegensatz zu den von einer Topologie unabhängigen Deployment-Regeln, sind die Policies in

diesen Konzepten eng an die jeweiligen Topologien gekoppelt. Bestehende Topologien oder Topologien die extern erstellt werden, aber nicht für die konkreten Sicherheitsrichtlinien des Unternehmens ausgelegt sind, müssen daher manuell angepasst werden. Ebenfalls müssen bei Änderungen der Sicherheitsrichtlinien alle entsprechenden Topologien angepasst werden.

Waizenegger et al. [WWB+13] stellen einen *IA-Approach* und einen *P-Approach* vor, um für TOSCA-basierte Topologien die Durchsetzung von Sicherheitsrichtlinien zu gewährleisten. Der IA-Approach erweitert Implementation Artifacts (vergleichbar mit Managementartefakten in einem D^3M), indem bereits existierende Managementoperationen noch einmal mit zusätzlichen Schritten zur Durchsetzung der Richtlinien implementiert werden. Der P-Approach erweitert den für die Bereitstellung der Anwendung erforderlichen Provisionierungsplan um zusätzliche Aktivitäten, welche die Überprüfung und Durchsetzung der Richtlinien ermöglichen. Im Gegensatz zu den Deployment-Regeln, sind die Elemente zur Durchsetzung der Richtlinien in diesen Ansätzen jedoch ebenfalls eng an die Topologien und ihre Artefakte gekoppelt, sodass gegebenenfalls Implementation Artifacts oder Pläne manuell angepasst werden müssen. Weiterhin wird hierzu, neben dem Fachwissen über die unternehmensinternen Sicherheitsrichtlinien, zusätzlich technisches Fachwissen über die jeweiligen Implementierungen benötigt.

Fischer et al. [FBKL17] und Krieger et al. [KBKL18] haben Ansätze zur Sicherstellung der Compliance von TOSCA-basierten Topologien während der Modellierung vorgestellt. Analog zu den Deployment-Regeln, ist auch eines ihrer Hauptziele, die Modellierung von Topologien sowie die Definition von Sicherheitsrichtlinien beziehungsweise Compliance Anforderungen zu trennen. Hierzu führen sie *Compliance Rules* ein, welche sicherstellen sollen, dass die Deployment Struktur einer Topologie mit den Compliance-Anforderungen eines Unternehmens konform sind. Allerdings erfolgt die Prüfung der Topologien in ihren Ansätzen während der Entwurfszeit. Dadurch können beispielsweise Eigenschaften, die erst vor dem Deployment der Anwendung spezifiziert werden, nicht überprüft werden. Weiterhin gibt es in ihren Ansätzen nur die Möglichkeit Strukturen explizit zu erlauben,

aber keine Möglichkeit bestimmte Strukturen explizit zu verbieten, so wie es mithilfe der vorgestellten Blacklisting Regeln möglich ist.

Da im Rahmen dieser Arbeit Komponenten, unter anderem auf Basis des Datenflusses einer Anwendung, verteilt werden und zudem die Topologien automatisiert vervollständigt werden, ist insbesondere die Möglichkeit, unternehmensspezifische Sicherheitsrichtlinien unabhängig von Topologien definieren zu können, ein wichtiger Faktor zur Realisierung der in Kapitel 3 vorgestellten Methode. Dadurch ist es nicht möglich, die Einhaltung von Sicherheitsrichtlinien bereits während der Modellierung einer Topologie zu überprüfen. Daher muss die Überprüfung sowie die Einhaltung der Sicherheitsrichtlinien unmittelbar vor dem Deployment einer Anwendung und dementsprechend innerhalb des Deployment Systems durchgeführt werden.

Eine weitere interessante Arbeit, die sich mit der Absicherung von Anwendungstopologien beschäftigt, stammt von Yussupov et al. [YFK+18]. Allerdings steht in dieser Arbeit vor allem die Sicherheit von Anwendungstopologien sowie Deployment-Paketen bei der kollaborativen Entwicklung durch mehrere Parteien im Mittelpunkt. Ihr Konzept erlaubt daher, sensible Teile, sowohl einer Anwendungstopologie, beispielsweise enthaltene Informationen wie Benutzernamen oder Passwörter, als auch eines Deployment-Pakets, beispielsweise enthaltene Artefakte, mittels definierten Sicherheitsrichtlinien zu schützen. So können beispielsweise bestimmte Topologie-Elemente oder Artefakte verschlüsselt oder signiert werden, um eine mögliche Manipulation durch Unberechtigte ausschließen oder zumindest erkennen zu können. Falls die in Kapitel 3 vorgestellte Shipping-Methode in einer kollaborativen Weise durchgeführt wird, kann dieses Konzept somit zusätzlich zur Absicherung der zu entwickelnden Anwendungstopologie verwendet werden.

Kapitel 8

Architektur, Prototypen und Validierung

Die vorherigen Kapitel haben unter anderem die verschiedenen konzeptionellen Forschungsbeiträge zur Realisierung der einzelnen Schritte der in Kapitel 3 eingeführten Shipping-Methode beschrieben. In diesem Kapitel wird daher eine Gesamtarchitektur vorgestellt, welche die einzelnen Komponenten zur Umsetzung der Forschungsbeiträge integriert. Weiterhin werden die im Rahmen dieser Arbeit entwickelten Prototypen vorgestellt um die Machbarkeit der jeweils diskutierten Konzepte und Ansätze zu validieren. Damit stellt dieses Kapitel Forschungsbeitrag 6 der vorliegenden Arbeit dar.

Das Kapitel ist wie folgt gegliedert: Abschnitt 8.1 veranschaulicht die integrierte Gesamtarchitektur zur Realisierung der Shipping-Methode. Abschnitt 8.2 stellt die prototypischen Implementierungen vor und validiert damit die Umsetzbarkeit der diskutierten Ansätze und Konzepte. Abschnitt 8.3 zeigt die Wiederverwendbarkeit der Prototypen und Konzepte in anderen Arbeiten und Projekten. Abschnitt 8.4 fasst dieses Kapitel zusammen.

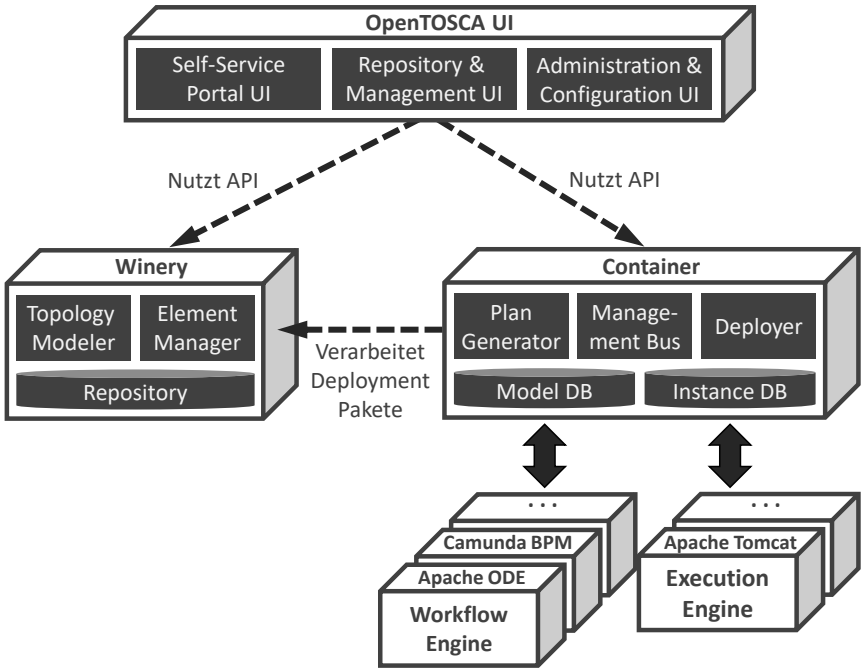


Abbildung 8.1: Gesamtarchitektur des OpenTOSCA Ökosystems, welches als Grundlage zur Implementierung der in dieser Arbeit vorgestellten Ansätze und Konzepte genutzt wird

8.1 Gesamtarchitektur

In diesem Abschnitt wird die integrierte Gesamtarchitektur des Frameworks zur Realisierung der Shipping-Methode vorgestellt. Hierzu werden zunächst die Hauptkomponenten dieses Frameworks schematisch dargestellt sowie einzeln grob beschrieben, bevor im Anschluss daran, die jeweils zur Umsetzung der in dieser Arbeit beschriebenen Ansätze und Konzepte benötigten Erweiterungen im Detail vorgestellt werden.

Das Framework basiert auf dem Open Source Ökosystem OpenTOSCA[1][2], einer Modellierungs-, Deployment- und Laufzeitumgebung für Cloud- und IoT-basierte Anwendungen. Die drei Hauptkomponenten, *OpenTOSCA UI*, *Winery* sowie *Container* und deren Zusammenspiel sind in Abbildung 8.1 dargestellt und werden nachfolgend kurz eingeführt.

Winery ist ein Modellierungstool zur grafischen Erstellung von TOSCA-basierten Anwendungstopologien (vgl. Abschnitt 2.4). Neben dem Modellieren von Topologien mithilfe der *Topology Modeler* Komponente, ermöglicht Winery ebenso das Erstellen, Verwalten und Verändern der einzelnen Elemente einer solchen Topologie (*Element Manager*), wie beispielsweise Knoten, Relationen und benötigte Artefakte. Weiterhin können die modellierten Anwendungen sowie alle zum Deployment und Management benötigten Artefakte als entsprechende Deployment-Pakete exportiert werden.

Der OpenTOSCA Container ermöglicht die Verarbeitung und Ausführung dieser Deployment-Pakete und ermöglicht damit weiterhin das Deployment sowie das Management der darin modellierten Anwendungen. Hierzu generiert der *Plan Generator* auf Basis der Anwendungstopologien jeweils einen ausführbaren Prozess, Plan genannt, der sowohl die einzelnen hierfür auszuführenden Schritte, als auch deren Reihenfolge beschreibt. Typischerweise werden hierzu Workflow-Sprachen wie beispielsweise BPEL [OAS07] oder BPMN [OMG11] genutzt. Dementsprechend ist der Container in der Lage mit entsprechenden *Workflow Engines* zu kommunizieren. Zur Ausführung der in einem Deployment-Paket enthaltenen Managementartefakte, sind ebenfalls *Execution Engines* an den Container angebunden. Pläne sowie Artefakte werden mittels der *Deployer* Komponente in ihrer jeweiligen Umgebung bereitgestellt und mittels dem *Management Bus* entsprechend aufgerufen. Bei Managementoperationen kann es sich beispielsweise um das Skalieren bestimmter Komponenten einer Anwendung oder das Terminieren einer Anwendung handeln. Um dies zu ermöglichen, sammelt und verwaltet der Container die Instanzdaten aller gemanagten Anwendungsinstanzen.

[1] https://www.opentosca.org
[2] https://github.com/OpenTOSCA

Zur benutzerfreundlichen Bedienung, insbesondere des OpenTOSCA Containers, stellt die OpenTOSCA UI verschiedene grafische Schnittstellen für den Benutzer bereit. Beispielweise ermöglicht die *Self-Service Portal UI* die Instanziierung und das Management einer verfügbaren Anwendung mittels dem OpenTOSCA Container. Mithilfe der *Repository und Management UI* kann das Deployment-Paket einer bestimmten Anwendung aus einem Repository ausgewählt und dem Container zur Installation bereitgestellt werden. Weiterhin können mittels der *Administration und Configuration UI* beispielsweise gewünschte Funktionalitäten deaktiviert beziehungsweise aktiviert werden sowie benötigte Endpunkte, zum Beispiel bezüglich der Winery API und der Container API, angepasst werden.

Da die im Rahmen dieser Arbeit umgesetzten Ansätze und Konzepte und dementsprechend die resultierenden Prototypen vor allem in den beiden Komponenten Winery und Container beheimatet sind, werden diese nachfolgend nochmals detaillierter vorgestellt. Da dabei insbesondere die in dieser Arbeit neu eingeführten Komponenten im Fokus stehen, wird zur Betrachtung darüber hinausgehender Details des OpenTOSCA Ökosystems auf Binz et al. [BBH+13] sowie Breitenbücher et al. [BEK+16] verwiesen.

8.2 Prototypische Implementierungen

Dieser Abschnitt stellt die beiden zuvor eingeführten Hauptkomponenten Winery und Container sowohl aus technischer Sicht als auch anschließend aus konzeptioneller Sicht einzeln genauer vor. Insbesondere werden hierbei die im Rahmen dieser Arbeit neu hinzugefügten prototypischen Implementierungen zur Umsetzung der vorgestellten Konzepte näher betrachtet.

Aus technischer Sicht sind der OpenTOSCA Container sowie das Backend von Winery in Java implementiert. Um eine Nutzung der implementierten Funktionalitäten sowie die Kommunikation zwischen den Komponenten zu ermöglichen, bieten beide Komponenten REST-APIs an. Diese sind weiter-

hin entsprechend der *OpenAPI Specification*[1] dokumentiert. Die grafische Benutzeroberfläche von Winery ist mithilfe des Web-Frameworks Angular[2] implementiert. Dementsprechend basiert sie vor allem auf clientseitigen Webtechnologien wie TypeScript, HTML sowie CSS und kann von einem Anwender dementsprechend mit einem Webbrowser verwendet werden.

8.2.1 OpenTOSCA Winery - Modellierung, Verwaltung und Paketierung

Winery, in seinem Ursprung, ist ein grafisches Modellierungswerkzeug für TOSCA-basierte Anwendungstopologien und ist Teil des OpenTOSCA Ökosystems [BEK+16; KBBL13]. Im Rahmen dieser Arbeit wurde Winery erweitert und angepasst, um die vorgestellten Konzepte sowie die einzelnen Forschungsbeiträge zu realisieren. Beispielsweise wurde Winery angepasst, um die Verwendung der in Abschnitt 4.1 vorgestellten D^{3}M-Elemente bei der Modellierung von Anwendungstopologien zu ermöglichen. Abbildung 8.2 gibt einen Überblick über die Architektur von Winery und stellt insbesondere die neu hinzugefügten und angepassten Komponenten dar, welche im Folgenden genauer betrachtet und vorgestellt werden.

Die UI-Komponente *D^{3}M Topology Model & Deployment Rules Editor* ermöglicht die grafische Modellierung von D^{3}M-basierten Anwendungstopologien. Aus einer Palette können hierzu Komponententypen in die Modellierungsfläche gezogen und dort mittels Relationstypen verbunden werden. Weiterhin können beispielsweise die konkreten Eigenschaften eines Komponententyps gesetzt oder verändert werden. Da, wie in Kapitel 7 diskutiert, die einheitliche Modellierung von Anwendungstopologien sowie Deployment-Regeln ermöglicht werden soll, unterstützt diese Komponente ebenfalls die Modellierung von Deployment-Regeln auf identische Weise und dient somit der Realisierung von Forschungsbeitrag 2 sowie Forschungsbeitrag 5.

Der *D^{3}M Element Manager* ermöglicht das Anlegen, Verändern und Löschen von D^{3}M Elementen, wie beispielsweise Komponententypen, Fähigkeitsty-

[1] https://www.openapis.org/
[2] https://angular.io/

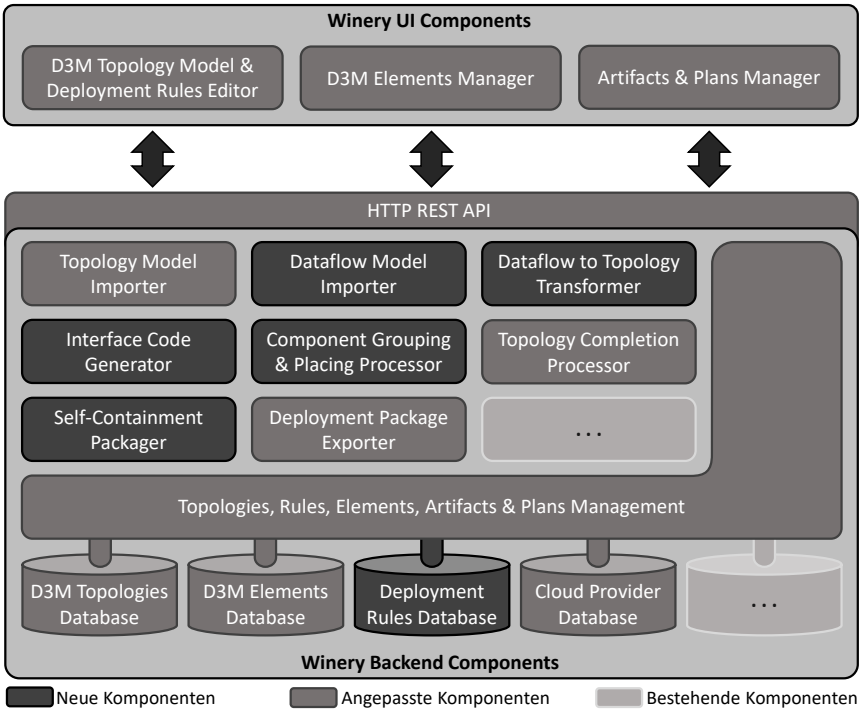


Abbildung 8.2: Erweiterte OpenTOSCA Winery Architektur

pen, Anforderungstypen oder Relationstypen und ist somit ebenfalls für die Realisierung von Forschungsbeitrag 2 nötig. Der *Artifacts & Plans Manager* ermöglicht die Verwaltung, also zum Beispiel das Hochladen oder Löschen, von Artefakten und Plänen. Weiterhin können hiermit beispielsweise Codefragmente zur vereinfachten Implementierung generiert werden, beispielsweise von Managementartefakten oder Anwendungsartefakten basierend auf deren jeweils definierten Schnittstellen und Operationen. Diese Komponente ist somit bei der Realisierung von Forschungsbeitrag 4 beteiligt.

Um die Kommunikation mit den Winery UI-Komponenten zu ermöglichen, bietet das Winery Backend eine *HTTP REST API* an. Damit können sowohl die angebotenen Funktionalitäten der einzelnen Komponenten genutzt werden,

als auch auf die von Winery verwalteten Objekte, wie beispielsweise die Anwendungstopologien, D^{3}M Elemente sowie Artefakte zugegriffen werden, welche in entsprechenden Datenbanken gehalten werden.

Die im Kontext dieser Arbeit wichtigsten Komponenten des Winery Backends, welche insbesondere die Anwendungslogik der diskutierten Ansätze und Konzepte implementieren, werden nachfolgend nacheinander und mit Bezug zu den damit jeweils umgesetzten Forschungsbeiträgen vorgestellt:

Der *Topology Model Importer* ermöglicht den Import von D^{3}M- sowie TOSCA-basierten Anwendungstopologien, um diese anschließend weiter zu verarbeiten, anzupassen oder zu verwalten. Hierzu wurde Winery erweitert, um neben TOSCA auch das in Abschnitt 4.1 vorgestellte D^{3}M-Metamodell zu unterstützen und damit Forschungsbeitrag 2 dieser Arbeit zu realisieren.

Der *Dataflow Model Importer* erlaubt den Import der in Kapitel 5 beschriebenen Datenflussmodelle und ermöglicht zusammen mit dem *Dataflow to Topology Transformer* die Ableitung einer Anwendungstopologie aus dem Datenflussmodell einer Anwendung[1]. Hierzu wird nach dem Hochladen des Datenflussmodells dieses geparst und eine erste unvollständige Anwendungstopologie erstellt (vgl. Abschnitt 5.2). Basierend darauf ermöglicht der *Component Grouping & Placing Processor* die Gruppierung und Platzierung der enthaltenen Komponenten dieser erstellten Anwendungstopologie, unter anderem auf Basis des Datenflusses der Anwendung sowie weiteren Merkmalen, wie zum Beispiel den Datenfaktoren der einzelnen datenverarbeitenden Komponenten und der Menge an insgesamt zu verarbeitenden Daten. Die hierfür genutzten Algorithmen wurden in Abschnitt 5.3 vorgestellt. Zusammen mit dem *Topology Completion Processor*, der die Vervollständigung von unvollständigen Anwendungstopologien ermöglicht, dienen diese Komponenten somit insbesondere der Realisierung von Forschungsbeitrag 3.

Der *Interface Code Generator* ermöglicht das Generieren von Code-Fragmenten zur Implementierung von Managementartefakten oder Anwendungsarte-

[1] Zur Erstellung entsprechender Datenflussmodelle wurde im Rahmen dieser Arbeit zusätzlich ein browserbasiertes Modellierungstool prototypisch implementiert: https://github.com/OpenTOSCA/DataFlowModeler

fakten auf Basis der Schnittstellenbeschreibungen eines Komponententyps. Diese generierten Code-Fragmente sollen unter anderem die Kommunikation sowie den Endpunktaustausch zwischen interagierenden Komponenten abstrahieren (vgl. Abschnitt 6.3), wodurch die Implementierung von Anwendungsartefakten und damit die Aufteilung in einzelne und verteilbare Komponenten vereinfacht wird. In Abschnitt 6.4 wurde hierzu eine systematische Entwicklung von D^{3}M-basierten Anwendungen und deren Komponenten sowie die Nutzung von Codegenerierung bei der Implementierung der Anwendungsartefakte vorgestellt. Diese Komponente dient damit der Umsetzung von Forschungsbeitrag 4. Der Aufbau und die Arbeitsweise eines solchen Code-Generators wurden in Abschnitt 6.5 diskutiert.

Der *Self-Containment Packager* ist für das Auflösen von externen Abhängigkeiten verantwortlich und ermöglicht dadurch die Erstellung von eigenständigen Deployment-Paketen. Solche Deployment-Pakete können in ihrer Zielumgebung ohne externe Dateien und dementsprechend auch ohne Internetzugriff erfolgreich bereitgestellt werden (vgl. Definition 4.37). Das hierzu erforderliche Vorgehen sowie die konkrete Architektur eines solchen Paketierungsframeworks wurden in Abschnitt 4.3 vorgestellt.

In einer durchgeführten Studie wurden weiterhin die beiden Deployment-Paket Varianten hinsichtlich ihrer jeweiligen Größe sowie der zur Bereitstellung benötigten Zeit verglichen [ZBH+20]: Als beispielhafte Anwendung wurde hierfür ein LAMP-Stack[1] genutzt. Um die Auswirkungen verschiedener Technologien zu prüfen, wurde weiterhin ein Deployment-Paket ausschließlich mit Skripten realisiert und ein zweites Deployment-Paket mithilfe Docker realisiert. Die Ausgangsgröße der beiden nichteigenständigen Deployment-Pakete betrug 28 MB für das mittels Skripten realisierte Deployment-Paket und 72 MB für das mittels Docker realisierte Deployment-Paket. Bei der Transformation von einem nichteigenständigen Deployment-Paket in ein eigenständiges Deployment-Paket werden alle für die Bereitstellung benötigten Dateien direkt mit in das jeweilige Deployment-Paket gepackt. Im Falle der Skripte werden dementsprechend die jeweiligen Installationsdateien und

[1] Software-Stack bestehend aus Linux, Apache, MySQL und PHP.

Abhängigkeiten der einzelnen Komponenten aufgelöst, heruntergeladen und paketiert. Im Falle von Docker werden die entsprechenden Docker-Images auf Basis der definierten Dockerfiles lokal erstellt und anschließend ebenso paketiert. Durch die Umwandlung zu einem eigenständigen Deployment-Paket wuchs die Größe des mithilfe von Skripten realisierten Deployment-Pakets somit um 115 MB auf insgesamt 143 MB an. Die Größe des mittels Docker realisierten Deployment-Pakets wuchs um 140 MB auf insgesamt 212 MB an. Die zur Bereitstellung benötigte Zeit betrug 113 Sekunden für die nichteigenständige Variante und 109 Sekunden für die eigenständige Variante des mithilfe von Skripten realisierten Deployment-Pakets. Für das mittels Docker realisierte Deployment-Paket betrug die Zeit zur Bereitstellung 48 Sekunden für die nichteigenständige Variante und 60 Sekunden für die eigenständige Variante. Bei der Bewertung und dem Vergleich der Bereitstellungszeiten ist jedoch zu beachten, dass im Falle von eigenständigen Deployment-Paketen insbesondere der zur Verfügung stehende Upload einen großen Einfluss auf die jeweils benötigte Zeit hat, da hier bereits alle benötigten Dateien im Paket enthalten sind und in die Zielumgebung geladen werden müssen. Für nichteigenständige Deployment-Pakete dagegen hat vor allem die Downloadrate in der Zielumgebung einen großen Einfluss, da die benötigten Dateien zur Bereitstellung dort zunächst heruntergeladen werden müssen.

Solche Deployment-Pakete, unabhängig davon, ob eigenständig oder nicht, können mit dem dargestellten *Deployment Package Exporter* erstellt sowie exportiert werden und anschließend entweder beliebig verteilt oder zur Ausführung an eine kompatible Laufzeitumgebung als Eingabe geschickt werden. Die Erstellung von Deployment-Paketen vervollständigt damit die Realisierung von Forschungsbeitrag 2 dieser Arbeit.

Neben diesen Komponenten wurde weiterhin die *Cloud Provider Database* sowie die dazugehörige Managementschicht angepasst. In dieser Datenbank werden alle von Cloud-Anbietern angebotenen und als Komponententypen implementierten Modellelemente in entsprechenden Namensräumen verwaltet. Dementsprechend wurde die Managementschicht erweitert, um Cloud-Anbieter nach ihrem Standort, zum Beispiel *US-West* oder *US-Ost*,

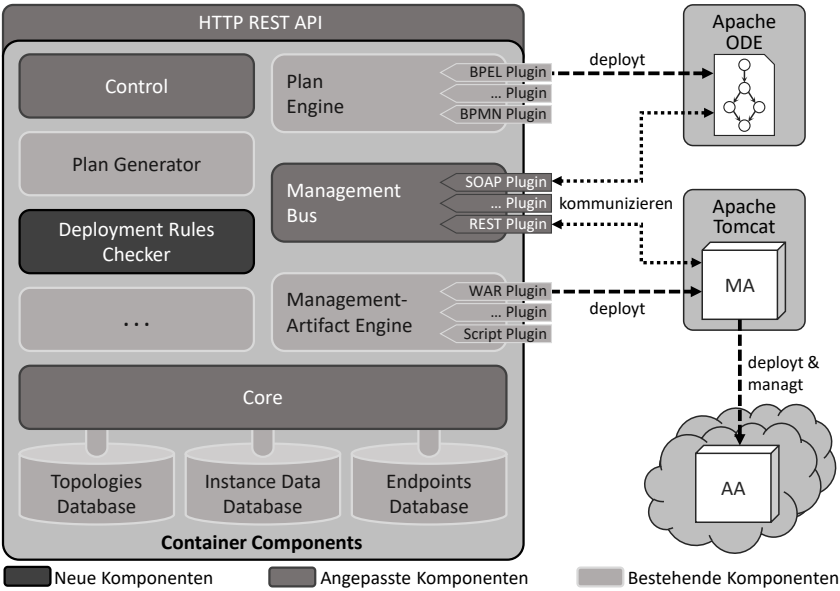


Abbildung 8.3: Erweiterte OpenTOSCA Container Architektur (links) sowie verwendete externe Komponenten (rechts)

filtern sowie auswählen zu können. Diese Anpassungen dienen der Gruppierung und Platzierung von Komponenten sowie der Vervollständigung von unvollständigen Anwendungstopologien. Ebenfalls wurden die weiteren Datenbanken angepasst um D^3M-Elemente verwalten zu können.

8.2.2 OpenTOSCA Container - Deployment und Management

Die im Rahmen dieser Arbeit erweiterte und angepasste Architektur des OpenTOSCA Containers, der Deployment- und Managementlaufzeitumgebung des OpenTOSCA Ökosystems, ist in Abbildung 8.3 abgebildet. Die Abbildung zeigt weiterhin die Erweiterbarkeit einzelner Komponenten durch entsprechende Plugins sowie deren Möglichkeiten der Konnektivität zu externen Komponenten und verwalteten Artefakten.

Um die vorhandenen Funktionalitäten zur Verfügung zu stellen, beinhaltet der OpenTOSCA Container die abgebildete Komponente *HTTP REST API*. Damit können unter anderem Deployment-Pakete in den Container geladen werden, das Deployment von Anwendungen initialisiert werden, Instanzdaten von verwalteten Komponenten abgerufen werden sowie bestimmte weitere Funktionen der enthaltenen Komponenten, wie beispielsweise das Generieren von Plänen mithilfe des *Plan Generators*, aufgerufen werden. Um dem Nutzer gegebenenfalls fehlgeschlagene Deployment-Regeln in der Benutzeroberfläche anzeigen zu können (vgl. Abschnitt 7.1), musste diese Komponente entsprechend angepasst werden.

Um bei der automatisierten Bereitstellung einer Anwendung, die einzelnen Funktionen des Containers in ihrer erforderlichen Reihenfolge aufzurufen, koordiniert die *Control* Komponente den Ablauf innerhalb des Containers. Beispielweise wird überprüft, ob im entsprechenden Deployment-Paket bereits ein zum Deployment benötigter Plan enthalten ist oder nicht. Falls kein Plan enthalten ist, wird ein entsprechender Plan mithilfe des *Plan Generators* erstellt. Hierzu müssen die Anwendungstopologie, die enthaltenen Komponenten sowie deren Relationen zueinander analysiert werden. Ein solcher Plan, auch Provisionierungsplan genannt, beschreibt die einzelnen auszuführenden Arbeitsschritte, die für das Deployment ausgeführt werden müssen, sowie deren Reihenfolge [BBK+14; KBF+17]. Dadurch kann sowohl das imperative sowie das deklarative Deployment durch den OpenTOSCA Container ermöglicht werden (vgl. Abschnitt 2.4). Weiterhin können auch Managementpläne generiert werden, um beispielsweise Komponenten einer modellierten Anwendung zu skalieren [HBB+21]. Auch diese Komponente musste angepasst werden, um bei definierten Deployment-Regeln die Komponente zur Überprüfung der Deployment-Regeln aufzurufen.

Eine weitere Komponente des OpenTOSCA Containers, welche zur Realisierung von Forschungsbeitrag 5 dieser Arbeit implementiert wurde, ist dementsprechend der *Deployment Rules Checker*. Diese Komponente überprüft direkt vor dem Deployment einer Anwendung, ob alle hierzu definierten Deployment-Regeln (vgl. Abschnitt 7.2) dabei eingehalten werden. Falls min-

destens eine Regel nicht eingehalten wird, bricht der Bereitstellungsvorgang der Anwendung mit einem entsprechendem Hinweis ab. Damit kann unter anderem sichergestellt werden, dass Sicherheitsanforderungen eingehalten werden und keine Komponenten oder Daten an einem Ort oder auf einer Plattform ausgeführt oder gespeichert werden, wo dies nicht vorgesehen ist.

Da Pläne durch ausführbare Workflows realisiert werden, müssen diese in entsprechenden Laufzeitumgebungen bereitgestellt werden. Die *Plan Engine* ist daher für das Deployment der, in einem Deployment-Paket enthaltenen oder im Container generierten, Pläne verantwortlich. Um unterschiedliche Workflow-Sprachen sowie Laufzeitumgebungen zu unterstützen, wird mithilfe eines Plugin-Systems eine einfache Erweiterbarkeit ermöglicht. So gibt es beispielsweise jeweils ein Plugin um BPEL-basierte Pläne auf Apache ODE[1] zu deployen oder BPMN-basierte Pläne auf Camunda[2] zu deployen.

Analog zur *Plan Engine* enthält der OpenTOSCA Container eine *Management-Artifact Engine* um die in einem Deployment-Paket enthaltenen Managementartefakte auf entsprechenden Laufzeitumgebungen bereitstellen zu können. Managementartefakte implementieren Operationen zum Management von Komponenten einer Anwendung und werden hierzu von Plänen aufgerufen (vgl. Abschnitt 4.1). Da Managementartefakte auf unterschiedlichste Weise implementiert sein können, beispielsweise als WAR[3] paketierte Komponente, die auf einem Apache Tomcat[4] ausgeführt werden muss, enthält die *Management-Artifact Engine* ebenfalls ein Plugin-System.

Um eine einfache Kommunikation zwischen dem OpenTOSCA Container, Plänen sowie Managementartefakten zu ermöglichen, enthält der Container die Komponente *Management Bus*, die einheitliche Schnittstellen zum Aufruf der durch Managementartefakte angebotenen Managementoperationen sowie Plänen bereitstellt [WBB+14a; WBB+14b]. Dadurch müssen Pläne beispielsweise nicht an konkrete Managementartefakte gebunden werden,

[1] https://ode.apache.org/
[2] https://camunda.com/
[3] Web Application Archive - Java-Servlet-Spezifikation entsprechendes Dateiformat für Webanwendungen
[4] https://tomcat.apache.org/

sondern können die abstrakten Schnittstellen- und Operationsbeschreibungen referenzieren, unabhängig der jeweils zur Implementierung genutzten Technologien. Somit wird die Portabilität der Artefakte sowie Pläne erhöht. Weiterhin ermöglicht der *Management Bus* den initialen Aufruf und damit den Start eines Plans. Um verschiedene Kommunikationsprotokolle zu unterstützen sowie eine einfache Erweiterbarkeit dieser zu ermöglichen, enthält auch der *Management Bus* ein entsprechendes Plugin-System. Zur Realisierung von Forschungsbeitrag 4 wurde diese Komponente ebenfalls angepasst, um bei der Verwendung der generierten Proxies und Stubs, diesen die Endpunktinformationen ihrer jeweiligen Kommunikationspartner mitzuteilen. Hierzu werden insbesondere die gespeicherten Instanzdaten der einzelnen verwalteten Anwendungsinstanzen sowie die jeweiligen Endpunktinformationen benötigt (vgl. Abschnitt 6.2). Daher kann der *Management Bus* mithilfe der *Core* Komponente auf alle verfügbaren Daten, die in den verschiedenen Datenbanken, wie beispielsweise *Topologies Database*, *Instance Data Database* sowie *Endpoints Database* gehalten werden, zugreifen.

8.3 Verwendung der vorgestellten prototypischen Implementierungen und Konzepte in anderen Arbeiten

Dieser Abschnitt stellt exemplarische Arbeiten und Projekte vor, in denen die vorgestellten Prototypen und Konzepte dieser Arbeit, insbesondere aus Forschungsbeitrag 2 sowie Forschungsbeitrag 4, wiederverwendet werden.

8.3.1 Modellierung verteilbarer Anwendungen und Paketierung eigenständiger Deployment-Pakete

Hirsch et al. [HILZ22] stellen eine Arbeit vor, um die Wiederverwendbarkeit und Bereitstellung von Forschungssoftware zu verbessern. Dabei soll zudem ermöglicht werden, dass diese unabhängig der Ausführungsumgebung auch in Zukunft ausführbar und nutzbar sind. Um dies zu erreichen, sollen die Forschungssoftware, alle zur Bereitstellung und Ausführung benötigten Ar-

tefakte sowie weitere, für die jeweilige Forschungsarbeit relevanten Daten, entsprechend zusammen paketiert werden. Zur Umsetzung der Modellierung und Paketierung wird Winery und deren Funktionalität zur Paketierung von eigenständigen Deployment-Paketen verwendet. Dies zeigt, dass die Erstellung solcher eigenständiger Deployment-Pakete, wie in Abschnitt 4.3 vorgestellt, nicht nur für komplexe, datenverarbeitende und zu verteilende Anwendungen genutzt werden kann, sondern auch für typischerweise kleinere und simplere prototypische Forschungssoftware relevant ist.

In einer weiteren Arbeit erläutern Weder et al. [WBLZ21], dass Quantenanwendungen typischerweise aus klassischer Software sowie Quantenalgorithmen bestehen und daher als hybrid zu bezeichnen sind. Dementsprechend müssen unter anderem der Kontroll- sowie Datenfluss zwischen diesen verschiedenen Komponenten orchestriert werden. Um dies zu ermöglichen, beschreiben sie die Architektur einer Modellierungs- und Laufzeitumgebung für solche Quantenanwendungen und stellen weiterhin eine prototypische Implementierung auf Grundlage von OpenTOSCA vor. Darüber hinaus definieren sie sogenannte *Quantum Application Archives (QAA)* um hybride Quantenanwendungen sowohl einheitlich als auch gesamtheitlich paketieren und ausliefern zu können. Diese Archive enthalten unter anderem das Topologiemodell der Anwendung, die klassischen Softwareteile und deren Abhängigkeiten, die implementierten Quantenalgorithmen, verschiedene Workflows zur Orchestrierung der Bereitstellung und Ausführung, sowie gegebenenfalls weitere benötigte Daten. Bei ihrer prototypischen Implementierung wird Winery und deren, im Rahmen dieser Arbeit erarbeiteten, Funktionalität zur Erstellung eigenständiger Pakete genutzt. Dies zeigt, dass die in Forschungsbeitrag 2 vorgestellten Konzepte auch in anderen Domänen, wie dem Quantencomputing, relevant sind und angewendet werden können.

Das vom BMWK geförderte Projekt *SePiA.Pro* [BMWKa; GPR+19] beschäftigte sich mit der Erfassung, Analyse und intelligenten Verwertung von Sensor- und Auftragsdaten moderner Produktionsanlagen. Hierzu wurde eine Plattform gebaut um sogenannte Smart Services entwickeln sowie ausführen zu können. Mittels einem solchen Smart Service sollen beispielsweise die

Zustände von Maschinen überwacht werden und potenzielle Fehler oder Probleme gegebenenfalls schon vor ihrem Auftreten erkannt werden [FBK+16]. Im Rahmen des Projekts wurde hierzu unter anderem ein sicheres und selbstbeschreibendes Format zur Paketierung dieser Smart Services entwickelt. Die so paketierten Smart Services wurden nicht nur in einer simulierten Umgebungen getestet, sondern ebenso in einer Fabrik mit sich in Betrieb befindlichen Maschinen ausgeführt. Aufgrund vorhandener Zugriffsbeschränkungen mussten die Smart Services hierfür als eigenständige Deployment-Pakete paketiert werden, um ein erfolgreiches Deployment zu ermöglichen. Weiterhin wurden die in Abschnitt 4.2 vorgestellten Modellierungskonzepte verwendet, um Smart Services sowohl flexibel als auch sicher, bezüglich den weiterzugebenden Daten über die vorhandene Infrastruktur, modellieren zu können. Das Projekt zeigt weiterhin die Wichtigkeit von Sicherheitsrichtlinien (vgl. Forschungsbeitrag 5) für Unternehmen, um sicherzustellen, dass Anwendungen und deren Komponenten nur an dafür vorgesehenen Orten bereitgestellt und betrieben werden können.

8.3.2 Programmiermodell zur Abstraktion der Kommunikation zwischen Komponenten verteilter Anwendungen

Wild et al. [WBH+20] beschreiben in ihrer Arbeit zwei Modellierungskonzepte für Topologien zur Automatisierung der Bereitstellung und Orchestrierung von Quantenanwendungen. Das erste Modellierungskonzept, *SDK-S Modelle* genannt, beschreibt alle technischen und SDK-spezifischen Details bezüglich der Bereitstellung, wie beispielsweise hierfür benötigte Laufzeitumgebungen und Bibliotheken. Das zweite Modellierungskonzept, *SDK-A Modelle* genannt, verbirgt diese technischen sowie SDK-spezifischen Details, wodurch die Topologien einfacher zu lesen sowie zu erstellen sind. Allerdings ist eine solche Topologie, aufgrund der fehlenden technischen Details, nicht ausführbar. Um dennoch die Arbeit mit den einfacheren SDK-A Modellen zu ermöglichen, stellen Wild et al. in ihrer Arbeit weiterhin einen modellgetriebenen Entwicklungsansatz vor. Dieser ermöglicht es, die jeweils notwendige technische Umsetzung aus SDK-A Modellen abzuleiten, um daraus anschließend

ausführbare SKD-S Modelle zu erstellen. Hierzu müssen Code-Fragmente zu den einzelnen Komponenten hinzugefügt werden, welche dann die Kommunikation zwischen den verschiedenen Komponenten der Anwendung ermöglichen. Dies ist in ihrem beschriebenen und implementierten Prototyp jedoch nicht automatisiert durchführbar. Um dies zu ermöglichen, wird daher auf die Möglichkeit zur Integration der in Forschungsbeitrag 4 dieser Arbeit vorgestellten Konzepte zur Generierung von Stubs auf Grundlage von Anwendungsschnittstellen verwiesen.

Im vom BMWK geförderten Projekt *SmartOrchestra* [BMWKb; LHG+19] wurde eine offene und standardbasierte Plattform entworfen und prototypisch implementiert, um als *Smart Services* paketierte Cyber-physische Anwendungen erstellen und verwalten sowie sicher miteinander zu kombinieren und orchestrieren zu können. Als Teil der Plattform wurde weiterhin ein Marktplatz erschaffen, um diese Smart Services vermarkten zu können. Dadurch soll unter anderem die Entwicklung und Bildung neuer und intelligenter Geschäftsmodelle ermöglicht werden. Insbesondere die Integration verschiedener CPS-Technologien sowie die dadurch ermöglichte Vereinfachung der Nutzung von Sensoren und Aktuatoren innerhalb von Anwendungen standen hierbei im Fokus des Projekts. Um eine einfache Kommunikation zwischen verschiedenen Services und Komponenten zu ermöglichen und somit die Entwicklung und Komposition der Anwendungen zu vereinfachen, wurden im Projekte unter anderem die in Kapitel 6 beschriebenen Ansätze eingesetzt. Dadurch müssen hierfür keine komplexen Endpunktinformationen ausgetauscht werden. Weiterhin muss nicht manuell gegen eine Schnittstelle einer anderen Komponente implementiert werden.

8.4 Zusammenfassung und Diskussion

In diesem Kapitel wurde zunächst eine Gesamtarchitektur vorgestellt, welche die Hauptkomponenten zur Umsetzung der Forschungsbeiträge dieser Arbeit integriert und damit weiterhin die Realisierung der in Kapitel 3 vorgestellten Shipping-Methode ermöglicht. Darauf folgend wurden die prototypischen

Implementierungen, der in den vorhergehenden Kapiteln diskutierten Konzepte und Ansätze, sowohl aus technischer als auch konzeptioneller Sicht vorgestellt. Als Grundlage dieser Implementierungen wurde das OpenTOSCA Ökosystem genutzt. Dessen beiden Hauptkomponenten Winery, zur Modellierung und Verwaltung von Anwendungstopologien sowie zur Paketierung von Deployment-Paketen, und Container, zur Bereitstellung und Verwaltung der Anwendungsinstanzen, wurden daher jeweils sowohl aus architektonischer Sicht beschrieben und eingeführt sowie hinsichtlich ihrer jeweiligen Grundfunktionalitäten vorgestellt. Weiterhin wurden die im Rahmen dieser Arbeit durchgeführten Erweiterungen und Anpassungen der beiden Hauptkomponenten Winery und Container präsentiert sowie unter Bezugnahme der dadurch jeweils realisierten Forschungsbeiträge diskutiert. Außerdem wurden exemplarische Arbeiten und Projekte vorgestellt, in denen die vorgestellten Prototypen und Konzepte dieser Arbeit wiederverwendet wurden. Beispielsweise zeigen die Arbeiten aus dem Bereich des Quantencomputings, dass die vorgestellten Prototypen und Konzepte auch in anderen Domänen relevant sind und angewendet werden können.

Zusammenfassend validieren die vorgestellten Prototypen die technische Realisierbarkeit der in dieser Arbeit vorgestellten Konzepte und Ansätze. Weiterhin wird die Relevanz sowie die Anwendbarkeit der im Rahmen dieser Arbeit vorgestellten Prototypen, Konzepte und Ansätze, durch ihre jeweilige Wiederverwendung beziehungsweise Anwendung in anderen wissenschaftlichen Arbeiten und Projekten aufgezeigt. Dieses Kapitel stellt somit Forschungsbeitrag 6 der vorliegenden Arbeit dar und schließt die Vorstellungen der einzelnen Forschungsbeiträge (siehe Abschnitt 1.1) dieser Arbeit ab.

Kapitel 9

Zusammenfassung und Ausblick

Die bedarfsgerechte sowie anforderungsabhängige Verteilung von Komponenten über heterogene IT-Umgebungen hinweg, ist ein grundlegender und für die Leistungsfähigkeit sehr wichtiger Aspekt von verteilten Anwendungen. Durch eine stetig zunehmende Menge an generierten und erfassten Daten sowie der wirtschaftlichen Absicht und Notwendigkeit, diese Daten zu analysieren, zu interpretieren sowie zur Eröffnung neuer Wertschöpfungsmöglichkeiten zu nutzen, hat sich der Fokus dabei vor allem auch auf die Betrachtung und Berücksichtigung der zu verarbeitenden Daten sowie den Datenfluss einer Anwendung gerichtet. Weiterhin hat dadurch die Nähe von Komponenten und zu verarbeitenden Daten an Wichtigkeit gewonnen und muss daher bei der Gruppierung und Platzierung von Anwendungskomponenten sowie Daten entsprechend berücksichtigt werden. In dieser Arbeit wurden daher ein Vorgehensmodell sowie verschiedene Konzepte eingeführt, die den Nutzer systematisch sowie mittels Automatisierungen, über den Lebenszyklus einer Anwendung hinweg, hierbei unterstützen.

Dieses Kapitel schließt die vorliegende Arbeit ab. Hierzu werden in Abschnitt 9.1 zunächst die wissenschaftlichen Forschungsbeiträge dieser Arbeit nochmals zusammengefasst und eingeordnet. Anschließend wird in Abschnitt 9.2 ein Ausblick auf mögliche zukünftige Forschungsarbeiten im Kontext der Beiträge und Konzepte dieser Arbeit gegeben.

9.1 Zusammenfassung der Forschungsbeiträge

In Forschungsbeitrag 1 wurde zunächst die Shipping-Methode für das automatisierte Platzieren und Verteilen von Funktionen bzw. Anwendungskomponenten sowie Daten verteilter Anwendungen in heterogenen IT-Umgebungen vorgestellt. Der Fokus bei der Platzierung liegt dabei vor allem auf der Berücksichtigung des Datenflusses einer Anwendung, um eine möglichst hohe Datenlokalität zu erhalten. Die Methode ermöglicht dabei die technologieunabhängige sowie anbieterunabhängige Gruppierung und Platzierung von sowohl ganzer Softwarestacks als auch einzelner Komponenten. Weiterhin wird der gesamte Lebenszyklus einer Anwendung unterstützt. Die Methode umfasst neben der Platzierung der Komponenten auch die Modellierung der Anwendung, die Implementierung der einzelnen Komponenten, die Definition von Richtlinien, die bei der Verteilung und dem Deployment eingehalten werden müssen, sowie das Deployment und Management der Anwendung.

Das zur Realisierung der Shipping-Methode genutzte und im Rahmen dieser Arbeit eingeführte D^3M-Metamodell wurde in Forschungsbeitrag 2 vorgestellt. Dieses wird zur deklarativen Modellierung verteilter Anwendungen genutzt. Weiterhin erlaubt es neben der Modellierung von Komponenten und deren Relationen zueinander auch die Modellierung der zur Durchführung der Shipping-Methode zusätzlich benötigten Informationen, wie zum Beispiel den Datenfaktoren sowie Schnittstellen der einzelnen datenverarbeitenden Anwendungskomponenten. Auf der Grundlage des D^3M-Metamodells wurden drei verschiedene Modellierungsansätze für das Deployment von Anwendungen sowie einzelner Komponenten vorgestellt. Neben der automatisierten Bereitstellung und Verwaltung in einer entfernten Infrastruktur

ermöglichen diese die Verknüpfung dieser Anwendungen oder Komponenten mit in der Zielumgebung verfügbaren Datenquellen. Im Rahmen dieses Beitrags wurde weiterhin ein Konzept präsentiert, um eigenständige Deployment-Pakete, also Deployment-Pakete ohne externe Abhängigkeiten, zu erstellen. Dadurch kann die automatisierte Bereitstellung von Anwendungen auch in Umgebungen ohne Internetzugang, wie zum Beispiel innerhalb von zugriffsbeschränkten Produktionsumgebungen, ermöglicht werden.

Zur Umsetzung der automatisierten Gruppierung und Platzierung von Komponenten einer datenverarbeitenden Anwendung wurde in Forschungsbeitrag 3 ein entsprechendes Konzept vorgestellt. Die hierzu präsentierten Algorithmen berücksichtigen dabei insbesondere den Standort von Datenquellen, die Größe der einzelnen Datensätze, den Datenfluss der Anwendung sowie die Datenfaktoren der datenverarbeitenden Komponenten. Auf diesen sowie weiteren Faktoren, wie zum Beispiel definierten Anforderungen, werden die Komponenten auf verschiedene Standorte verteilt. Dabei kann es sich unter anderem beispielsweise um private sowie öffentliche Clouds, Fog-Clouds sowie Edge-Clouds oder auch IoT-Geräte handeln.

Um den Nutzer dabei zu unterstützen, seine Anwendung komponentenbasiert zu entwickeln und dadurch von den Vorteilen des vorherigen Beitrags leichter zu profitieren, wurde in Forschungsbeitrag 4 ein entsprechendes Programmiermodell sowie eine darauf aufbauende Entwicklungsmethode vorgestellt. Das Ziel dabei ist es, die Kommunikation sowie den Endpunktaustausch zwischen den interagierenden Komponenten einer Anwendung zu abstrahieren, sodass sich der Entwickler auf die Implementierung der reinen Anwendungslogik konzentrieren kann. Hierfür werden auf Basis der in einem D^3M angegebenen Informationen, wie beispielsweise IDs und Schnittstellenbeschreibungen der Komponenten, entsprechende Code-Fragmente zur Implementierung der Komponenten sowie zur Kommunikation generiert.

Beim Betrieb von Anwendungen, insbesondere wenn diese Daten speichern oder verarbeiten, ist die Einhaltung von Gesetzen und internen Richtlinien wichtig. Zum einen, um Verletzungen gesetzgeberischer Vorgaben zu vermeiden und zum anderen, um die Sicherheit der eigenen IT-Infrastruktur sowie

Betriebsgeheimnisse zu wahren. Daher wurde in Forschungsbeitrag 5 ein Konzept zur Definition von D^{3}M-basierten und wiederverwendbaren Sicherheitsrichtlinien sowie ein automatisierter Kontrollmechanismus vorgestellt. Damit kann verhindert werden, dass eine Anwendung versehentlich an einem falschen Ort, beispielsweise außerhalb der eigenen unternehmensweiten IT-Infrastruktur, deployt wird. Weiterhin kann dadurch die Verwendung von veralteter und möglicherweise nicht sicherer Software unterbunden werden.

Abschließend wurde in Forschungsbeitrag 6 die Gesamtarchitektur eines Frameworks vorgestellt, welches die einzelnen Komponenten zur Umsetzung der Forschungsbeiträge integriert und damit die vorgestellte Shipping-Methode realisiert. Die Machbarkeit der jeweils diskutierten Konzepte und Ansätze wurde weiterhin durch die Implementierung von Prototypen, sowohl aus technischer als auch konzeptioneller Sicht validiert. Darüber hinaus wurde durch die Vorstellung wissenschaftlicher Arbeiten und Projekte, welche die im Rahmen dieser Arbeit erarbeiteten Konzepte anwenden und implementierten Prototypen wiederverwenden, die Relevanz sowie Anwendbarkeit dieser Konzepte sowie prototypischen Implementierungen aufgezeigt.

9.2 Ausblick

In diesem Abschnitt werden mögliche zukünftige Forschungsarbeiten skizziert, die aufbauend auf dieser Arbeit durchgeführt werden können. Darüber hinaus werden weitere Bereiche skizziert, in denen die erarbeiteten Konzepte und Prototypen dieser Arbeit eingesetzt werden können.

In Abschnitt 4.2 wurden verschiedene Modellierungskonzepte für Anwendungstopologien vorgestellt: vollständige Anwendungstopologien, konfigurierbare Anwendungstopologien, sowie variable Anwendungstopologien. Da diese verschiedene Vorteile und Eigenschaften haben, muss die Wahl der geeigneten Modellierungsart jeweils abhängig von den Rahmenbedingungen eines konkreten Anwendungsfalls getroffen werden. Falls sich die Rahmenbedingungen jedoch ändern, muss die Modellierung entsprechend manuell angepasst werden. In zukünftigen Arbeiten kann daher untersucht werden,

wie konfigurierbare sowie variable Anwendungstopologien ausgehend von einer vollständigen Anwendungstopologie generiert werden können beziehungsweise, ob eine Transformation zwischen diesen Modellierungsarten realisiert werden kann. Weiterhin kann die Modellierung verschiedener Varianten einer Anwendungstopologie innerhalb eines Modells ein interessanter zukünftiger Forschungsansatz sein. Damit wäre es beispielsweise möglich eine Anwendung so zu modellieren, dass sie sowohl in der Entwicklungsumgebung, Testumgebung als auch in der Produktivumgebung deployt werden kann, ohne separate Anwendungstopologien hierfür erstellen zu müssen.

Im Rahmen dieser Arbeit wird die Gruppierung und Platzierung der Komponenten unter anderem auf Basis des Datenflusses der Anwendung, der Menge an Daten sowie definierten Richtlinien durchgeführt. In zukünftigen Arbeiten kann dieser Ansatz ausgebaut werden und um weitere Kriterien, beispielsweise entstehende Kosten, erweitert werden. Weiterhin kann eine kontinuierliche Überwachung der Verbindungsqualität zwischen den Knoten eines Netzwerks genutzt werden, um eine bessere Platzierung der Komponenten zu erreichen. Bestehende relevante Arbeiten hierzu, die gegebenenfalls integriert werden könnten, wurden in Abschnitt 2.3 vorgestellt. Durch zusätzlich integrierte Kriterien könnte zudem ein interaktives Empfehlungssystem realisiert werden, dass dem Nutzer mehr Möglichkeiten der Einflussnahme auf die Gruppierung und Platzierung der Komponenten gibt, beispielsweise durch die Gewichtung der verschiedenen unterstützten Kriterien.

Die in Abschnitt 7.2 eingeführten Deployment-Regeln ermöglichen die Einhaltung von zuvor festgelegten Richtlinien während der Bereitstellung einer Anwendung. Bei Verstößen wird eine entsprechende Fehlermeldung ausgegeben und der Bereitstellungsvorgang abgebrochen. Der Nutzer muss anschließend die Anwendungstopologie oder nicht eingehaltene Deployment-Regel kontrollieren und anpassen. Dieser Ansatz könnte durch ein Konzept zur automatisierten Erkennung und Umsetzung von Lösungsmöglichkeiten erweitert werden und dem Nutzer dadurch eine Zeitersparnis bieten. Hierzu könnten beispielsweise Data Mining Verfahren eingesetzt werden um bestehende und bewährte Lösungen zu identifizieren und entsprechende Lö-

sungsvarianten davon abzuleiten. Ebenfalls könnten dadurch möglicherweise Deployment-Regeln abgeleitet und dem Nutzer vorgeschlagen werden.

Bei sich verändernden Rahmenbedingungen oder Anforderungen kann eine Anwendungsmigration nötig werden. In solchen Fällen könnten die vorgestellten Forschungsergebnisse der vorliegenden Arbeit, insbesondere die Deployment-Regeln sowie die Gruppierung und Platzierung von Komponenten, genutzt werden. Zum einen, um die Komponenten entsprechend den neuen Anforderungen verteilen und platzieren zu können und zum anderen, um dabei die Einhaltung der definierten Deployment-Regeln weiterhin zu gewährleisten. Die vorgestellten Konzepte können weiterhin im Bereich des Quantencomputings eingesetzt werden. Je nach Anwendungsfall und konkreten Komponenten kann es hier aus Effizienzgründen sinnvoll sein, die klassischen sowie quantenspezifischen Anwendungskomponenten möglichst nah beieinander zu betreiben. Zum anderen kann die Ausführung durch vollständige Deployment-Pakete, im Vergleich zu unvollständigen Deployment-Paketen, beschleunigt werden, insbesondere bei einer wiederholten Ausführung der Anwendung. Ebenfalls kann die modellbasierte Generierung von Code-Fragmenten genutzt werden, um die Kommunikation zwischen den verschiedenen Teilen zu ermöglichen.

Danksagungen

An dieser Stelle möchte ich mich herzlich bei all denjenigen bedanken, die mich während meiner Dissertation unterstützt und begleitet haben. Zuerst möchte ich meinem Doktorvater Prof. Dr. Dr. h. c. Frank Leymann für die wissenschaftliche Betreuung, durchgehende Unterstützung und Ermöglichung meiner Promotion danken. Weiterhin möchte ich mich bei meinem Zweitgutachter Prof. Dr. Guido Wirtz für die Übernahme des Mitberichts bedanken. Mein herzlichster Dank gehört auch allen Kollegen am IAAS für die freundschaftliche Zusammenarbeit, den wissenschaftlichen Austausch und die vielfältige Unterstützung. Insbesondere danke ich dabei Kálmán Képes und Benjamin Weder für die wertvollen Diskussionen und das Korrekturlesen dieser Arbeit. Für die fachliche Begleitung und zahlreiche Ratschläge, insbesondere zu Beginn meiner Promotionszeit, möchte ich ebenso Prof. Dr. Uwe Breitenbücher meinen Dank aussprechen. Zum Schluss möchte ich mich ganz besonders bei meiner Familie bedanken, ohne deren fortwährende Unterstützung und Ermutigung diese Arbeit nicht entstanden wäre.

Literaturverzeichnis

[AAA+17] H. F. Atlam, A. Alenezi, A. Alharthi, R. J. Walters, G. B. Wills. „Integration of Cloud Computing with Internet of Things: Challenges and Open Issues“. In: *Proceedings of the IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2017, S. 670–675 (Zitiert auf Seite 34).

[AAA21] A. Alashaikh, E. Alanazi, A. Al-Fuqaha. „A Survey on the Use of Preferences for Virtual Machine Placement in Cloud Data Centers“. In: *ACM Computing Surveys* 54.5 (2021) (Zitiert auf Seite 45).

[AAR22] N. Alharbe, A. Aljohani, M. A. Rakrouki. „A Fuzzy Grouping Genetic Algorithm for Solving a Real-World Virtual Machine Placement Problem in a Healthcare-Cloud“. In: *Algorithms* 15.4 (2022), S. 1–17 (Zitiert auf Seite 45).

[ABB+01] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, I. Foster. „Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing“. In: *Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies*. IEEE. 2001, S. 13–28 (Zitiert auf Seite 42).

[ABQ13] L. Aniello, R. Baldoni, L. Querzoni. „Adaptive Online Scheduling in Storm“. In: *Proceedings of the 7th ACM International Conference on*

Distributed Event-based Systems (DEBS 2013). ACM, 2013, S. 207–218 (Zitiert auf Seite 47).

[ACB+15] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, R. Buyya. „Big Data computing and clouds: Trends and future directions“. In: *Journal of Parallel and Distributed Computing* 79-80 (2015), S. 3–15 (Zitiert auf Seite 85).

[ÁCJ+16] E. Ábrahám, F. Corzilius, E. B. Johnsen, G. Kremer, J. Mauro. „Zephyrus2: On the Fly Deployment Optimization Using SMT and CP Technologies“. In: *Proceedings of the International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer. 2016, S. 229–245 (Zitiert auf Seite 48).

[ADP17] H. R. Arkian, A. Diyanat, A. Pourkhalili. „MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications“. In: *Journal of Network and Computer Applications* 82 (2017), S. 152–165 (Zitiert auf Seite 46).

[AFG+09] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. Techn. Ber. University of California, Berkeley, 2009 (Zitiert auf Seite 28).

[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“. In: *Commun. ACM* 53.4 (2010), S. 50–58 (Zitiert auf den Seiten 11, 40, 42).

[AFZ97] S. Acharya, M. Franklin, S. Zdonik. „Balancing Push and Pull for Data Broadcast“. In: *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*. ACM, 1997, S. 183–194 (Zitiert auf Seite 139).

[AGLW14] V. Andrikopoulos, S. Gómez Sáez, F. Leymann, J. Wettinger. „Optimal Distribution of Applications in the Cloud“. In: *Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014)*. Springer, Juni 2014, S. 75–90 (Zitiert auf den Seiten 48, 51).

[AGT20] M. S. Aslanpour, S. S. Gill, A. N. Toosi. „Performance Evaluation Metrics for Cloud, Fog and Edge Computing: A Review, Taxonomy, Benchmarks and Standardsf or Future Research“. In: *Internet of Things* 12 (2020), S. 1–24 (Zitiert auf Seite 45).

[AIM10] L. Atzori, A. Iera, G. Morabito. „The Internet of Things: A Survey“. In: *Computer Networks* 54.15 (2010), S. 2787–2805 (Zitiert auf den Seiten 11, 39).

[AIM17] L. Atzori, A. Iera, G. Morabito. „Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm“. In: *Ad Hoc Networks* 56 (2017), S. 122–140 (Zitiert auf Seite 32).

[AKR+19] A. P. Achilleos, K. Kritikos, A. Rossini, G. M. Kapitsaki, J. Domaschka, M. Orzechowski, D. Seybold, F. Griesinger, N. Nikolov, D. Romero et al. „The cloud application modelling and execution language“. In: *Journal of Cloud Computing: Advances, Systems and Applications* 8.1 (2019), S. 1–25 (Zitiert auf Seite 55).

[AL06] Z. Abrams, J. Liu. „Greedy is Good: On Service Tree Placement for In-Network Stream Processing“. In: *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*. IEEE. 2006, S. 72–72 (Zitiert auf den Seiten 46, 48).

[AL13] M. Alicherry, T. V. Lakshman. „Optimizing Data Access Latencies in Cloud Systems by Intelligent Virtual Machine Placement“. In: *Proceedings of the 32nd Annual IEEE International Conference on Computer Communications (INFOCOM 2013)*. IEEE. 2013, S. 647–655 (Zitiert auf Seite 45).

[AMA+18] J. Araujo, P. Maciel, E. Andrade, G. Callou, V. Alves, P. Cunha. „Decision making in cloud environments: an approach based on multiple-criteria decision analysis and stochastic models“. In: *Journal of Cloud Computing* 7.1 (2018), S. 7 (Zitiert auf Seite 45).

[Ame21] A. Amerland. *Das sind die Größten DSGVO-Sünder*. 2021. URL: https://www.springerprofessional.de/compliance/dsgvo/das-sind-die-groessten-dsgvo-suender/19765940 (Zitiert auf Seite 186).

[APHS21] M. Aledhari, M. D. Pierro, M. Hefeida, F. Saeed. „A Deep Learning-Based Data Minimization Algorithm for Fast and Secure Transfer of Big Genomic Datasets“. In: *IEEE Transactions on Big Data* 7.2 (2021), S. 271–284 (Zitiert auf den Seiten 134, 147).

[ARJE22] M. Azimzadeh, A. Rezaee, S. J. Jassbi, M. Esnaashari. „Placement of IoT services in fog environment based on complex network features: a genetic-based approach“. In: *Cluster Computing* 25 (2022), S. 3423–3445 (Zitiert auf Seite 45).

[ARSL14] V. Andrikopoulos, A. Reuter, S. G. Sáez, F. Leymann. „A GENTL Approach for Cloud Application Topologies“. In: *Proceedings of the Third European Conference on Service-Oriented and Cloud Computing (ESOCC 2014)*. Springer. 2014, S. 148–159 (Zitiert auf Seite 53).

[ARXL14] V. Andrikopoulos, A. Reuter, M. Xiu, F. Leymann. „Design Support for Cost-Efficient Application Distribution in the Cloud“. In: *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014)*. 2014, S. 697–704 (Zitiert auf Seite 53).

[ASL13] 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 (CLOUD 2013)*. IEEE, 2013, S. 565–572 (Zitiert auf Seite 48).

[AWS] Amazon Web Services, Inc. *AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning*. URL: https://aws.amazon.com/cloudformation (Zitiert auf Seite 58).

[AWS15] Amazon Web Services, Inc. *AWS Serverless Multi-Tier Architectures: Using Amazon API Gateway and AWS Lambda*. Nov. 2015. URL: https://d0.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf (Zitiert auf Seite 44).

[AWW18] H. F. Atlam, R. J. Walters, G. B. Wills. „Fog Computing and the Internet of Things: A Review“. In: *Big Data and Cognitive Computing* 2.2 (2018), S. 1–18 (Zitiert auf Seite 36).

[BBB+11] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, D. H. Lorenz. „Guaranteeing High Availability Goals for Virtual Machine Placement“. In: *Proceedings of the 31st IEEE International Conference on Distributed Computing Systems (ICDCS 2011)*. IEEE. 2011, S. 700–709 (Zitiert auf Seite 45).

[BBF+14] U. Breitenbücher, T. Binz, C. Fehling, O. Kopp, F. Leymann, M. Wieland. „Policy-Aware Provisioning and Management of Cloud Applications“. In: *International Journal On Advances in Security* 7.1&2 (2014) (Zitiert auf Seite 70).

[BBF+18] A. Bergmayr, U. Breitenbücher, N. Ferry, A. Rossini, A. Solberg, M. Wimmer, G. Kappel. „A Systematic Review of Cloud Modeling Languages“. In: *ACM Computing Surveys (CSUR)* 51.1 (2018), S. 1–38 (Zitiert auf den Seiten 53, 57, 58).

[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 (ICSOC 2013)*. Springer, Dez. 2013, S. 692–695 (Zitiert auf Seite 208).

[BBK+12] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, D. Schumm. „Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA“. In: *On the Move to Meaningful Internet Systems: OTM 2012 (CoopIS 2012)*. Springer, Sep. 2012, S. 416–424 (Zitiert auf Seite 53).

[BBK+13a] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, J. Wettinger. „Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies“. In: *On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013)*. Springer, Sep. 2013, S. 130–148 (Zitiert auf den Seiten 12, 50, 112, 160, 161).

[BBK+13b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, M. Wieland. „Policy-Aware Provisioning of Cloud Applications“. In: *Proceedings of the Seventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2013)*. Xpert Publishing Services, Aug. 2013, S. 86–95 (Zitiert auf Seite 70).

[BBK+14] 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 International Conference on Cloud Engineering (IC2E 2014)*. IEEE, März 2014, S. 87–96 (Zitiert auf den Seiten 57, 215).

[BBK+16] A. Bergmayr, U. Breitenbücher, O. Kopp, M. Wimmer, G. Kappel, F. Leymann. „From Architecture Modeling to Application Provisioning for the Cloud by Combining UML and TOSCA“. In: *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016)*. 2016, S. 97–108 (Zitiert auf Seite 55).

[BBKL13] 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 and Applications (SOCA 2013)*. IEEE, Dez. 2013, S. 126–134 (Zitiert auf den Seiten 66, 85).

[BBKL14] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. „TOSCA: Portable Automated Deployment and Management of Cloud Applications“. In: Advanced Web Services. Springer, Jan. 2014, S. 527–549 (Zitiert auf den Seiten 50, 51, 56).

[BBLS12] T. Binz, G. Breiter, F. Leymann, T. Spatzier. „Portable Cloud Services Using TOSCA“. In: *IEEE Internet Computing* 16.03 (Mai 2012), S. 80–85 (Zitiert auf Seite 56).

[BCM+14] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, S. Swanson. „Near-Data Processing: Insights from a MICRO-46 Workshop“. In: *IEEE Micro* 34.4 (Aug. 2014), S. 36–42 (Zitiert auf Seite 41).

[BCS19] A. Brogi, A. Corradini, J. Soldani. „Estimating costs of multi-component enterprise applications“. In: *Formal Aspects of Computing* 31.4 (2019), S. 421–451 (Zitiert auf Seite 48).

[BCV+19] A. Bernal, M. E. Cambronero, V. Valero, A. Núñez, P. C. Cañizares. „A Framework for Modeling Cloud Infrastructures and User Interactions“. In: *IEEE Access* 7 (2019), S. 43269–43285 (Zitiert auf Seite 55).

[BDMA13] A. Botta, A. Davy, B. Meskill, G. Aceto. „Active Techniques for Available Bandwidth Estimation: Comparison and Application“. In: *Data Traffic Monitoring and Analysis: From Measurement, Classification, and Anomaly Detection to Quality of Experience*. Springer, 2013, S. 28–43 (Zitiert auf Seite 147).

[BDS17] A. Brogi, A. Di Tommaso, J. Soldani. „Validating TOSCA Application Topologies“. In: *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*. 2017, S. 667–678 (Zitiert auf Seite 86).

[BED19] E. H. Bourhim, H. Elbiaze, M. Dieye. „Inter-container Communication Aware Container Placement in Fog Computing“. In: *Proceedings of the 15th International Conference on Network and Service Management (CNSM 2019)*. 2019, S. 1–6 (Zitiert auf Seite 46).

[BEK+16] U. Breitenbücher, C. Endres, K. Képes, O. Kopp, F. Leymann, S. Wagner, J. Wettinger, M. Zimmermann. „The OpenTOSCA Ecosystem - Concepts & Tools“. In: *European Space project on Smart Systems, Big Data, Future Internet - Towards Serving the Grand Societal Challenges - Volume 1* (2016), S. 112–130 (Zitiert auf den Seiten 23, 86, 208, 209).

[BFB+11] P. Banerjee, R. Friedrich, C. Bash, P. Goldsack, B. Huberman, J. Manley, C. Patel, P. Ranganathan, A. Veitch. „Everything as a Service: Powering the New Information Economy“. In: *Computer* 44.3 (2011), S. 36–43 (Zitiert auf Seite 30).

[BFG+95] C. K. Baru, G. Fecteau, A. Goyal, H. Hsiao, A. Jhingran, S. Padmanabhan, G. P. Copeland, W. G. Wilson. „DB2 Parallel Edition“. In: *IBM Systems Journal* 34.2 (1995), S. 292–322 (Zitiert auf Seite 12).

[BFGL19a] A. Brogi, S. Forti, C. Guerrero, I. Lera. „How to Place Your Apps in the Fog - State of the Art and Open Challenges“. In: *CoRR* abs/1901.05717 (2019) (Zitiert auf Seite 45).

[BFGL19b] A. Brogi, S. Forti, C. Guerrero, I. Lera. „Meet Genetic Algorithms in Monte Carlo: Optimised Placement of Multi-Service Applications in the Fog“. In: *Proceedings of the IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2019, S. 13–17 (Zitiert auf Seite 47).

[BGO+16] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes. „Borg, Omega, and Kubernetes“. In: *Communications of the ACM* 59.5 (2016), S. 50–57 (Zitiert auf Seite 161).

[Bin15] T. Binz. „Crawling von Enterprise Topologien zur automatisierten Migration von Anwendungen: eine Cloud-Perspektive“. Dissertation. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2015 (Zitiert auf den Seiten 51, 66).

[BIS+18] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, O. Rana. „The Internet of Things, Fog and Cloud Continuum: Integration and Challenges“. In: *Internet of Things* 3 (2018), S. 134–155 (Zitiert auf Seite 35).

[BK06] A. Brown, A. Keller. „A Best Practice Approach for Automating IT Management Processes“. In: *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*. IEEE, Apr. 2006, S. 33–44 (Zitiert auf Seite 50).

[BKH05] A. Brown, A. Keller, J. Hellerstein. „A model of configuration complexity and its application to a change management system“. In: *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*. IEEE, Mai 2005, S. 631–644 (Zitiert auf Seite 50).

[BKK+14] A. Blehm, V. Kalach, A. Kicherer, G. Murawski, T. Waizenegger, M. Wieland. „Policy-Framework - Eine Methode zur Umsetzung von Sicherheits-Policies im Cloud-Computing“. In: *44. Jahrestagung der Gesellschaft für Informatik, Big Data - Komplexität meistern*. GI, 2014, S. 277–288 (Zitiert auf Seite 201).

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley, 1996 (Zitiert auf Seite 139).

[BMW21] BMW Group. *Autonomous Driving*. 2021. URL: https://www.bmwgroup.com/en/innovation/technologies-and-mobility/autonomes-fahren.html (Zitiert auf Seite 37).

[BMWKa] Bundesministerium für Wirtschaft und Klimaschutz. *SePiA.Pro (Service Plattform für die intelligente Anlagenoptimierung in der Produktion)*. URL: https://www.digitale-technologien.de/DT/Redaktion/DE/Standardartikel/SmartServiceWeltProjekte/Produktion/SSWI_Projekt_SepiaPro.html (Zitiert auf Seite 218).

[BMWKb] Bundesministerium für Wirtschaft und Klimaschutz. *SmartOrchestra*. URL: https://www.digitale-technologien.de/DT/Redaktion/DE/Standardartikel/SmartServiceWeltProjekte/Wohnen_Leben/SSWI_Projekt_SmartOrchestra.html (Zitiert auf Seite 220).

[BMZA12] F. Bonomi, R. Milito, J. Zhu, S. Addepalli. „Fog Computing and Its Role in the Internet of Things“. In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. Association for Computing Machinery, 2012, S. 13–16 (Zitiert auf den Seiten 35, 36).

[BN84] A. D. Birrell, B. J. Nelson. „Implementing Remote Procedure Calls“. In: *ACM Transactions on Computer Systems (TOCS)* 2.1 (1984), S. 39–59 (Zitiert auf Seite 43).

[BO16] B. Burns, D. Oppenheimer. „Design patterns for container-based distributed systems“. In: *Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2016)*. 2016 (Zitiert auf Seite 182).

[BOH+17] F. W. Baumann, U. Odefey, S. Hudert, M. Falkenthal, M. Zimmermann. „Cyber-physical System Control via Industrial Protocol OPC UA“. In: *Proceedings of the Eleventh International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2017)*. Xpert Publishing Services, Nov. 2017, S. 45–49 (Zitiert auf den Seiten 24, 65).

[BPR16] J. O. Benson, J. J. Prevost, P. Rad. „Survey of Automated Software Deployment for Computational and Engineering Research“. In: *Proceedings of the Annual IEEE Systems Conference (SysCon)*. 2016, S. 1–6 (Zitiert auf Seite 57).

[Bre16] U. Breitenbücher. „Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements“. Dissertation. Universität Stuttgart, Fakultaet Informatik, Elektrotechnik und Informationstechnik, 2016 (Zitiert auf den Seiten 92, 129, 170).

[BSG+15] D. Baur, D. Seybold, F. Griesinger, A. Tsitsipas, C. B. Hauser, J. Domaschka. „Cloud Orchestration Features: Are Tools Fit for Purpose?“ In: *Proceedings of the 8th International Conference on Utility and Cloud Computing (UCC)*. 2015, S. 95–101 (Zitiert auf Seite 160).

[BSI] Bundesamt für Sicherheit in der Informationstechnik. *Cloud Computing Grundlagen*. URL: https://www.bsi.bund.de/EN/Topics/CloudComputing/Basics/Basics_node.html (Zitiert auf Seite 28).

[BSLR10] S. Bhattacharya, A. Saifullah, C. Lu, G.-C. Roman. „Multi-Application Deployment in Shared Sensor Networks Based on Quality of Monitoring“. In: *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. 2010, S. 259–268 (Zitiert auf Seite 75).

[BSW+14] A. Brogi, J. Soldani, P. Wang, A. Brogi, J. Soldani, P. Wang. „TOSCA in a Nutshell: Promises and Perspectives“. In: *Service-Oriented and Cloud Computing, volume 8745 of LNCS*. Springer, 2014, S. 171–186 (Zitiert auf Seite 58).

[BTN+14] A. Bergmayr, J. Troya, P. Neubauer, M. Wimmer, G. Kappel. „UML-based Cloud Application Modeling with Libraries, Profiles, and Templates“. In: *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE 2014)*. CEUR-WS.org, Sep. 2014, S. 56–65 (Zitiert auf Seite 55).

[Bur95] M. Burgess. „A Site Configuration Engine“. In: *Computing Systems* 8.2 (1995), S. 309–337 (Zitiert auf Seite 57).

[BWKG14] A. Bergmayr, M. Wimmer, G. Kappel, M. Grossniklaus. „Cloud Modeling Languages by Example“. In: *Proceedings of the 7th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2014)*. IEEE, Nov. 2014, S. 137–146 (Zitiert auf Seite 53).

[Can] Canonical Ltd. *Juju | Operator lifecycle manager for K8s and traditional workloads*. URL: https://juju.is (Zitiert auf Seite 58).

[CCLS11] V. Cardellini, E. Casalicchio, F. Lo Presti, L. Silvestri. „SLA-aware Resource Management for Application Service Providers in the Cloud“. In: *Proceedings of the First International Symposium on Network Cloud Computing and Applications*. IEEE. 2011, S. 20–27 (Zitiert auf Seite 45).

[CD14] Z. Cao, S. Dong. „An energy-aware heuristic framework for virtual machine consolidation in Cloud computing“. In: *The Journal of Supercomputing* 69.1 (2014), S. 429–451 (Zitiert auf Seite 45).

[CDP86] D. W. Cornell, D. M. Dias, S. Y. Philip. „On Multisystem Coupling Through Function Request Shipping“. In: *IEEE Transactions on Software Engineering* 10 (1986), S. 1006–1017 (Zitiert auf den Seiten 12, 42).

[CEM+10] C. Chapman, W. Emmerich, F. G. Márquez, S. Clayman, A. Galis. „Software Architecture Definition for On-demand Cloud Provisioning“. In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010)*. ACM, Juni 2010, S. 61–72 (Zitiert auf Seite 56).

[CGLN16] V. Cardellini, V. Grassi, F. Lo Presti, M. Nardelli. „Optimal Operator Placement for Distributed Stream Processing Applications“. In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (DEBS 2016)*. ACM, 2016, S. 69–80 (Zitiert auf Seite 46).

[CGR+17] A. Chiuchiarelli, R. Gandhi, S. M. Rossi, L. H. H. Carvalho, F. Caggioni, J. C. R. F. Oliveira, J. D. Reis. „Single Wavelength 100G Real-Time Transmission for High-Speed Data Center Communications“. In: *Proceedings of the Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE, 2017, S. 1–3 (Zitiert auf Seite 147).

[CH09] D. Catteddu, G. Hogben. „Cloud Computing: Benefits, risks and recommendations for information security“. In: *Proceedings of the Iberic Web Application Security Conference*. Springer. Nov. 2009 (Zitiert auf Seite 28).

[Cha04] D. A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004 (Zitiert auf Seite 162).

[Che] Chef. *Configuration Management System Software - Chef Infra*. URL: https://www.chef.io/products/chef-infra (Zitiert auf Seite 58).

[CIP04] M. Caporuscio, P. Inverardi, P. Pelliccione. „Compositional Verification of Middleware-Based Software Architecture Descriptions“. In: *Proceedings of the 26th International Conference on Software Engineering*. 2004, S. 221–230 (Zitiert auf Seite 86).

[Cis20] Cisco. *Cisco Annual Internet Report (2018–2023) White Paper*. 2020. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (Zitiert auf Seite 33).

[CJP+11] C. Curino, E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, N. Zeldovich. „Relational Cloud: A Database-as-a-Service for the Cloud“. In: *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR 2011)*. 2011 (Zitiert auf Seite 30).

[CMPW12] P. Costa, M. Migliavacca, P. Pietzuch, A. L. Wolf. „NaaS: Network-as-a-Service in the Cloud“. In: *Proceedings of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12)*. USENIX Association, Apr. 2012 (Zitiert auf Seite 30).

[CNCF] Cloud Native Computing Foundation. *Production-Grade Container Orchestration*. URL: https://kubernetes.io (Zitiert auf Seite 58).

[CP] Cloudify Platform Ltd. *Cloudify Orchestration Platform - Multi Cloud, Cloud Native & Edge*. URL: https://cloudify.co (Zitiert auf Seite 58).

[CV14] A. Chatzistergiou, S. D. Viglas. „Fast Heuristics for Near-Optimal Task Allocation in Data Stream Processing over Clusters“. In: *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management (CIKM '14)*. ACM, 2014, S. 1579–1588 (Zitiert auf Seite 46).

[CZ14] C. P. Chen, C.-Y. Zhang. „Data-intensive applications, challenges, techniques and technologies: A survey on Big Data“. In: *Information Sciences* 275 (2014), S. 314–347 (Zitiert auf den Seiten 12, 40, 135, 185).

[CZL+18] M. Caballer, S. Zala, Á. López García, G. Moltó, P. O. Fernández, M. Velten. „Orchestrating Complex Application Architectures in Heterogeneous Clouds“. In: *Journal of Grid Computing* 16.1 (2018), S. 3–18 (Zitiert auf Seite 160).

[Dat83] C. J. Date. *An Introduction to Database Systems*. Bd. 2. Addison-Wesley, 1983 (Zitiert auf Seite 41).

[DB16] A. V. Dastjerdi, R. Buyya. „Fog Computing: Helping the Internet of Things Realize Its Potential“. In: *Computer* 49.8 (2016), S. 112–116 (Zitiert auf Seite 35).

[DC08] E. Deelman, A. Chervenak. „Data Management Challenges of Data-Intensive Scientific Workflows“. In: *Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid*. 2008, S. 687–692 (Zitiert auf Seite 43).

[DFZ+15] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, B. Hu. „Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends“. In: *Proceedings of the 8th IEEE International Conference on Cloud Computing (CLOUD 2015)*. IEEE. 2015, S. 621–628 (Zitiert auf Seite 30).

[DG08] J. Dean, S. Ghemawat. „MapReduce: Simplified Data Processing on Large Clusters“. In: *Communications of the ACM* 51.1 (2008), S. 107–113 (Zitiert auf Seite 43).

[DGL+17] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina. „Microservices: Yesterday, Today, and Tomorrow“. In: *Present and Ulterior Software Engineering*. Springer, 2017, S. 195–216 (Zitiert auf Seite 160).

[DJV10] T. Delaet, W. Joosen, B. Vanbrabant. „A Survey of System Configuration Tools“. In: *Proceedings of the 24th International Conference on Large Installation System Administration (LISA 2010)*. USENIX, Nov. 2010 (Zitiert auf Seite 57).

[DM12] C. Doukas, I. Maglogiannis. „Bringing IoT and Cloud Computing towards Pervasive Healthcare“. In: *Proceedings of the Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. 2012, S. 922–926 (Zitiert auf Seite 34).

[Doc] Docker, Inc. *Overview of Docker Compose*. URL: https://docs.docker.com/compose (Zitiert auf Seite 58).

[DPA+18] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, J. Vetter. „The future of scientific workflows“. In: *The International Journal of High Performance Computing Applications* 32.1 (2018), S. 159–175 (Zitiert auf Seite 43).

[DSVT07] A. C. Dias Neto, R. Subramanyan, M. Vieira, G. H. Travassos. „A Survey on Model-Based Testing Approaches: A Systematic Review“. In: *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*. ACM, 2007, S. 31–36 (Zitiert auf Seite 86).

[DWX+20] H.-N. Dai, H. Wang, G. Xu, J. Wan, M. Imran. „Big data analytics for manufacturing internet of things: opportunities, challenges and enabling technologies“. In: *Enterprise Information Systems* 14.9-10 (2020), S. 1279–1303 (Zitiert auf Seite 12).

[DX14] C. Dobre, F. Xhafa. „Intelligent services for Big Data science“. In: *Future Generation Computer Systems* 37 (Juli 2014), S. 267–281 (Zitiert auf Seite 33).

[EBF+17] C. Endres, U. Breitenbücher, M. Falkenthal, O. Kopp, F. Leymann, J. Wettinger. „Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications“. In: *Proceedings of the 9th International Conference on Pervasive Patterns and Applications*. Xpert Publishing Services, Feb. 2017, S. 22–27 (Zitiert auf Seite 57).

[EBLW17] C. Endres, U. Breitenbücher, F. Leymann, J. Wettinger. „Anything to Topology - A Method and System Architecture to Topologize Technology-Specific Application Deployment Artifacts“. In: *Proceedings of the 7th*

International Conference on Cloud Computing and Services Science (CLOSER 2017). SCITEPRESS, 2017, S. 180–190 (Zitiert auf Seite 177).

[ECB+11] X. Etchevers, T. Coupaye, F. Boyer, N. de Palma, G. Salaun. „Automated Configuration of Legacy Applications in the Cloud“. In: *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing*. 2011, S. 170–177 (Zitiert auf Seite 54).

[ECBP11] X. Etchevers, T. Coupaye, F. Boyer, N. de Palma. „Self-Configuration of Distributed Applications in the Cloud“. In: *Proceedings of the 4th IEEE International Conference on Cloud Computing (CLOUD 2011)*. 2011, S. 668–675 (Zitiert auf Seite 54).

[EGHS16] C. Ebert, G. Gallardo, J. Hernantes, N. Serrano. „DevOps“. In: *IEEE Software* 33.3 (2016), S. 94–100 (Zitiert auf Seite 57).

[EKK+06] T. Eilam, M. H. Kalantar, A. V. Konstantinou, G. Pacifici, J. A. Pershing, A. Agrawal. „Managing the Configuration Complexity of Distributed Applications in Internet Data Centers“. In: *Communications Magazine* 44.3 (2006), S. 166–177 (Zitiert auf den Seiten 12, 50, 112).

[EL16] R. Eidenbenz, T. Locher. „Task Allocation for Distributed Stream Processing“. In: *Proceedings of the 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, S. 1–9 (Zitiert auf Seite 47).

[ESS+99] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, R. Mckenzie. „Computational RAM: Implementing Processors in Memory“. In: *IEEE Design Test of Computers* 16.1 (1999), S. 32–41 (Zitiert auf Seite 41).

[FAB+11] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, I. Hanschke. „Automation Processes for Enterprise Architecture Management“. In: *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference Workshops*. 2011, S. 340–349 (Zitiert auf Seite 85).

[FBC+16] M. Falkenthal, U. Breitenbücher, M. Christ, C. Endres, A. W. Kempa-Liehr, F. Leymann, M. Zimmermann. „Towards Function and Data Shipping in Manufacturing Environments: How Cloud Technologies leverage the 4th Industrial Revolution“. In: *Proceedings of the 10th Advanced Summer School on Service Oriented Computing*. IBM Research Report, Sep. 2016, S. 16–25 (Zitiert auf den Seiten 12, 23, 40).

[FBG+17] M. Falkenthal, F. W. Baumann, G. Grünert, S. Hudert, F. Leymann, M. Zimmermann. „Requirements and Enforcement Points for Policies in Industrial Data Sharing Scenarios“. In: *Proceedings of the 11th Advanced Summer School on Service Oriented Computing*. IBM Research Division, Nov. 2017, S. 28–40 (Zitiert auf Seite 24).

[FBK+16] M. Falkenthal, U. Breitenbücher, K. Képes, F. Leymann, M. Zimmermann, M. Christ, J. Neuffer, N. Braun, A. W. Kempa-Liehr. „OpenTOSCA for the 4th Industrial Revolution: Automating the Provisioning of Analytics Tools Based on Apache Flink“. In: *Proceedings of the 6th International Conference on the Internet of Things*. ACM, Okt. 2016, S. 179–180 (Zitiert auf den Seiten 23, 219).

[FBKL17] M. P. Fischer, U. Breitenbücher, K. Képes, F. Leymann. „Towards an Approach for Automatically Checking Compliance Rules in Deployment Models“. In: *Proceedings of The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)*. Xpert Publishing Services (XPS), 2017, S. 150–153 (Zitiert auf Seite 202).

[FCR+13] N. Ferry, F. Chauvel, A. Rossini, B. Morin, A. Solberg. „Managing Multi-cloud Systems with CloudMF“. In: *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud 2013)*. ACM, Sep. 2013, S. 38–45 (Zitiert auf Seite 75).

[FCS+18] N. Ferry, F. Chauvel, H. Song, A. Rossini, M. Lushpenko, A. Solberg. „CloudMF: Model-Driven Management of Multi-Cloud Applications“. In: *ACM Transactions on Internet Technology* 18.2 (2018) (Zitiert auf Seite 54).

[FH10] S. Frey, W. Hasselbring. „Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach“. In: *Proceedings of the First International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2010)*. IARIA, Nov. 2010, S. 155–158 (Zitiert auf Seite 56).

[FH11a] S. Frey, W. Hasselbring. „The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications“. In:

International Journal On Advances in Software 4.3&4 (2011), S. 342–353 (Zitiert auf Seite 48).

[FH11b] S. Frey, W. Hasselbring. „The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications“. In: *International Journal on Advances in Software* 4.3 and 4 (2011), S. 342–353 (Zitiert auf Seite 56).

[Fie00] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Dissertation. University of California, Irvine, 2000 (Zitiert auf Seite 76).

[FJK96] M. J. Franklin, B. T. Jónsson, D. Kossmann. „Performance Tradeoffs for Client-server Query Processing“. In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. ACM, 1996, S. 149–160 (Zitiert auf den Seiten 12, 40).

[FMCB17] M. H. Ferdaus, M. Murshed, R. N. Calheiros, R. Buyya. „An algorithm for network and data-aware placement of multi-tier applications in cloud data centers“. In: *Journal of Network and Computer Applications* 98 (2017), S. 65–83 (Zitiert auf Seite 47).

[FME12] J. Fischer, R. Majumdar, S. Esmaeilsabzali. „Engage: A Deployment Management System“. In: *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2012, S. 263–274 (Zitiert auf Seite 75).

[FRC+13] N. Ferry, A. Rossini, F. Chauvel, B. Morin, A. Solberg. „Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems“. In: *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD 2013)*. IEEE, Juli 2013, S. 887–894 (Zitiert auf Seite 54).

[FRL+14] A. Fernández, S. del Río, V. López, A. Bawakid, M. J. del Jesus, J. M. Benítez, F. Herrera. „Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks“. In: *WIREs Data Mining and Knowledge Discovery* 4.5 (2014), S. 380–409 (Zitiert auf Seite 85).

[FS05] D. F. Ferguson, M. L. Stockton. „Service-oriented architecture: Programming model and product architecture“. In: *IBM Systems Journal* 44.4 (2005), S. 753–780 (Zitiert auf Seite 163).

[FSB13] L. Fischer, T. Scharrenbach, A. Bernstein. „Scalable Linked Data Stream Processing via Network-Aware Workload Scheduling“. In: *Proceedings of the 9th International Conference on Scalable Semantic Web Knowledge Base Systems*. Bd. 1046. 2013, S. 81–96 (Zitiert auf den Seiten 46, 47).

[FZ15] X. Fu, C. Zhou. „Virtual machine selection and placement for dynamic consolidation in Cloud computing environment“. In: *Frontiers of Computer Science* 9.2 (2015), S. 322–330 (Zitiert auf Seite 45).

[GAK15] M. Gao, G. Ayers, C. Kozyrakis. „Practical Near-Data Processing for In-memory Analytics Frameworks“. In: *Proceedings of the 2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 2015, S. 113–124 (Zitiert auf Seite 41).

[Gar17] J. L. García-Dorado. „Bandwidth Measurements within the Cloud: Characterizing Regular Behaviors and Correlating Downtimes“. In: *ACM Transactions on Internet Technology* 17.4 (2017), S. 1–25 (Zitiert auf Seite 147).

[GBAP12] D. Gachet, M. de Buenaga, F. Aparicio, V. Padrón. „Integrating Internet of Things and Cloud Computing for Health Services Provisioning: The Virtual Cloud Carer Project“. In: *Proceedings of the Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. 2012, S. 918–921 (Zitiert auf Seite 34).

[GBF+16] J. Guth, U. Breitenbücher, M. Falkenthal, F. Leymann, L. Reinfurt. „Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture“. In: *Proceedings of the International Conference on Cloudification of the Internet of Things (CIoT 2016)*. IEEE, 2016 (Zitiert auf Seite 162).

[GD18] M. Gusev, S. Dustdar. „Going Back to the Roots—The Evolution of Edge Computing, An IoT Perspective“. In: *IEEE Internet Computing* 22.2 (2018), S. 5–15 (Zitiert auf Seite 37).

[GDV20] T. Goethals, F. De Turck, B. Volckaert. „Near real-time optimization of fog service placement for responsive edge computing“. In: *Journal of Cloud Computing* 9.1 (2020), S. 1–17 (Zitiert auf Seite 46).

[GES+11] G. Gonçalves, P. Endo, M. Santos, D. Sadok, J. Kelner, B. Melander, J.-E. Mangs. „CloudML: An Integrated Language for Resource, Service and Request Description for D-Clouds“. In: *Proceedings of the Third International Conference on Cloud Computing Technology and Science (CloudCom 2011)*. IEEE, Nov. 2011, S. 399–406 (Zitiert auf Seite 56).

[Get12] V. Getov. „Security as a Service in Smart Clouds – Opportunities and Concerns“. In: *Proceedings of the 36th IEEE Annual Computer Software and Applications Conference*. IEEE. 2012, S. 373–379 (Zitiert auf Seite 30).

[GFZ12] Z. Guo, G. Fox, M. Zhou. „Investigation of Data Locality in MapReduce“. In: *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*. IEEE Computer Society. 2012, S. 419–426 (Zitiert auf den Seiten 41, 43).

[GGL03] S. Ghemawat, H. Gobioff, S.-T. Leung. „The Google File System“. In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. 2003, S. 20–43 (Zitiert auf Seite 43).

[GGQ+13] Y. Gao, H. Guan, Z. Qi, Y. Hou, L. Liu. „A multi-objective ant colony system algorithm for virtual machine placement in cloud computing“. In: *Journal of Computer and System Sciences* 79.8 (2013), S. 1230–1242 (Zitiert auf Seite 45).

[GH15] A. Gandomi, M. Haider. „Beyond the hype: Big data concepts, methods, and analytics“. In: *International Journal of Information Management* 35.2 (2015), S. 137–144 (Zitiert auf den Seiten 11, 185).

[GHH+17] Y. Guo, X. Hu, B. Hu, J. Cheng, M. Zhou, R. Y. K. Kwok. „Mobile Cyber Physical Systems: Current Challenges and Future Networking Applications“. In: *IEEE Access* 6 (2017), S. 12360–12368 (Zitiert auf den Seiten 11, 28, 40).

[GHJ+09] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, S. Sengupta. „VL2: A Scalable and Flexible Data Center Network“. In: *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*. ACM, 2009, S. 51–62 (Zitiert auf Seite 147).

[GHJV94] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, Okt. 1994 (Zitiert auf den Seiten 68, 166).

[GHS10] S. Günther, M. Haupt, M. Splieth. *Very Large Business Applications Lab: Utilizing Internal Domain-Specific Languages for Deployment and Maintenance of IT Infrastructures*. Technical report. Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 2010 (Zitiert auf Seite 50).

[GMMC13a] J. Guillén, J. Miranda, J. M. Murillo, C. Canal. „A service-oriented framework for developing cross cloud migratable software“. In: *The Journal of Systems and Software* 86.9 (Sep. 2013), S. 2294–2308 (Zitiert auf Seite 55).

[GMMC13b] J. Guillén, J. Miranda, J. Murillo, C. Canal. „A UML Profile for Modeling Multicloud Applications“. In: *Proceedings of the Second European Conference on Service-Oriented and Cloud Computing (ESOCC 2013)*. Springer, Sep. 2013, S. 180–187 (Zitiert auf Seite 55).

[GÖÖ16] B. Gedik, H. G. Özsema, Ö. Öztürk. „Pipelined fission for stream programs with dynamic selectivity and partitioned state“. In: *Journal of Parallel and Distributed Computing* 96 (2016), S. 106–120 (Zitiert auf den Seiten 46, 47).

[GPGV14] V. Gunes, S. Peter, T. Givargis, F. Vahid. „A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems“. In: *KSII Transactions on Internet and Information Systems* (2014) (Zitiert auf den Seiten 11, 39).

[GPR+19] G. Grünert, M. Portail, Y. Ritter, G. M. O. Hönig, A. Riegg, L. Reinfurt, C. Bauer, H. Hemsen, N. Saini, M. Falkenthal, M. Zimmermann. *SePiA.Pro - Service Plattform für die intelligente Anlagenoptimierung in der Produktion: ein Verbundprojekt des Technologieprogramms Smart Service Welt - Internetbasierte Dienste für die Industrie*. Techn. Ber. 2019 (Zitiert auf Seite 218).

[Gra86] J. Gray. „Why Do Computers Stop and What Can Be Done About It?“ In: *Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems*. 1986, S. 3–12 (Zitiert auf Seite 51).

[GZG+16] L. Gu, D. Zeng, S. Guo, Y. Xiang, J. Hu. „A General Communication Cost Optimization Framework for Big Data Stream Processing in Geo-Distributed Data Centers“. In: *IEEE Transactions on Computers* 65.1 (2016), S. 19–29 (Zitiert auf Seite 48).

[HAR14] S. A. Haque, S. M. Aziz, M. Rahman. „Review of Cyber-Physical System in Healthcare“. In: *International Journal of Distributed Sensor Networks* 10.4 (2014), S. 217415 (Zitiert auf den Seiten 11, 28, 40).

[HAW11] H. Herry, P. Anderson, G. Wickler. „Automated Planning for Configuration Changes“. In: *Proceedings of the 25th International Conference on Large Installation System Administration (LISA 2011)*. USENIX, Dez. 2011, S. 57–68 (Zitiert auf Seite 57).

[HBB+21] L. Harzenetter, T. Binz, U. Breitenbücher, F. Leymann, M. Wurster. „Automated Generation of Management Workflows for Running Applications by Deriving and Enriching Instance Models“. In: *Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021)*. SciTePress, 2021, S. 99–110 (Zitiert auf Seite 215).

[HBBL14] P. Hirmer, U. Breitenbücher, T. Binz, F. Leymann. „Automatic Topology Completion of TOSCA-based Cloud Applications“. In: *GI-Jahrestagung*. Bd. P-251. GI, 2014, S. 247–258 (Zitiert auf den Seiten 141, 144, 178).

[HBLE14] H. Holm, M. Buschle, R. Lagerström, M. Ekstedt. „Automatic data collection for enterprise architecture models“. In: *Software & Systems Modeling* 13.2 (2014), S. 825–841 (Zitiert auf Seite 85).

[HC] HashiCorp. *Terraform by HashiCorp*. URL: https://www.terraform.io (Zitiert auf Seite 58).

[HC01] G. T. Heineman, W. T. Councill, Hrsg. *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, 2001 (Zitiert auf Seite 12).

[HILZ22] M. Hirsch, D. Iglezakis, F. Leymann, M. Zimmermann. „The ReSUS Project - Infrastructure for Sharing Research Software“. In: *E-Science-Tage 2021: Share Your Research Data*. heiBOOKS, 2022, S. 267–276 (Zitiert auf den Seiten 25, 217).

[HLT11] M. Hamdaqa, T. Livogiannis, L. Tahvildari. „A Reference Model for Developing Cloud Applications“. In: *Proceedings of the 1st International Conference on Cloud Computing and Services Science (CLOSER 2011)*. SciTePress, 2011, S. 98–103 (Zitiert auf Seite 54).

[HM21] A. Hedhli, H. Mezni. „A Survey of Service Placement in Cloud Environments“. In: *Journal of Grid Computing* 19.3 (2021), S. 1–32 (Zitiert auf Seite 45).

[HMD+15] L. M. Hillah, A.-P. Maesano, F. De Rosa, L. Maesano, M. Lettere, R. Fontanelli. „Service functional test automation“. In: *Proceedings of the 10th Workshop on System Testing and Validation*. 2015 (Zitiert auf Seite 86).

[HNYL17] Z. Hao, E. Novak, S. Yi, Q. Li. „Challenges and Software Architecture for Fog Computing“. In: *IEEE Internet Computing* 21.2 (2017), S. 44–53 (Zitiert auf Seite 37).

[HPO16] M. Hermann, T. Pentek, B. Otto. „Design Principles for Industrie 4.0 Scenarios“. In: *Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS 2016)*. IEEE, 2016, S. 3928–3937 (Zitiert auf Seite 70).

[HT15] M. Hamdaqa, L. Tahvildari. „StratusML: A Layered Cloud Modeling Framework“. In: *Proceedings of the IEEE International Conference on Cloud Engineering*. 2015, S. 96–105 (Zitiert auf Seite 54).

[HYX14] W. He, G. Yan, L. D. Xu. „Developing Vehicular Data Cloud Services in the IoT Environment“. In: *IEEE Transactions on Industrial Informatics* 10.2 (2014), S. 1587–1595 (Zitiert auf Seite 34).

[HYZ22] L. Heng, G. Yin, X. Zhao. „Energy aware cloud-edge service placement approaches in the Internet of Things communications“. In: *International Journal of Communication Systems* 35.1 (2022), S. 1–23 (Zitiert auf Seite 45).

[IEE18] IEEE Standards Association. „IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing“. In: *IEEE Std 1934-2018* (2018) (Zitiert auf Seite 36).

[INS+14] C. Inzinger, S. Nastic, S. Sehic, M. Vögler, F. Li, S. Dustdar. „MADCAT: A Methodology for Architecture and Deployment of Cloud Application Topologies“. In: *Proceedings of the 8th IEEE International Symposium on Service Oriented System Engineering*. 2014, S. 13–22 (Zitiert auf Seite 51).

[IOY+11] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema. „Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing“. In: *IEEE Transactions on Parallel and Distributed Systems* 22.6 (2011), S. 931–945 (Zitiert auf Seite 28).

[Jaz14] N. Jazdi. „Cyber Physical Systems in the Context of Industry 4.0“. In: *Proceedings of the IEEE International Conference on Automation, Quality and Testing, Robotics*. 2014, S. 1–4 (Zitiert auf Seite 65).

[JD02] M. Jain, C. Dovrolis. „End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput“. In: *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2002, S. 295–308 (Zitiert auf Seite 147).

[JE19a] A. Jumagaliyev, Y. Elkhatib. „A Modelling Language to Support the Evolution of Multi-tenant Cloud Data Architectures“. In: *Proceedings of the 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2019)*. 2019, S. 139–149 (Zitiert auf Seite 54).

[JE19b] A. Jumagaliyev, Y. Elkhatib. „CadaML: A Modeling Language for Multi-Tenant Cloud Application Data Architectures“. In: *Proceedings of the 12th IEEE International Conference on Cloud Computing (CLOUD 2019)*. 2019, S. 430–434 (Zitiert auf Seite 54).

[JSW20] J. Jasperneite, T. Sauter, M. Wollschlaeger. „Why We Need Automation Models: Handling Complexity in Industry 4.0 and the Internet of Things“. In: *IEEE Industrial Electronics Magazine* 14.1 (2020), S. 29–40 (Zitiert auf Seite 50).

[KAEM13] S. Kaisler, F. Armour, J. A. Espinosa, W. Money. „Big Data: Issues and Challenges Moving Forward“. In: *Proceedings of the 46th Hawaii International Conference on System Sciences*. IEEE, 2013, S. 995–1004 (Zitiert auf Seite 185).

[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 (ICSOC 2013)*. Springer, Dez. 2013, S. 700–704 (Zitiert auf Seite 209).

[KBF+17] K. Képes, U. Breitenbücher, M. P. Fischer, F. Leymann, M. Zimmermann. „Policy-Aware Provisioning Plan Generation for TOSCA-based Applications“. In: *Proceedings of The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)*. Xpert Publishing Services, Sep. 2017, S. 142–149 (Zitiert auf den Seiten 24, 201, 215).

[KBKL18] C. Krieger, U. Breitenbücher, K. Képes, F. Leymann. „An Approach to Automatically Check the Compliance of Declarative Deployment Models“. In: *Papers from the 12th Advanced Summer School on Service-Oriented Computing (SummerSoC 2018)*. IBM Research Division, 2018, S. 76–89 (Zitiert auf Seite 202).

[KBL17] K. Képes, U. Breitenbücher, F. Leymann. „The SePaDe System: Packaging Entire XaaS Layers for Automatically Deploying and Managing Applications“. In: *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*. SciTePress, 2017, S. 626–635 (Zitiert auf Seite 178).

[KGSS12] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, I. Sommerville. „The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise“. In: *Software: Practice and Experience* 42.4 (2012), S. 447–465 (Zitiert auf Seite 48).

[KKW+17] L. Kong, M. K. Khan, F. Wu, G. Chen, P. Zeng. „Millimeter-Wave Wireless Communications for IoT-Cloud Supported Autonomous Vehicles: Overview, Design, and Challenges“. In: *IEEE Communications Magazine* 55.1 (2017), S. 62–68 (Zitiert auf Seite 34).

[KL10] K. Kumar, Y.-H. Lu. „Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?“ In: *Computer* 43.4 (2010), S. 51–56 (Zitiert auf Seite 45).

[KLZ20] K. Képes, F. Leymann, M. Zimmermann. „Situation-Aware Updates for Cyber-Physical Systems“. In: *Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2020)*. Springer International Publishing, Dez. 2020, S. 12–32 (Zitiert auf Seite 25).

[KMH+21] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, R. Prodan. „Cloud, Fog, or Edge: Where to Compute?“ In: *IEEE Internet Computing* 25.4 (2021), S. 30–36 (Zitiert auf Seite 45).

[KMM18] M. Kalske, N. Mäkitalo, T. Mikkonen. „Challenges When Moving from Monolith to Microservice Architecture“. In: *Current Trends in Web Engineering*. Springer, 2018, S. 32–47 (Zitiert auf Seite 160).

[Kos00] D. Kossmann. „The State of the Art in Distributed Query Processing“. In: *ACM Computing Surveys* 32.4 (2000), S. 422–469 (Zitiert auf Seite 139).

[KPL+09] S. Kaune, K. Pussep, C. Leng, A. Kovacevic, G. Tyson, R. Steinmetz. „Modelling the Internet Delay Space Based on Geographical Locations“. In: *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2009, S. 301–310 (Zitiert auf Seite 135).

[KSMM19] K. Kritikos, P. Skrzypek, A. Moga, O. Matei. „Towards the Modelling of Hybrid Cloud Applications“. In: *Proceedings of the 12th IEEE International Conference on Cloud Computing (CLOUD 2019)*. 2019, S. 291–295 (Zitiert auf Seite 55).

[KWXD17] M. Khan, X. Wu, X. Xu, W. Dou. „Big Data Challenges and Opportunities in the Hype of Industry 4.0“. In: *Proceedings of the IEEE International Conference on Communications (ICC 2017)*. 2017, S. 1–6 (Zitiert auf den Seiten 11, 135, 185).

[KYT14] J. Kuo, H. Yang, M. Tsai. „Optimal Approximation Algorithm of Virtual Machine Placement for Data Latency Minimization in Cloud Systems“. In: *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM 2014)*. IEEE. 2014, S. 1303–1311 (Zitiert auf Seite 45).

[LBK15] J. Lee, B. Bagheri, H.-A. Kao. „A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems“. In: *Manufacturing letters* 3 (2015), S. 18–23 (Zitiert auf Seite 65).

[LDK09] J. Li, A. Deshpande, S. Khuller. „Minimizing Communication Cost in Distributed Multi-query Processing“. In: *Proceedings of the 25th International Conference on Data Engineering*. IEEE. 2009, S. 772–783 (Zitiert auf Seite 46).

[Ley09] F. Leymann. „Cloud Computing: The Next Revolution in IT“. In: *Proceedings of the 52th Photogrammetric Week*. Wichmann Verlag, Sep. 2009, S. 3–12 (Zitiert auf den Seiten 11, 28, 39, 50).

[LFM+11] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, S. Dustdar. „Moving Applications to the Cloud: An Approach based on Application Model Enrichment“. In: *International Journal of Cooperative Information Systems* 20.3 (2011), S. 307–356 (Zitiert auf den Seiten 48, 51, 54).

[LGJ18] I. Lera, C. Guerrero, C. Juiz. „Comparing Centrality Indices for Network Usage Optimization of Data Placement Policies in Fog Devices“. In: *Proceedings of the Third IEEE International Conference on Fog and Mobile Edge Computing (FMEC 2018)*. IEEE, 2018, S. 115–122 (Zitiert auf Seite 49).

[LHG+19] A. Liebing, J. Heidrich, T. Günther, M. Hahn, D. Olschewski, M. Virtel, P. Niehues. *SmartOrchestra - Orchestrierung cyberphysischer Systeme und Services*. Techn. Ber. 2019 (Zitiert auf Seite 220).

[LLS10] G. T. Lakshmanan, Y. Li, R. Strom. „Placement of Replicated Tasks for Distributed Stream Processing Systems“. In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems (DEBS 2010)*. ACM, 2010, S. 128–139 (Zitiert auf Seite 47).

[LQLW13] X. Li, Z. Qian, S. Lu, J. Wu. „Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center“. In: *Mathematical and Computer Modelling* 58.5 (2013), S. 1222–1235 (Zitiert auf Seite 45).

[LRS02] F. Leymann, D. Roller, M.-T. Schmidt. „Web services and business process management“. In: *IBM Systems Journal* 41.2 (2002), S. 198–211 (Zitiert auf Seite 75).

[LSS+13] H. Lu, M. Shtern, B. Simmons, M. Smit, M. Litoiu. „Pattern-Based Deployment Service for Next Generation Clouds“. In: *Proceedings of the IEEE Ninth World Congress on Services (SERVICES 2013)*. IEEE, Juni 2013, S. 464–471 (Zitiert auf Seite 55).

[LT19] A. Laghrissi, T. Taleb. „A Survey on the Placement of Virtual Resources and Virtual Network Functions“. In: *IEEE Communications Surveys Tutorials* 21.2 (2019), S. 1409–1434 (Zitiert auf Seite 45).

[LTX15] T. Li, J. Tang, J. Xu. „A predictive scheduling framework for fast and distributed stream data processing“. In: *Proceedings of the IEEE International Conference on Big Data (Big Data 2015)*. IEEE. 2015, S. 333–338 (Zitiert auf Seite 46).

[MA21] M. Mubarkoot, J. Altmann. „Towards Software Compliance Specification and Enforcement Using TOSCA“. In: *Proceedings of the International Conference on the Economics of Grids, Clouds, Systems, and Services*. Springer. 2021, S. 168–177 (Zitiert auf Seite 201).

[Mah11] Z. Mahmood. „Data Location and Security Issues in Cloud Computing“. In: *Proceedings of the International Conference on Emerging Intelligent Data and Web Technologies*. IEEE. 2011, S. 49–54 (Zitiert auf Seite 71).

[Man22] Z. Á. Mann. „Decentralized Application Placement in Fog Computing“. In: *Transactions on Parallel and Distributed Systems* 33.12 (2022), S. 3262–3273 (Zitiert auf Seite 45).

[Mar18] B. Marr. *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. 2018. URL: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read (Zitiert auf den Seiten 11, 39).

[MB16] O. Mesnard, L. A. Barba. „Reproducible and replicable CFD: it's harder than you think“. In: *arXiv preprint arXiv:1605.04339* (2016) (Zitiert auf Seite 74).

[MB17] P. Mach, Z. Becvar. „Mobile Edge Computing: A Survey on Architecture and Computation Offloading“. In: *IEEE Communications Surveys Tutorials* 19.3 (2017), S. 1628–1656 (Zitiert auf Seite 45).

[MDW+00] V. Machiraju, M. Dekhil, K. Wurster, P. Garg, M. Griss, J. Holland. „Towards Generic Application Auto-discovery“. In: *Network Operations and Management Symposium*. 2000, S. 75–87 (Zitiert auf Seite 85).

[Meu95] R. Meunier. „The Pipes and Filters Architecture“. In: *Pattern Languages of Program Design*. 1995, S. 427–440 (Zitiert auf Seite 137).

[MG11] P. Mell, T. Grance. *The NIST Definition of Cloud Computing*. 2011 (Zitiert auf Seite 28).

[MGAD13] M. Miglierina, G. P. Gibilisco, D. Ardagna, E. Di Nitto. „Model Based Control for Multi-cloud Applications“. In: *Proceedings of the 5th International Workshop on Modeling in Software Engineering (MiSE 2013)*. IEEE, 2013, S. 37–43 (Zitiert auf Seite 48).

[MGGA19] O. Mutlu, S. Ghose, J. Gómez-Luna, R. Ausavarungnirun. „Processing Data Where It Makes Sense: Enabling In-Memory Computation“. In: *Microprocessors and Microsystems* 67 (2019), S. 28–41 (Zitiert auf Seite 41).

[Mic] Microsoft. *Azure Resource Manager | Microsoft Azure*. URL: https://azure.microsoft.com/en-us/features/resource-manager (Zitiert auf Seite 58).

[Mic18] Microsoft. *2019 Manufacturing Trends Report*. 2018. URL: https://info.microsoft.com/rs/157-GQE-382/images/EN-US-CNTNT-Report-2019-Manufacturing-Trends.pdf (Zitiert auf Seite 33).

[Mic22] Microsoft. *Asynchronous Request-Reply pattern*. 2022. URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/async-request-reply (Zitiert auf Seite 181).

[MKL09] T. Mather, S. Kumaraswamy, S. Latif. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. O'Reilly Media, Inc., 2009 (Zitiert auf Seite 70).

[MMM10] F. Machida, Masahiro Kawato, Y. Maeno. „Redundant Virtual Machine Placement for Fault-tolerant Consolidated Server Clusters“. In: *Proceedings of the IEEE Network Operations and Management Symposium (NOMS 2010)*. IEEE. 2010, S. 32–39 (Zitiert auf Seite 45).

[MMP+22] Z. Á. Mann, A. Metzger, J. Prade, R. Seidl, K. Pohl. „Cost-optimized, data-protection-aware offloading between an edge data center and the cloud“. In: *Transactions on Services Computing* (2022), S. 1–14 (Zitiert auf Seite 36).

[MMS+17] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury, V. Kumar. „Security and Privacy in Fog Computing: Challenges“. In: *IEEE Access* 5 (2017), S. 19293–19304 (Zitiert auf Seite 37).

[Mob02] R. Mobley. *An Introduction to Predictive Maintenance*. 2nd. Plant Engineering. Elsevier, 2002 (Zitiert auf Seite 11).

[Mor16] K. Morris. *Infrastructure as Code: Managing Servers in the Cloud*. 1st. O'Reilly Media, Inc., 2016 (Zitiert auf Seite 57).

[Mow09] M. Mowbray. „The Fog over the Grimpen Mire: Cloud Computing and the Law“. In: *Scripted Journal of Law, Technology and Society* 6.1 (2009), S. 132–146 (Zitiert auf Seite 186).

[MPZ10] X. Meng, V. Pappas, L. Zhang. „Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement“. In: *Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM 2010)*. IEEE. 2010, S. 1–9 (Zitiert auf Seite 46).

[MR12] M. Menzel, R. Ranjan. „CloudGenius: Decision Support for Web Server Cloud Migration“. In: *Proceedings of the 21st International Conference on World Wide Web*. ACM, 2012, S. 979–988 (Zitiert auf Seite 48).

[MSKV19] S. Mazumdar, D. Seybold, K. Kritikos, Y. Verginadis. „A survey on data storage and placement methodologies for Cloud-Big Data ecosystem“. In: *Journal of Big Data* 6.1 (2019), S. 1–37 (Zitiert auf Seite 45).

[MZ19] A. E. Malki, U. Zdun. „Guiding Architectural Decision Making on Service Mesh Based Microservice Architectures“. In: *Proceedings of the European Conference on Software Architecture (ECSA 2019)*. Springer, 2019, S. 3–19 (Zitiert auf Seite 182).

[Nai17] N. Naik. „Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP“. In: *Proceedings of the IEEE International Systems Engineering Symposium (ISSE 2017)*. 2017, S. 1–7 (Zitiert auf Seite 162).

[New15] S. Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, 2015 (Zitiert auf den Seiten 39, 69).

[NLBR18] M. I. Naas, L. Lemarchand, J. Boukhobza, P. Raipin. „A Graph Partitioning-Based Heuristic for Runtime IoT Data Placement Strategies in a Fog Infrastructure". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 2018, S. 767–774 (Zitiert auf Seite 49).

[NLPH12] D. K. Nguyen, F. Lelli, M. P. Papazoglou, W.-J. van den Heuvel. „Blueprinting Approach in Support of Cloud Computing". In: *Future Internet* 4.1 (2012), S. 322–346 (Zitiert auf Seite 53).

[NLT+11] D. K. Nguyen, F. Lelli, Y. Taher, M. Parkin, M. P. Papazoglou, W.-J. van den Heuvel. „Blueprint Template Support for Engineering Cloud-Based Services". In: *Towards a Service-Based Internet - 4th European Conference (ServiceWave 2011)*. Springer, Okt. 2011, S. 26–37 (Zitiert auf Seite 53).

[NPBL17] M. I. Naas, P. R. Parvedy, J. Boukhobza, L. Lemarchand. „iFogStor: An IoT Data Placement Strategy for Fog Infrastructure". In: *Proceedings of the 1st IEEE International Conference on Fog and Edge Computing (ICFEC 2017)*. IEEE, 2017, S. 97–104 (Zitiert auf Seite 49).

[NTAS] Northern.tech AS. *CFEngine - Automate your infrastructure, security & compliance*. URL: https://cfengine.com (Zitiert auf Seite 58).

[OAS07] OASIS. *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2007 (Zitiert auf Seite 207).

[OAS13a] OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2013 (Zitiert auf den Seiten 56, 97, 170, 182).

[OAS13b] OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS). 2013 (Zitiert auf den Seiten 56, 92, 129, 170, 182).

[OAS19] OASIS. *TOSCA Simple Profile in YAML Version 1.2*. Organization for the Advancement of Structured Information Standards (OASIS). 2019 (Zitiert auf Seite 116).

[OBG+18] U. Odefey, F. Baumann, G. Grünert, S. Hudert, M. Zimmermann, M. Falkenthal, F. Leymann. „Manufacturing Smart Services for automotive production lines“. In: *18. Internationales Stuttgarter Symposium*. Springer, Mai 2018, S. 813–825 (Zitiert auf Seite 25).

[OCP+06] D. Oppenheimer, B. Chun, D. Patterson, A. C. Snoeren, A. Vahdat. „Service Placement in a Shared Wide-Area Platform“. In: *Proceedings of the USENIX Annual Technical Conference, General Track*. 2006, S. 273–288 (Zitiert auf Seite 48).

[OFC17] OpenFog Consortium. *The OpenFog Consortium Reference Architecture: Executive Summary*. 2017. URL: https://www.iiconsortium.org/pdf/OpenFog-Reference-Architecture-Executive-Summary.pdf (Zitiert auf Seite 36).

[OJ15] M. Obitko, V. Jirkovský. „Big Data Semantics in Industry 4.0“. In: *Proceedings of the 7th International Conference on Industrial Applications of Holonic and Multi-Agent Systems - Volume 9266*. Springer, 2015, S. 217–229 (Zitiert auf den Seiten 135, 185).

[OMG07] OMG. *OMG Unified Modeling Language (UML)*. Object Management Group (OMG). 2007 (Zitiert auf Seite 55).

[OMG11] OMG. *Business Process Model and Notation (BPMN) Version 2.0*. Object Management Group (OMG). 2011 (Zitiert auf Seite 207).

[OMG12] OMG. *CORBA 3.3*. Object Management Group (OMG). 2012 (Zitiert auf den Seiten 43, 181).

[Opp03] D. Oppenheimer. „The importance of understanding distributed system configuration“. In: *Proceedings of the 2003 Conference on Human Factors in Computer Systems Workshop (CHI 2003)*. ACM, Apr. 2003 (Zitiert auf Seite 50).

[Ora10] Oracle. *Java Remote Method Invocation - Distributed Computing for Java*. 2010. URL: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html (Zitiert auf den Seiten 43, 181).

[OSF] OpenStack Foundation. *Heat – OpenStack*. URL: https://wiki.openstack.org/wiki/Heat (Zitiert auf Seite 58).

[PAC+97] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, K. Yelick. „A Case for Intelligent RAM“. In: *IEEE micro* 17.2 (1997), S. 34–44 (Zitiert auf Seite 41).

[PBL+17] A. Panarello, U. Breitenbücher, F. Leymann, A. Puliafito, M. Zimmermann. „Automating the Deployment of Multi-Cloud Applications in Federated Cloud Environments“. In: *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2017)*. ACM, Mai 2017, S. 1–8 (Zitiert auf den Seiten 24, 86, 141).

[PDMC03] R. Prasad, C. Dovrolis, M. Murray, K. Claffy. „Bandwidth Estimation: Metrics, Measurement Techniques, and Tools“. In: *IEEE Network* 17.6 (2003), S. 27–35 (Zitiert auf Seite 147).

[PK98] V. A. Pham, A. Karmouch. „Mobile Software Agents: An Overview“. In: *IEEE Communications Magazine* 36.7 (1998), S. 26–37 (Zitiert auf Seite 43).

[PTBP08] P. Pelliccione, M. Tivoli, A. Bucchiarone, A. Polini. „An architectural approach to the correct and automatic assembly of evolving component-based systems“. In: *Journal of Systems and Software* 81.12 (2008), S. 2237–2251 (Zitiert auf Seite 86).

[Pup] Puppet. *Powerful infrastructure automation and delivery: Puppet*. URL: https://puppet.com (Zitiert auf Seite 58).

[PY10] J. T. Piao, J. Yan. „A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing“. In: *Proceedings of the Ninth International Conference on Grid and Cloud Computing*. IEEE. 2010, S. 87–92 (Zitiert auf Seite 45).

[RBL+09] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, F. Galan. „The RESERVOIR Model and Architecture for Open Federated Cloud Computing“. In: *IBM Journal of Research and Development* 53.4 (2009), 4:1–4:11 (Zitiert auf Seite 56).

[RCBW16] M. H. ur Rehman, V. Chang, A. Batool, T. Y. Wah. „Big data reduction framework for value creation in sustainable enterprises“. In: *International Journal of Information Management* 36.6 (2016), S. 917–928 (Zitiert auf Seite 85).

[RDR10] S. Rizou, F. Dürr, K. Rothermel. „Solving the Multi-Operator Placement Problem in Large-Scale Operator Networks“. In: *Proceedings of the 19th International Conference on Computer Communications and Networks*. IEEE. 2010, S. 1–6 (Zitiert auf Seite 46).

[REC15] K. Rose, S. Eldridge, L. Chapin. „The Internet of Things: An Overview“. In: *The Internet Society (ISOC 2015)* (Okt. 2015), S. 1–50 (Zitiert auf Seite 32).

[RFG+18] S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, F. Herrera. „Big Data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce“. In: *Information Fusion* 42 (2018), S. 51–61 (Zitiert auf Seite 85).

[RH] Red Hat, Inc. *Ansible is Simple IT Automation*. URL: https://www.ansible.com (Zitiert auf Seite 58).

[RHH11] Z. U. Rehman, F. K. Hussain, O. K. Hussain. „Towards Multi-criteria Cloud Service Selection“. In: *Proceedings of the Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2011)*. 2011, S. 44–48 (Zitiert auf Seite 149).

[Rob16] M. Roberts. *Serverless Architectures*. Aug. 2016. URL: http://martinfowler.com/articles/serverless.html (Zitiert auf Seite 44).

[RR00] M. Rodríguez-Martínez, N. Roussopoulos. „MOCHA: A Self-extensible Database Middleware System for Distributed Data Sources“. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. ACM, 2000, S. 213–224 (Zitiert auf Seite 42).

[RT17] B. L. Risteska Stojkoska, K. V. Trivodaliev. „A review of Internet of Things for smart home: Challenges and solutions“. In: *Journal of Cleaner Production* 140 (2017), S. 1454–1464 (Zitiert auf Seite 34).

[Rus11] P. Russom. „Big Data Analytics“. In: *TDWI Best Practices Report* 19.4 (Sep. 2011), S. 1–34 (Zitiert auf den Seiten 11, 34, 39).

[SAH+13] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, C.-H. Lung. „Smart Home: Integrating Internet of Things with Web Services and Cloud Computing“. In: *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2013)*. Bd. 2. 2013, S. 317–320 (Zitiert auf Seite 34).

[Sat17] M. Satyanarayanan. „The Emergence of Edge Computing“. In: *Computer* 50.1 (2017), S. 30–39 (Zitiert auf den Seiten 37, 40).

[SBK+18] K. Saatkamp, U. Breitenbücher, K. Képes, F. Leymann, M. Zimmermann. „OpenTOSCA Injector: Vertical and Horizontal Topology Model Injection“. In: *Proceedings of the Service-Oriented Computing – ICSOC 2017 Workshops*. Springer, Jan. 2018, S. 379–383 (Zitiert auf den Seiten 24, 149, 178).

[SBKL17] K. Saatkamp, U. Breitenbücher, O. Kopp, F. Leymann. „Topology Splitting and Matching for Multi-Cloud Deployments“. In: *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*. SciTePress, 2017, S. 247–258 (Zitiert auf den Seiten 141, 149, 158).

[SBKL18] K. Saatkamp, U. Breitenbücher, O. Kopp, F. Leymann. „Application Scenarios for Automated Problem Detection in TOSCA Topologies by Formalized Patterns“. In: *Papers From the 12thAdvanced Summer School on Service-Oriented Computing (SummerSOC 2018)*. IBM Research Division, 2018, S. 43–53 (Zitiert auf Seite 86).

[SBKL19] K. Saatkamp, U. Breitenbücher, O. Kopp, F. Leymann. „Method, formalization, and algorithms to split topology models for distributed cloud application deployments“. In: *Computing* 102 (2019), S. 343–363 (Zitiert auf Seite 158).

[SBLW17] K. Saatkamp, U. Breitenbücher, F. Leymann, M. Wurster. „Generic Driver Injection for Automated IoT Application Deployments“. In: *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2017, S. 320–329 (Zitiert auf Seite 177).

[SC13] J. J. Seddon, W. L. Currie. „Cloud computing and trans-border health data: Unpacking U.S. and EU healthcare regulation and compliance“. In: *Health Policy and Technology* 2.4 (2013), S. 229–241 (Zitiert auf Seite 186).

[Sch18] H. R. Schmidtke. „Is the internet spatial?“ In: *Journal of Reliable Intelligent Environments* 4 (2018), S. 123–129 (Zitiert auf den Seiten 134, 135, 147).

[SCY19] Y. Sahni, J. Cao, L. Yang. „Data-Aware Task Allocation for Achieving Low Latency in Collaborative Edge Computing“. In: *IEEE Internet of Things Journal* 6.2 (2019), S. 3512–3524 (Zitiert auf Seite 46).

[SCZ+16] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu. „Edge Computing: Vision and Challenges“. In: *IEEE Internet of Things Journal* 3.5 (2016), S. 637–646 (Zitiert auf Seite 37).

[SD16] W. Shi, S. Dustdar. „The Promise of Edge Computing“. In: *Computer* 49.5 (2016), S. 78–81 (Zitiert auf Seite 37).

[SDAV16] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, V. Vlassov. „SpanEdge: Towards Unifying Stream Processing over Central and Near-the-Edge Data Centers“. In: *Proceedings of the IEEE/ACM Symposium on Edge Computing (SEC 2016)*. IEEE. 2016, S. 168–178 (Zitiert auf Seite 46).

[SDH+11] B. W. Settlemyer, J. D. Dobson, S. W. Hodson, J. A. Kuehn, S. W. Poole, T. M. Ruwart. „A Technique for Moving Large Data Sets over High-Performance Long Distance Networks“. In: *Proceedings of the 27th IEEE Symposium on Mass Storage Systems and Technologies (MSST 2011)*. IEEE, 2011, S. 1–6 (Zitiert auf den Seiten 134, 147).

[SDL20] F. A. Salaht, F. Desprez, A. Lebre. „An Overview of Service Placement Problem in Fog and Edge Computing“. In: *ACM Computing Surveys (CSUR)* 53.3 (2020) (Zitiert auf den Seiten 12, 45).

[Sel03] B. Selic. „The Pragmatics of Model-Driven Development“. In: *IEEE Software* 20.5 (2003), S. 19–25 (Zitiert auf Seite 165).

[SGG+13] D. Spath, O. Ganschar, S. Gerlach, M. Hämmerle, T. Krause, S. Schlund. *Produktionsarbeit der Zukunft - Industrie 4.0*. Bd. 150. Fraunhofer Verlag Stuttgart, 2013 (Zitiert auf den Seiten 11, 40).

[SKK03] J. Strauss, D. Katabi, F. Kaashoek. „A Measurement Study of Available Bandwidth Estimation Tools“. In: *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. ACM, 2003, S. 39–44 (Zitiert auf Seite 147).

[SKRC10] K. Shvachko, H. Kuang, S. Radia, R. Chansler. „The Hadoop Distributed File System“. In: *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST 2010)*. IEEE Computer Society, 2010, S. 1–10 (Zitiert auf Seite 43).

[SM15] F. Shrouf, G. Miragliotta. „Energy management based on Internet of Things: Practices and framework for adoption in production management“. In: *Journal of Cleaner Production* (2015) (Zitiert auf den Seiten 11, 28, 40).

[SM22] S. Smolka, Z. Á. Mann. „Evaluation of fog application placement algorithms: A survey“. In: *Computing* 104 (2022), S. 1397–1423 (Zitiert auf Seite 45).

[SMN17] P. Smirnov, M. Melnik, D. Nasonov. „Performance-aware scheduling of streaming applications using genetic algorithm“. In: *Procedia Computer Science* 108 (2017), S. 2240–2249 (Zitiert auf Seite 46).

[SR20] M. Shirer, J. Rydning. *IDC's Global DataSphere Forecast Shows Continued Steady Growth in the Creation and Consumption of Data*. 2020. URL: https://www.idc.com/getdoc.jsp?containerId=prUS46286020 (Zitiert auf den Seiten 11, 39).

[SRC14] G. C. Silva, L. M. Rose, R. Calinescu. „Cloud DSL: A Language for Supporting Cloud Portability by Describing Cloud Entities“. In: *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE 2014)*. Sep. 2014, S. 36–45 (Zitiert auf Seite 56).

[SSSS11] U. Sharma, P. Shenoy, S. Sahu, A. Shaikh. „Kingfisher: Cost-aware Elasticity in the Cloud“. In: *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM 2011)*. IEEE, 2011, S. 206–210 (Zitiert auf Seite 48).

[SSX+15] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, B. Amos. „Edge Analytics in the Internet of Things“. In: *IEEE Pervasive Computing* 14.2 (2015), S. 24–31 (Zitiert auf den Seiten 37, 44).

[STV18] J. Soldani, D. A. Tamburri, W.-J. Van Den Heuvel. „The Pains and Gains of Microservices: A Systematic Grey Literature Review“. In: *Journal of Systems and Software* 146 (2018), S. 215–232 (Zitiert auf Seite 39).

[SVC06] T. Stahl, M. Voelter, K. Czarnecki. *Model-driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, Juli 2006 (Zitiert auf Seite 165).

[SWW15] A.-R. Sadeghi, C. Wachsmann, M. Waidner. „Security and Privacy Challenges in Industrial Internet of Things“. In: *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC 2015)*. 2015, S. 1–6 (Zitiert auf den Seiten 35, 111).

[Sza11] A. Szalay. „Extreme Data-Intensive Scientific Computing“. In: *Computing in Science & Engineering* 13.6 (2011), S. 34–41 (Zitiert auf den Seiten 11, 40).

[TAG22] F. Tavousi, S. Azizi, A. Ghaderzadeh. „A fuzzy approach for optimal placement of IoT applications in fog-cloud computing“. In: *Cluster Computing* 25.1 (2022), S. 303–320 (Zitiert auf Seite 40).

[TASFa] The Apache Software Foundation. *Apache Storm*. URL: http://storm.apache.org/ (Zitiert auf den Seiten 43, 46).

[TASFb] The Apache Software Foundation. *Welcome to Apache Hadoop!* URL: http://hadoop.apache.org/ (Zitiert auf Seite 43).

[TFK+18] A. Tsuchiya, F. Fraile, I. Koshijima, A. Ortiz, R. Poler. „Software defined networking firewall for industry 4.0 manufacturing systems“. In: *Journal of Industrial Engineering and Management (JIEM 2018)* 11.2 (2018), S. 318–333 (Zitiert auf Seite 74).

[TSZS06] K. Tan, J. Song, Q. Zhang, M. Sridharan. „A Compound TCP Approach for High-Speed and Long Distance Networks“. In: *Proceedings of the 25th IEEE International Conference on Computer Communications*. IEEE, 2006, S. 1–12 (Zitiert auf den Seiten 134, 147).

[VCF18] M. Villari, A. Celesti, M. Fazio. „Towards Osmotic Computing: Looking at Basic Principles and Technologies“. In: *Proceedings of the Conference on Complex, Intelligent, and Software Intensive Systems*. Springer, 2018, S. 906–915 (Zitiert auf den Seiten 38, 45).

[VFD+16] M. Villari, M. Fazio, S. Dustdar, O. Rana, R. Ranjan. „Osmotic Computing: A New Paradigm for Edge/Cloud Integration“. In: *IEEE Cloud Computing* 3.6 (2016), S. 76–83 (Zitiert auf den Seiten 38, 39).

[VGC+15] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, S. Gil. „Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud“. In: *Proceedings of the 10th Computing Colombian Conference (CCC 2015)*. IEEE, 2015, S. 583–590 (Zitiert auf Seite 160).

[VMW] VMware, Inc. *Support for SaltStack*. URL: https://www.saltstack.com (Zitiert auf Seite 58).

[VOE11] R. L. Villars, C. W. Olofson, M. Eastwood. „Big data: What It Is and Why You Should Care“. In: *White Paper, IDC* 14 (2011), S. 1–14 (Zitiert auf den Seiten 39, 185).

[Vos16] W. G. Voss. „European Union Data Privacy Law Reform: General Data Protection Regulation, Privacy Shield, and the Right to Delisting“. In: *The Business Lawyer* 72.1 (2016), S. 221–234 (Zitiert auf Seite 73).

[VÖU04] K. Voruganti, M. T. Özsu, R. C. Unrau. „An Adaptive Data-Shipping Architecture for Client Caching Data Management Systems“. In: *Distributed and Parallel Databases* 15.2 (2004), S. 137–177 (Zitiert auf Seite 12).

[VR14] L. M. Vaquero, L. Rodero-Merino. „Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing“. In: *ACM SIGCOMM Computer Communication Review* 44.5 (Okt. 2014), S. 27–32 (Zitiert auf Seite 36).

[VRCL08] L. M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner. „A Break in the Clouds: Towards a Cloud Definition“. In: *ACM SIGCOMM Computer Communication Review* 39.1 (Dez. 2008), S. 50–55 (Zitiert auf Seite 28).

[VTM09] H. N. Van, F. D. Tran, J.-M. Menaud. „Autonomic virtual resource management for service hosting platforms“. In: *Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE. 2009, S. 1–8 (Zitiert auf Seite 45).

[VWB+16] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, D. S. Nikolopoulos. „Challenges and Opportunities in Edge Computing“. In: *Proceedings of the IEEE International Conference on Smart Cloud (SmartCloud 2016)*. 2016, S. 20–26 (Zitiert auf Seite 37).

[WBB+14a] J. Wettinger, T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. „Streamlining Cloud Management Automation by Unifying the Invocation of Scripts and Services Based on TOSCA“. In: *International Journal of Organizational and Collective Intelligence (IJOCI 2014)* 4.2 (Apr. 2014), S. 45–63 (Zitiert auf den Seiten 161, 216).

[WBB+14b] J. Wettinger, T. Binz, U. Breitenbücher, O. Kopp, F. Leymann, M. Zimmermann. „Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA“. In: *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. SciTePress, Apr. 2014, S. 559–568 (Zitiert auf den Seiten 23, 161, 216).

[WBB+19] M. Wurster, U. Breitenbücher, A. Brogi, G. Falazi, L. Harzenetter, F. Leymann, J. Soldani, V. Yussupov. „The EDMM Modeling and Transformation System“. In: *Proceedings of the Service-Oriented Computing – ICSOC 2019 Workshops*. 2019, S. 294–298 (Zitiert auf Seite 58).

[WBF+19] M. Wurster, U. Breitenbücher, M. Falkenthal, C. Krieger, F. Leymann, K. Saatkamp, J. Soldani. „The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies“. In: *Software-Intensive Cyber-Physical Systems (SICS 2019)* (2019) (Zitiert auf den Seiten 13, 57, 58, 92, 129, 170).

[WBH+20] K. Wild, U. Breitenbücher, L. Harzenetter, F. Leymann, D. Vietz, M. Zimmermann. „TOSCA4QC: Two Modeling Styles for TOSCA to Automate the Deployment and Orchestration of Quantum Applications“. In: *Proceedings of the 24th International Enterprise Distributed Object Computing Conference (EDOC 2020)*. IEEE, 2020, S. 125–134 (Zitiert auf den Seiten 25, 219).

[WBK+20] B. Weder, U. Breitenbücher, K. Képes, F. Leymann, M. Zimmermann. „Deployable Self-Contained Workflow Models“. In: *Proceedings of the 8th European Conference on Service-Oriented and Cloud Computing (ESOCC 2020)*. Springer International Publishing, März 2020, S. 85–96 (Zitiert auf Seite 25).

[WBKL16] J. Wettinger, U. Breitenbücher, O. Kopp, F. Leymann. „Streamlining DevOps Automation for Cloud Applications Using TOSCA as Standardized Metamodel“. In: *Future Generation Computer Systems* 56 (2016), S. 317–332 (Zitiert auf den Seiten 75, 177).

[WBKL18] M. Wurster, U. Breitenbücher, O. Kopp, F. Leymann. „Modeling and Automated Execution of Application Deployment Tests“. In: *Proceedings of the 22nd IEEE International Enterprise Distributed Object Computing Conference (EDOC 2018)*. IEEE Computer Society, Okt. 2018, S. 171–180 (Zitiert auf Seite 86).

[WBL14a] J. Wettinger, U. Breitenbücher, F. Leymann. „Compensation-based vs. Convergent Deployment Automation for Services Operated in the Cloud“. In: *Proceedings of the 12th International Conference on Service-Oriented Computing (ICSOC 2014)*. Springer, Nov. 2014, S. 336–350 (Zitiert auf den Seiten 57, 73, 122).

[WBL14b] J. Wettinger, U. Breitenbücher, F. Leymann. „Standards-based DevOps Automation and Integration Using TOSCA“. In: *Proceedings of the 7th International Conference on Utility and Cloud Computing (UCC 2014)*. IEEE, Dez. 2014, S. 59–68 (Zitiert auf Seite 178).

[WBLZ21] B. Weder, J. Barzen, F. Leymann, M. Zimmermann. „Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective“. In: *Proceedings of the 18th IEEE International Conference on Web Services (ICWS 2021)*. IEEE, 2021, S. 1–13 (Zitiert auf den Seiten 25, 218).

[WCO03] Y. Wu, M.-H. Chen, J. Offutt. „UML-based Integration Testing for Component-based Software“. In: *Proceedings of the International Conference on COTS-Based Software Systems*. Springer, 2003, S. 251–260 (Zitiert auf Seite 86).

[WGR17] A. Wegner, J. Graham, E. Ribble. „A New Approach to Cyberphysical Security in Industry 4.0“. In: *Cybersecurity for Industry 4.0*. Springer, 2017, S. 59–72 (Zitiert auf Seite 70).

[WL91] M. E. Wolf, M. S. Lam. „A Data Locality Optimizing Algorithm“. In: *ACM Sigplan Notices*. Bd. 26. 6. ACM. 1991, S. 30–44 (Zitiert auf Seite 41).

[WTTL12] G. Wu, M. Tang, Y.-C. Tian, W. Li. „Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm“. In: *Neural Information Processing*. Springer, 2012, S. 315–323 (Zitiert auf Seite 45).

[WVY+10] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, C. Fu. „Cloud Computing: a Perspective Study“. In: *New generation computing* 28.2 (2010), S. 137–146 (Zitiert auf Seite 28).

[WWB+13] T. Waizenegger, M. Wieland, T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, B. Mitschang, A. Nowak, S. Wagner. „Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing“. In: *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*. Springer, Sep. 2013, S. 360–376 (Zitiert auf Seite 202).

[XCTS14] J. Xu, Z. Chen, J. Tang, S. Su. „T-Storm: Traffic-Aware Online Scheduling in Storm“. In: *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS 2014)*. IEEE. 2014, S. 535–544 (Zitiert auf Seite 47).

[XYWV12] F. Xia, L. T. Yang, L. Wang, A. Vinel. „Internet of Things“. In: *International Journal of Communication Systems* 25.9 (Sep. 2012), S. 1101–1102 (Zitiert auf Seite 32).

[YFK+18] V. Yussupov, M. Falkenthal, O. Kopp, F. Leymann, M. Zimmermann. „Secure Collaborative Development of Cloud Application Deployment Models“. In: *Proceedings of The Twelfth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2018)*. Hrsg. von G. Yee, S. Rass, S. Schauer, M. Latzenhofer. Xpert Publishing Services, Sep. 2018, S. 48–57 (Zitiert auf den Seiten 24, 203).

[YQL15] S. Yi, Z. Qin, Q. Li. „Security and Privacy Issues of Fog Computing: A Survey“. In: *Wireless Algorithms, Systems, and Applications*. Springer, 2015, S. 685–695 (Zitiert auf Seite 37).

[ZA08] Q. Zhu, G. Agrawal. „Resource Allocation for Distributed Streaming Applications“. In: *Proceedings of the 37th International Conference on Parallel Processing*. IEEE. 2008, S. 414–421 (Zitiert auf den Seiten 46, 48).

[ZBF+17a] M. Zimmermann, F. W. Baumann, M. Falkenthal, F. Leymann, U. Odefey. „Automating the Provisioning and Integration of Analytics Tools with Data Resources in Industrial Environments using OpenTOSCA“. In: *Proceedings of the 21st IEEE International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW 2017)*. IEEE, Okt. 2017, S. 3–7 (Zitiert auf den Seiten 22, 70, 135, 185).

[ZBF+17b] M. Zimmermann, U. Breitenbücher, M. Falkenthal, F. Leymann, K. Saatkamp. „Standards-based Function Shipping - How to use TOSCA for Shipping and Executing Data Analytics Software in Remote Manufacturing Environments“. In: *Proceedings of the 21st IEEE International Enterprise Distributed Object Computing Conference (EDOC 2017)*. IEEE, Okt. 2017, S. 50–60 (Zitiert auf den Seiten 12, 21, 40, 50, 70, 72, 111, 178, 185, 186).

[ZBG+18] M. Zimmermann, U. Breitenbücher, J. Guth, S. Hermann, F. Leymann, K. Saatkamp. „Towards Deployable Research Object Archives Based on TOSCA“. In: *Papers from the 12th Advanced Summer School on Service-Oriented Computing (SummerSoC 2018)*. IBM Research Division, Okt. 2018, S. 31–42 (Zitiert auf den Seiten 22, 123).

[ZBH+20] M. Zimmermann, U. Breitenbücher, L. Harzenetter, F. Leymann, V. Yussupov. „Self-Contained Service Deployment Packages“. In: *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*. SciTePress, Mai 2020, S. 371–381 (Zitiert auf den Seiten 22, 74, 122, 212).

[ZBK+20] M. Zimmermann, U. Breitenbücher, K. Képes, F. Leymann, B. Weder. „Data Flow Dependent Component Placement of Data Processing Cloud Applications“. In: *Proceedings of the 2020 IEEE International*

Conference on Cloud Engineering (IC2E 2020). IEEE Computer Society, Apr. 2020, S. 83–94 (Zitiert auf den Seiten 23, 36, 66, 71, 134).

[ZBKL18] M. Zimmermann, U. Breitenbücher, C. Krieger, F. Leymann. „Deployment Enforcement Rules for TOSCA-based Applications“. In: *Proceedings of The Twelfth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2018)*. Xpert Publishing Services, Sep. 2018, S. 114–121 (Zitiert auf den Seiten 22, 35, 40, 70, 140, 186).

[ZBL17] M. Zimmermann, U. Breitenbücher, F. Leymann. „A TOSCA-based Programming Model for Interacting Components of Automatically Deployed Cloud and IoT Applications“. In: *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017)*. SciTePress, Apr. 2017, S. 121–131 (Zitiert auf den Seiten 22, 69, 76, 85, 161, 165).

[ZBL18] M. Zimmermann, U. Breitenbücher, F. Leymann. „A Method and Programming Model for Developing Interacting Cloud Applications Based on the TOSCA Standard“. In: *Enterprise Information Systems*. Springer, Juni 2018, S. 265–290 (Zitiert auf den Seiten 22, 50, 69, 76, 85, 161, 165).

[ZCZ+18] J. Zhang, B. Chen, Y. Zhao, X. Cheng, F. Hu. „Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues“. In: *IEEE Access* 6 (2018), S. 18209–18237 (Zitiert auf Seite 12).

[Zha18] D. Zhang. „Big Data Security and Privacy Protection“. In: *Proceedings of the 8th International Conference on Management and Computer Science (ICMCS 2018)*. Bd. 77. 2018, S. 275–278 (Zitiert auf Seite 70).

[Zim17] O. Zimmermann. „Microservices tenets“. In: *Computer Science - Research and Development* 32.3 (2017), S. 301–310 (Zitiert auf Seite 39).

[ZLL+16] Q. Zheng, R. Li, X. Li, N. Shah, J. Zhang, F. Tian, K.-M. Chao, J. Li. „Virtual machine consolidated placement based on multi-objective biogeography-based optimization“. In: *Future Generation Computer Systems* 54 (2016), S. 95–122 (Zitiert auf Seite 45).

[ZMK+15] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, J. Kubiatowicz. „The Cloud is Not Enough: Saving IoT from the Cloud“. In: *Proceedings of the 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2015)*. USENIX Association, Juli 2015 (Zitiert auf Seite 35).

[ZR13] C. Zhang, J. E. Ramirez-Marquez. „Protecting critical infrastructures against intentional attacks: A two-stage game with incomplete information“. In: *IIE Transactions* 45.3 (2013), S. 244–258 (Zitiert auf den Seiten 70, 111).

[ZWC+17] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, R. Buyya. „Cloud Service Reliability Enhancement via Virtual Machine Placement Optimization“. In: *IEEE Transactions on Services Computing* 10.6 (2017), S. 902–913 (Zitiert auf Seite 45).

[ZZZB12] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba. „Dynamic Service Placement in Geographically Distributed Clouds“. In: *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems (ICDCS 2012)*. 2012, S. 526–535 (Zitiert auf Seite 48).

Alle URLs wurden zuletzt am 13.03.2023 geprüft.

Abbildungsverzeichnis

Definitionsverzeichnis

Algorithmenverzeichnis

Listingsverzeichnis