Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3615

# Konzepte zur Vorbereitung des Softwarebetriebs aus Entwicklersicht

Sebastian Kiesewetter

| | |
|---|---|
| **Studiengang:** | Informatik |
| **Prüfer/in:** | Prof. Dr. Frank Leymann |
| **Betreuer/in:** | Dipl.-Inf. Johannes Wettinger |
| **Beginn am:** | 1. April 2014 |
| **Beendet am:** | 1. Oktober 2014 |
| **CR-Nummer:** | D.2.1, D.2.9 |

## Kurzfassung

Die Paradigmen Cloud Computing und Infrastructure as Code eröffnen neue Möglichkeiten für die Entwicklung von Software sowie für das IT-Service Management, die gegenüber den traditionellen Vorgehensweisen einige Vorteile mit sich bringen. Insbesondere durch fortschreitende Automatisierung von Prozessen wird Zeitersparnis und somit Flexibilität und Reduzierung der Kosten erreicht. Darüber hinaus wird die Fehleranfälligkeit von zuvor manuell durchgeführten Managementaufgaben verringert. Vor allem die enge Verzahnung der Unternehmensbereiche Softwareentwicklung und Softwarebetrieb spielt für die erfolgreiche Umsetzung der genannten Vorteile eine große Rolle. Bestrebungen in diese Richtung verfolgt das Paradigma DevOps, indem Methoden der agilen Softwareentwicklung auf die Bereitstellung von virtualisierter IT-Infrastruktur, Middleware und Applikationen angewendet werden.

Es existiert eine Reihe von zumeist proprietären Werkzeugen, mit deren Hilfe die Verwaltung von bereitgestellten Diensten in der Cloud standardisiert und automatisiert werden kann. Diese Werkzeuge haben ihre Wurzeln in der Domäne des Softwarebetriebs und sind daher für Softwareentwickler nicht ohne ein gewisses Maß an spezifischem Wissen bedienbar.

Die vorliegende Arbeit adressiert dieses Problem und integriert Ansätze für die Automatisierung von Management und Bereitstellung von IT-Infrastruktur mit Ausdrucksmöglichkeiten aus der Domäne der Softwareentwickler. Ziel ist es, Softwareentwickler hiermit bei der Erstellung von Spezifikationen zu unterstützen, die als Eingaben für Infrastructure as Code-Werkzeuge erstellt werden müssen.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Verzeichnis der Listings

# 1. Einleitung

Der Begriff *Cloud Computing* bezeichnet ein Paradigma für die Bereitstellung von IT-Ressourcen. Hervorgegangen aus *Cluster Computing* und *Grid Computing* zeichnet sich Cloud Computing vor allem durch hohe Elastizität sowie einem verbrauchsbasierten Abrechnungsmodell aus. Hierbei wird insbesondere auf die Technologien Virtualisierung und *Web Services* zurückgegriffen. [Ley09]

Eine Vielzahl von Anbietern offeriert eine breite Auswahl an Diensten, die entsprechend der tatsächlich stattgefundenen Nutzung in Rechnung gestellt werden. Diese Dienste können sich auf virtualisierte Komponenten einer Infrastruktur beschränken, wie beispielsweise auf Datenbanken oder Netzwerkkomponenten (Google Compute Engine [com], Amazon Web Services [awsa]) *(Infrastructure as a Service)* oder sie umfassen virtualisierte Umgebungen aus Hardware und Software (Google App Engine [dev], AWS Elastic Beanstalk [awsb]) *(Platform as a Service)*. Darüber hinaus werden Dienste angeboten, welche vollständige Softwarelösungen darstellen und somit den höchsten Abstraktionsgrad bieten (Google Cloud DNS [cloa]) *(Software as a Service)*. Die Bereitstellung von Infrastruktur- und Plattform-Diensten ist durch lizenzfrei verfügbare, quelloffene Software wie z.B. OpenStack [opea] möglich, so dass nicht unbedingt auf externe Anbieter zurückgegriffen werden muss.

Die *agile Softwareentwicklung* [TS10], mit deren Hilfe Änderungen und neue Anforderungen an Softwareprojekte flexibel und schnell umgesetzt werden können, ermöglicht die Erfüllung von Kundenwünschen innerhalb einer deutlich kürzeren Zeit als bei einer traditionellen Vorgehensweise. Darüber hinaus bietet sie ein *Trackingsystem* für den Fortschritt eines Projektes und erleichtert dessen paralleles Vorantreiben. [PW11]

Als Erweiterung der agilen Softwareentwicklung beginnt man derzeit damit, die Entwicklung und den Betrieb von Software zu integrieren. Dieses Paradigma ist unter dem Namen *DevOps* (Development and Operations) bekannt und ist in der Praxis bereits seit einigen Jahren verbreitet [JA09]. Der Fokus von DevOps liegt auf der Automatisierung von wiederkehrenden Konfigurations- und Deploymentaufgaben *(Infrastructure as Code)* sowie auf der Integration der Unternehmensbereiche Softwareentwicklung und Softwarebetrieb. Mit Werkzeugen wie Chef [get] , Juju [jujc] oder Puppet [pupb] werden konfigurierbare Infrastruktur und virtualisierte Middleware-Services (z.B. *Database-as-a-Service* [LS10]) automatisiert bereitgestellt. Insbesondere trifft dies auch auf virtualisierte Infrastruktur im Cloud Computing zu.

## 1.1. Problembeschreibung

Der Ansatz aus der DevOps-Community, die Trennung der Organisationseinheiten Softwareentwicklung und Softwarebetrieb durch die Einrichtung von interdisziplinären Teams mit Vertretern beider

Seiten aufzubrechen und somit eine dauerhafte Zusammenarbeit über den gesamten Produktlebenszyklus der Software zu etablieren, ermöglicht es, von vielen Vorteilen der agilen Softwareentwicklung sowie des Cloud Computings zu profitieren [Deb11]. Der Betrieb von Software wird erleichtert und der *Releasezyklus* sowie der Integrationsprozess werden beschleunigt. Softwareentwickler, die hinreichende Kenntnisse aus dem Bereich Softwarebetrieb mitbringen, profitieren von der Verwendung neuer Werkzeuge für die Automatisierung wiederkehrender Prozesse. Beispielsweise kann ohne großen Aufwand eine Reihe gleich konfigurierter Entwicklungssysteme erstellt werden, die jeweils unterschiedliche Versionen einer Laufzeitumgebung enthalten und so zum Testen und Debuggen der zu entwickelnden Anwendung hinsichtlich Kompatibilität mit der jeweiligen Version verwendet werden können ([vag], [doc]).

Trotz vieler Vorteile, die sowohl Softwareentwicklung als auch Softwarebetrieb aus den Ansätzen des DevOps-Paradigmas und der Nutzung zugehöriger Werkzeuge ziehen können, besteht nach wie vor eine Lücke zwischen den beiden Seiten. So stellen die Implementierung der automatisierten Konfigurations- und Deploymentaufgaben und die Implementierung der Anwendungslogik zwei völlig unterschiedliche Domänen dar, was dem Ziel der Integration von Entwicklung und Betrieb widerspricht. [PCS+13], [Sac12]

Um beiden Seiten, Entwicklung und Betrieb, eine gemeinsame Sicht auf den Gesamtprozess der Erstellung und Bereitstellung von Software zu ermöglichen, wird in dieser Arbeit ein Lösungsansatz vorgestellt, der eine effizientere Zusammenarbeit durch Automatisierung ermöglicht. Ziel ist die Unterstützung von Entwicklern mit Mechanismen, die das Bereitstellen von Anwendungen unter Berücksichtigung von Anforderungen an die Umgebung sowie von Einschränkungen für die Anwendung durchführen. Schematisch ist dies in Abbildung 1.1 dargestellt.

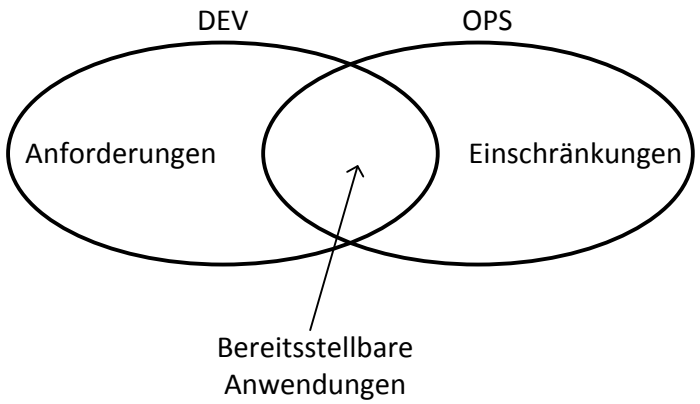


**Abbildung 1.1.:** Anforderungen und Einschränkungen

## 1.2. Einführendes Beispiel

Ein beispielhaftes Szenario stellt eine Webanwendung dar, die auf PHP basiert und einen Webserver sowie einen Datenbankserver benötigt. Als Benutzerschnittstelle dienen HTML-basierte Webseiten, die im Internetbrowser des Anwenders dargestellt werden. Die Kommunikation zwischen Anwender und Webserver erfolgt über HTTP(S). Als beispielhafte Anwendung soll im Folgenden das *Issue Tracking System Mantis* [man] dienen, welches unter der GPL2-Lizenz [gnu] frei verfügbar ist.

Für die Bereitstellung dieser konkreten Anwendung werden aus Entwicklersicht einige wenige grundlegende Anforderungen an die Umgebung gestellt, in welcher diese ausgeführt werden soll. So muss z.B. ein Webserver mit PHP-Unterstützung zur Verfügung stehen, der die Anwendung ausführt. Für die Datenhaltung muss eine Datenbank vorhanden sein. Damit die Anwendung erreichbar ist, muss Port 80 (HTTP) nach außen verfügbar gemacht werden. Die grundlegenden Anforderungen können optional noch genauer spezifiziert werden. Beispielsweise kann die Notwendigkeit bestehen, dass die Anforderung *PHP-Unterstützung* mit einem Attribut *Version* versehen werden kann.

Aus Sicht des Softwarebetriebs wiederum werden verschiedene Einschränkungen für die Bereitstellung der Anwendung gemacht. Hier ist denkbar, dass bereits ein MySQL [mys]-Server betrieben wird und daher die Anforderung *Zugriff auf Datenbankserver* eingeschränkt wird auf *Zugriff auf MySQL-Server*. Aber auch neue Einschränkungen, die über die vorliegenden Anforderungen hinausgehen, müssen gemacht werden können. Als Beispiel ist hier zu nennen, dass der Zugriff auf den Webserver nur innerhalb eines abgegrenzten Netzbereichs (z.B. Intranet einer Firma) erfolgen kann.

## 1.3. Übersicht

Im Anschluss an das in die Thematik und Problemstellung einleitende Kapitel 1 folgen in Kapitel 2 Informationen zum aktuellen Stand der Technik, wobei die Bereiche *Cloud Computing*, *Infrastructure as Code*, *DevOps* und *NoOps* abgedeckt werden. In Kapitel 3 werden die erarbeiteten, abstrakten Konzepte im Detail erläutert. Kapitel 4 beschreibt eine prototypische Implementierung der Lösung, die im folgenden Kapitel 5 anhand von drei konkreten Anwendungsfällen evaluiert wird. In Kapitel 6 wird abschließend ein Fazit der vorgestellten Arbeit gezogen.

# 2. Stand der Technik

Dieses Kapitel gibt einen Überblick über den aktuellen Stand der Technik in den Bereichen, die für die vorliegende Arbeit von Bedeutung sind. In Abschnitt 2.1 wird der Begriff Cloud Computing erläutert und es wird ein Überblick über die aktuelle Marktsituation gegeben. In Abschnitt 2.2 werden das Konzept *Infrastructure as Code* sowie eine Auswahl von Werkzeugen für die Automatisierung von Betriebsaufgaben vorgestellt. In Abschnitt 2.3 wird das DevOps-Paradigma als konsequente Fortführung der agilen Softwareentwicklung beschrieben und in Abschnitt 2.4 werden mit NoOps neueste Entwicklungen in diesem Bereich vorgestellt. In Abschnitt 2.5 werden schließlich einige verwandte Arbeiten sowie deren Verbindungen zur vorliegenden Arbeit genannt.

## 2.1. Cloud Computing

Die *IT* im heutigen unternehmerischen Umfeld wird vor vielfältige Herausforderungen gestellt. Reaktionsfähigkeit auf sich schnell verändernde und neue Anforderungen sowie die Notwendigkeit zur Kostenminimierung haben eine hohe Bedeutung für den Erhalt der Konkurrenzfähigkeit. Als Konsequenz daraus ist insbesondere bei Entscheidungsträgern ein großes Interesse am Thema Cloud Computing zu erkennen. Verringerung der Bereitstellungszeiten für Server und Anwendungen von Wochen auf Minuten bei gleichzeitiger Verbesserung der Zuverlässigkeit sind starke Argumente für eine Hinwendung zum Cloud Computing. Darüber hinaus lassen sich Aufwände für Konfiguration, Betrieb, Management und Monitoring signifikant reduzieren. [Cor11]

### 2.1.1. Allgemeines

Nach [MG11] beschreibt der Begriff Cloud Computing ein Modell für die Bereitstellung von Kapazitäten in einem Netzwerk, die mit wenig Aufwand schnell bereitgestellt bzw. wieder freigegeben werden können und dabei jederzeit sowie einfach zugreifbar sind und unlimitiert zur Verfügung stehen. Hierfür benötigte Ressourcen werden durch *Pooling* heterogener Systeme zur Verfügung und entsprechend der tatsächlich stattgefundenen Nutzung in Rechnung gestellt. Hierbei werden drei verschiedene Konzepte der Bereitstellung von Kapazitäten als Dienste angegeben, wobei diese jeweils aufeinander aufbauen. Auf unterster Ebene werden grundlegende, virtualisierte Infrastrukturkomponenten wie z.B. Speicherplatz, Netzwerke oder Rechenleistung angeboten (*Infrastructure as a Service* (IaaS)). Auf der nächsthöheren Ebene werden vollständige und konfigurierbare Laufzeitumgebungen für Anwendungen angeboten (*Platform as a Service* (PaaS)). Auf der obersten Ebene werden schließlich vollständige Anwendungen angeboten (*Software as a Service* (SaaS)), wobei der Betrieb und die Verwaltung von Komponenten aus darunterliegenden Schichten hierbei dem Dienstanbieter obliegt. Organisatorisch lässt sich das Cloud Computing zusätzlich kategorisieren in *Private Clouds*

(exklusive Zuordnung zu einer einzelnen Organisation), *Community Clouds* (exklusive Zuordnung zu einer bestimmten Gruppe), *Public Clouds* (offen für alle) und *Hybrid Clouds* (einer Zusammensetzung von zwei oder mehr der vorangegangenen Kategorien).

Wie in [AFG+09] dargelegt wird, ergeben sich durch das Cloud Computing für den Benutzer eine Reihe von neuen Aspekten hinsichtlich der Verwendung in Anspruch genommener Dienste. Aufgrund der bewusst gemachten Annahme von unbeschränkt zur Verfügung stehenden Ressourcen, die bei Bedarf verwendet werden können, entfällt der Betrieb eigener Hardware (z.B. Rechenzentren oder Netzwerkkapazitäten) und damit auch die Notwendigkeit für eine weitreichende und langwierige diesbezügliche Planung [AFG+10]. So können Unternehmen problemlos mit einer geringen Menge an in Anspruch genommenen Ressourcen starten und diese Menge flexibel erhöhen, sobald es bei unternehmerischem Erfolg das Wachstum erfordert. Genauso können Ressourcen kurzfristig auch wieder freigegeben werden. Bei Anwendungen, die auf großen Datenmengen arbeiten und sich hinreichend gut parallelisieren lassen, wird ein weiterer großer Vorteil des Cloud Computing deutlich. So können viel Zeit in Anspruch nehmende Berechnungen durch die Verwendung einer großen Zahl von Instanzen virtueller Computer, die an dem Problem arbeiten und jeweils entsprechend schneller ein (Teil-) Ergebnis liefern, ohne Erzeugung von höheren Kosten im Vergleich zu einer sequentiellen Problemlösung durchgeführt werden. Die Verwendung von 100 Instanzen eines Rechenknotens im Cloud Computing für jeweils eine Stunde kostet das gleiche wie eine einzelne Instanz für 100 Stunden zu verwenden. Beim Cloud Computing fallen Kosten entsprechend der tatsächlich stattgefundenen Nutzung der Ressourcen an, was für den Benutzer wirtschaftliche Flexibilität und Verlagerung eines Teils des Geschäftsrisikos auf den Anbieter der in Anspruch genommenen Cloud Computing-Dienste bedeutet. Insbesondere die Risiken der Bereitstellung von zu vielen oder zu wenigen eigenen Ressourcen (in beiden Fällen resultierend in finanziellen Nachteilen für den Geschäftserfolg) wird aus Nutzersicht vollständig beseitigt.

In [Ley09] wird das Cloud Computing als nächster Schritt und logische Weiterentwicklung nach Cluster Computing und Grid Computing angesehen. Auch hier wird das Konzept des *Utility Computing* betont, also eine verbrauchsbasierte Abrechnung wie es beispielsweise bei Strom, Gas, Telefon oder Wasser der Fall ist. Virtualisierung ermöglicht es, Anwendungen zusammen mit der benötigten *Middleware* sowie des Betriebssystems innerhalb einer virtuellen Maschine zur Verfügung zu stellen. Auf einem physikalischen Computer lassen sich dann verschiedene, möglicherweise heterogene, virtuelle Maschinen betreiben, wodurch dessen Nutzungsgrad enorm gesteigert werden kann. Als technologische Grundlage für Utility Computing und *On Demand Computing* dienen *Web Services*, mit denen Software und Hardware virtualisiert zur Verfügung gestellt werden können [Ley04]. Eine Beschreibung der für eine Anwendung benötigten Umgebung sowie der zugehörigen Ressourcen wird in eine Abfolge von Interaktionen mit Web Services übersetzt, welche dann entsprechend der Beschreibung Hardware zuweisen und auf dieser dann Software installieren und verfügbar machen. Der gesamte Vorgang der Bereitstellung von Anwendungen wird *Provisioning* genannt. [Ley09] schlägt vor, zusätzlich zu den bereits erwähnten Ebenen Infrastructure as a Service, Platform as a Service und Software as a Service eine neue Ebene *Composite as a Service* (CaaS) einzuführen, die für die Orchestrierung der verwendeten Web Services zuständig ist (Abbildung 2.1). So können Abhängigkeiten modelliert werden und aus einer Menge von Web Services können durch Komposition neue Web Services geschaffen werden. Dieses Prinzip ist aus der Domäne der *Service Oriented Architecture* (SOA) bekannt [WCL+05].

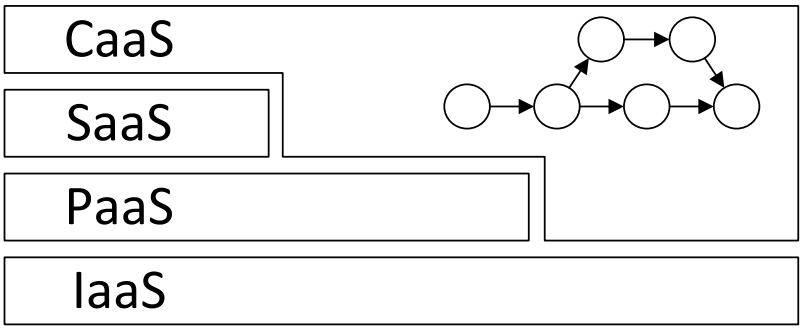


**Abbildung 2.1.:** Ebenen in der Cloud [Ley09]

In [OJ11] wird dargelegt, dass sich Standards im Bereich Cloud Computing erst in der Entwicklung befinden. Derzeit existiert eine unüberschaubare Anzahl von unterschiedlichen Ansätzen, die auf langfristige Sicht jedoch für den Erfolg des Cloud Computing in wenige grundlegende gemeinsame Definitionen, Standards und Spezifikationen zusammengeführt werden müssen. Eine aktuelle Übersicht über Standards im Bereich Cloud Computing gibt [clob]. Hier finden sich Informationen zu und Verweise auf Standards, die von verschiedenen Expertengruppen erarbeitet werden, wie z.B. der *Organization for the Advancement of Structured Information Standards* (OASIS) [oas].

[MHT11] beschreibt neben der Definition [MG11] eine Referenzarchitektur sowie Anwendungsfälle für das Cloud Computing. Der Entwicklungsstand von Standards in den drei Kategorien Interoperabilität, Portabilität und Sicherheit wird vorgestellt. Darüber hinaus bietet das *National Institute of Standards and Technology* (NIST) [nis] ein fortlaufend aktualisiertes Wiki an, das eine Übersicht über Standards mit Relevanz für das Cloud Computing gibt. Es beinhaltet im ersten Teil grundlegende (bestehende) Definitionen und Standards (z.B. HTTP, XML oder WSDL) sowie im zweiten Teil (neue) Standards mit hohem Abstraktionsgrad für das Cloud Computing sowie für Web Services (z.B. OVF [DMTF09], OCCI [occ] oder WS-Policy [ws-]).

### 2.1.2. Marktübersicht

Der Benutzer hat heute eine weitreichende Auswahl an Diensten, die in Public Clouds bereitgestellt werden. Nachfolgend werden drei ausgewählte Cloud Computing Provider vorgestellt, wobei hinsichtlich des Abstraktionsgrades der Umgebung, gegen die der Softwareentwickler bei diesen Diensten programmiert, und hinsichtlich der Verwaltung der Ressourcen große Unterschiede bestehen. Darüber hinaus werden vier Open-Source-Projekte vorgestellt, mit deren Softwarelösungen Cloud-Umgebungen betrieben werden können.

Seit 2006 ist *Amazon EC2* [awsc] als Teil der *Amazon Web Services* (AWS) [awsa] verfügbar. Eine EC2-Instanz kann mittels einer schlanken Programmierschnittstelle (API) angefordert und konfiguriert

werden und stellt ein Stück virtualisierte Hardware dar. Der Benutzer hat hierbei nahezu die vollständige Kontrolle über die betriebene Software bis hinunter zum Betriebssystemkern. [AFG+09]

Ein anderer Weg wird mit *Google AppEngine* [cloc] beschritten, das seit 2008 auf dem Markt ist. Diese Lösung bietet die Möglichkeit, Webanwendungen in der Cloud zu hosten und insbesondere von automatischer Skalierung, hoher Verfügbarkeit und der einfachen Einbindung von proprietären *MegaStore*-Datenbanken [BBC+11] zu profitieren. Die Anwendungen müssen hierbei nach dem *Request/Reply*-Prinzip arbeiten und sich in eine zustandslose Schicht für die Ausführung von Berechnungen und eine zustandsbehaftete Schicht für die Datenhaltung logisch zerlegen lassen. Google AppEngine ist aufgrund dieser Einschränkungen also nicht für *General-Purpose*-Anwendungen gedacht. [AFG+09]

Mit *Microsoft Azure* [azu] können seit 2010 Anwendungen, die entsprechende .NET-Bibliotheken [msn] verwenden und dann zu einer *Common Language Runtime* (CLR) [msc] kompiliert werden, in der Cloud verfügbar gemacht werden. Es werden General-Purpose-Anwendungen unterstützt und der Benutzer kann aus verschiedenen Programmiersprachen für die Implementierung wählen [AFG+09]. Seit 2012 bietet Microsoft Azure die Möglichkeit, virtuelle Maschinen mit Windows oder Linux als Betriebssystem zu betreiben [Fab13].

Mit Hilfe von Open Source-Lösungen wie *Eucalyptus* [git], *OpenNebula* [opeb], *OpenStack* [opea] sowie *CloudStack* [clod] besteht neben der Inanspruchnahme von Diensten eines Public Cloud Providers die Möglichkeit, eine eigene Cloud-Infrastruktur im Rahmen einer Private Cloud zu betreiben.

Eucalyptus ist eine Open Source Cloud Platform-Software, die sich durch ein stark dezentralisiertes Design auszeichnet. Hierbei ist zum einen das verteilte Speichersystem *Walrus* zu nennen, welches Amazons S3 [s3] immitiert. Virtuelle Laufwerke für die Instanzen der virtuellen Maschinen hingegen werden in lokalen Speicherbereichen abgelegt. Zusammen mit der Unterstützung von stark geclusterten Systemen folgt aus den genannten Eigenschaften, dass sich Eucalyptus vor allem für die Bereitstellung einer Vielzahl von virtuellen Maschinen eignet. Der Zugriff für Anwender auf das System erfolgt via Web Interface oder mit Front-End-Tools (z.B. Eucalyptus eigene *euca2ools*). Bereiche für Anwender und Administratoren sind streng voneinander getrennt und Komplexitäten des zugrunde liegenden Systems werden mit wenigen Ausnahmen vor dem Anwender versteckt. Bedingungen für den Betrieb von Eucalyptus sind anhand der vorgestellten Merkmale vornehmlich im unternehmerischen Umfeld auszumachen. [ST10]

Das im Rahmen eines europäischen Forschungsprojekts entstandene und sich aktiv in der Weiterentwicklung befindende Werkzeug OpenNebula dient der Bereitstellung, Orchestrierung und Konfiguration von virtuellen Maschinen. Das unter der Apache 2.0-Lizenz [apaa] stehende Projekt wurde ursprünglich für den Einsatz in Rechenzentren konzipiert. Mittlerweile konzentriert man sich aber ebenfalls auf das Cloud Computing (Private- / Public- / Hybrid Cloud). Hauptziele sind neben Skalierbarkeit vor allem Performanz und Stabilität, so dass auch ein Einsatz in einem sehr großen Maßstab kein Problem darstellt. Mit SURFsara [sur] und Telefonica [tel] hat das Projekt zwei bedeutende Partner aus den Bereichen (Hochleistungs-) Rechenzentren und Telekommunikation. OpenNebula unterstützt das *Open Grid Forum* (OGF) [ogf] *Open Cloud Computing Interface* (OCCI) [occ], die Amazon EC2-API und die vCloud-API [vcl] von VMware [vmw]. [MLM11]

Bei dem Softwareprojekt OpenStack handelt es sich um eine Cloud Platform, deren Fokus auf massiver Skalierbarkeit, Kompatibilität und Flexibilität hinsichtlich verschiedener Virtualisierungstechnologien

sowie auf Offenheit liegt. Viele Firmen unterstützen aktuell das Projekt, darunter AMD [amd], Canonical [can], Dell [del], Intel [int] und HP [hp.]. Es basiert größtenteils auf Code der NASA [nas] und *Rackspace Cloud* [rac]. Derzeit unterstützt OpenStack die APIs von Amazon EC2 und Rackspace. Aufgrund der weiten Unterstützung durch die Partner und der Community ist die zukünftige Entwicklung von OpenStack vielversprechend. [SAE12]

Apache CloudStack ist eine Open Source Cloud Platform für das Deployment und Management von großen Mengen virtueller Maschinen als hoch verfügbarer und skalierbarer Infrastructure as a Service. Insbesondere hinsichtlich der erreichbaren Performanz der virtuellen Maschinen erzielt CloudStack (z.B. im Vergleich zu Eucalyptus) sehr gute Ergebnisse und ist daher als Alternative zu den zuvor genannten Anbietern zu nennen. Vor allem für das Hosting von Webanwendungen und als Private Cloud-Lösung besitzt CloudStack Stärken. [AMM14]

## 2.2. Infrastructure as Code

Trends wie *Continuous Delivery* [HF10a], *Test-Driven Development* [Mad10] und Automatisierung der Vorgänge *Building* [HF10b] und *Deployment* [BBKL14] haben in den letzten Jahren den Markt durchdrungen. Sie erleichtern die Durchführung und Verwaltung vieler Teile des Produktlebenszyklus einer Software. Die Infrastruktur, die für den Betrieb einer Anwendung notwendig ist, wird von den genannten Konzepten jedoch zumeist außen vor gelassen. Sie wird häufig mit manueller Arbeit oder teilautomatisiert mit individuell angefertigten Skripten verwaltet, was aufwendig und fehleranfällig ist. Dieses Problem adressiert das Konzept *Infrastructure as Code*, mit dessen Hilfe die von einer Anwendung benötigte Laufzeitumgebung (z.B. Betriebssystem, Netzwerk oder Firewall) mittels ausführbarer Spezifikationen bereitgestellt und konfiguriert werden kann. Insbesondere schließt das Konzept auch die verwendete Middleware ein (z.B. Konfiguration eines Webservers durch Konfigurationsdateien, Softwarepakete als Teil des Betriebssystems oder *Cron-Jobs*). So kann mittels einer Spezifikation eine Entwicklungs- oder Testumgebung kurzfristig bereitgestellt oder wieder gelöscht werden. Idealerweise werden die Spezifikationen zentral in einem *Repository* unter *Versionskontrolle* verwaltet, wodurch bei Bedarf eine effiziente und reproduzierbare Bereitstellung von Ressourcen erreicht wird. [Hü12]

Für die Anwendung von Infrastructure as Code steht eine Reihe von Werkzeugen zur Verfügung. Zu nennen sind hier die Open Source-Werkzeuge Ansible [ans], BCFG2 [bcf], CFEngine [cfe], Juju [jujc], Opscode Chef [get], Puppet [pupb] und Salt [sal]. Puppet, Opscode Chef und Juju werden als Auswahl von Werkzeugen mit unterschiedlichem Funktionsumfang im Folgenden kurz vorgestellt. [DJV10]

### 2.2.1. Puppet

Puppet ist ein Open Source *Configuration Management*-Werkzeug zur zentralen Verwaltung von IT-Infrastruktur verschiedenster Systemarchitekturen (Linux, Unix und Windows), das unter der Apache 2.0-Lizenz [apaa] steht. Die Installation und Konfiguration von neuen Ressourcen erfolgt nach dem *Client/Sever*-Modell mit einem *Master* (Server) und einer Menge von *Agenten* (Clients). Puppet verarbeitet Code in einer *Domain Specific Language* (DSL), der sowohl deklarativ als auch imperativ

sein kann. Puppet hat eine sehr große User Community und modelliert das zu konfigurierende System auf einem hohen Abstraktionslevel [Rah12].

Abbildung 2.2 zeigt, wie ein Puppet-Agent mit Hilfe eines Puppet-Masters in einen definierten Systemzustand gebracht wird. Der Agent fordert vom Master periodisch die aktuell vorgesehene Systemkonfiguration an, indem er eine Menge von Daten sowie den Namen des Knotens, auf dem der Agent ausgeführt wird, an den Master sendet. Der Master ermittelt daraufhin alle benötigten Informationen und kompiliert einen *Catalog*, welcher anhand einer Liste von Ressourcen und deren Abhängigkeiten untereinander den vorgesehenen Zustand für den Agenten definiert. Nachdem der Agent den Catalog erhalten hat überprüft dieser die aufgelisteten Ressourcen und führt alle notwendigen Konfigurationsänderungen am System aus, so dass der gewünschte Zustand erreicht wird. Nachdem der Catalog angewendet worden ist, schickt der Agent abschließend einen Bericht an den Master. [pupa]

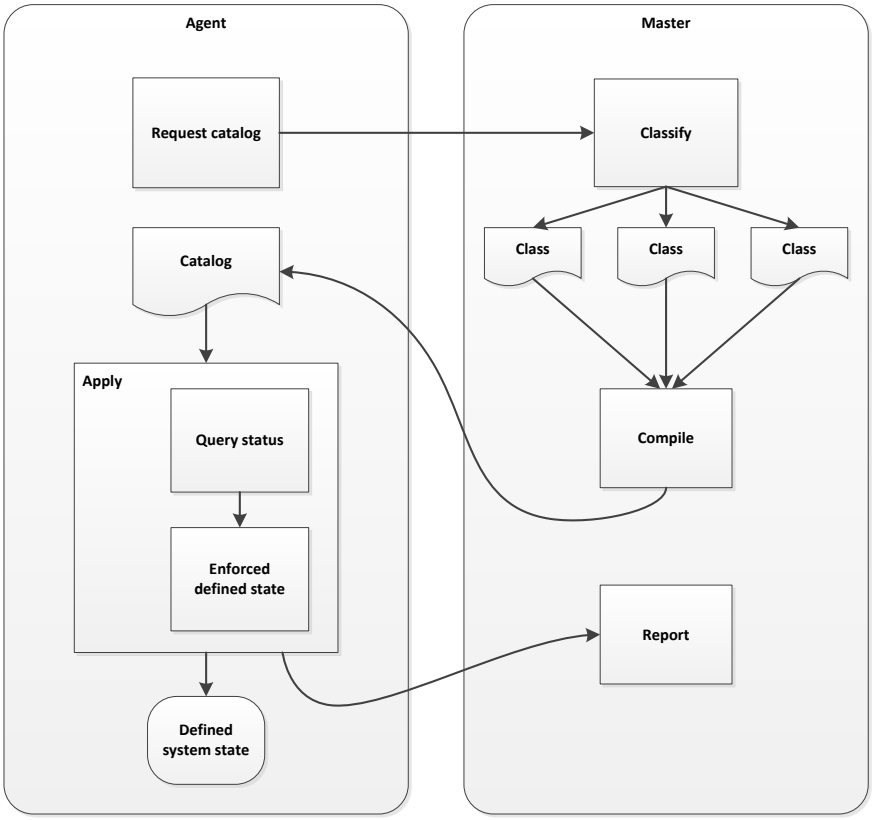


**Abbildung 2.2.:** Puppet im Agent/Master-Modus [pupa]

### 2.2.2. Opscode Chef

Ebenso wie Puppet ist auch Chef ein Open Source Configuration Management-Werkzeug unter der Apache 2.0-Lizenz [apaa], das nach dem Client/Server-Modell arbeitet und Eingaben in einer Ruby DSL erwartet. Anders als Puppet verhält sich Chef bei der Reihenfolge der Ausführung von Konfigurationsregeln deterministisch, was vor allem das *Debugging* erleichtert. Die Philosophie „thick client and thin server“, bei der soviel Rechenarbeit (Kompilierung und Ausführung von Code) wie möglich auf die Clients ausgelagert wird, ist für die Skalierbarkeit von Chef von Vorteil. [Pan12]

Abbildung 2.3 zeigt die grundlegenden Komponenten von Chef sowie deren Zusammenspiel. Die zu administrierenden Knoten (*nodes*) können unterschiedlicher Natur sein, die einzige Voraussetzung ist, dass sie die *chef-client*-Software ausführen können. *Cookbooks* stellen die wesentlichen Einheiten für die Konfiguration sowie für die Verteilung von Richtlinien dar. Sie enthalten alle Bestandteile, die für das Erreichen eines definierten Systemzustands notwendig sind. Der *chef-server* dient als Knotenpunkt für die Konfigurationsdaten. Hier werden *cookbooks*, anzuwendende Richtlinien und Metadaten für jeden registrierten Knoten gespeichert. Der chef-server wird mittels des Kommandozeilenwerkzeugs *knife* verwaltet, welches auf einem Arbeitsplatzrechner ausgeführt wird. Über diesen Rechner ist darüber hinaus der Zugriff auf ein Repository von cookbooks möglich. [che]

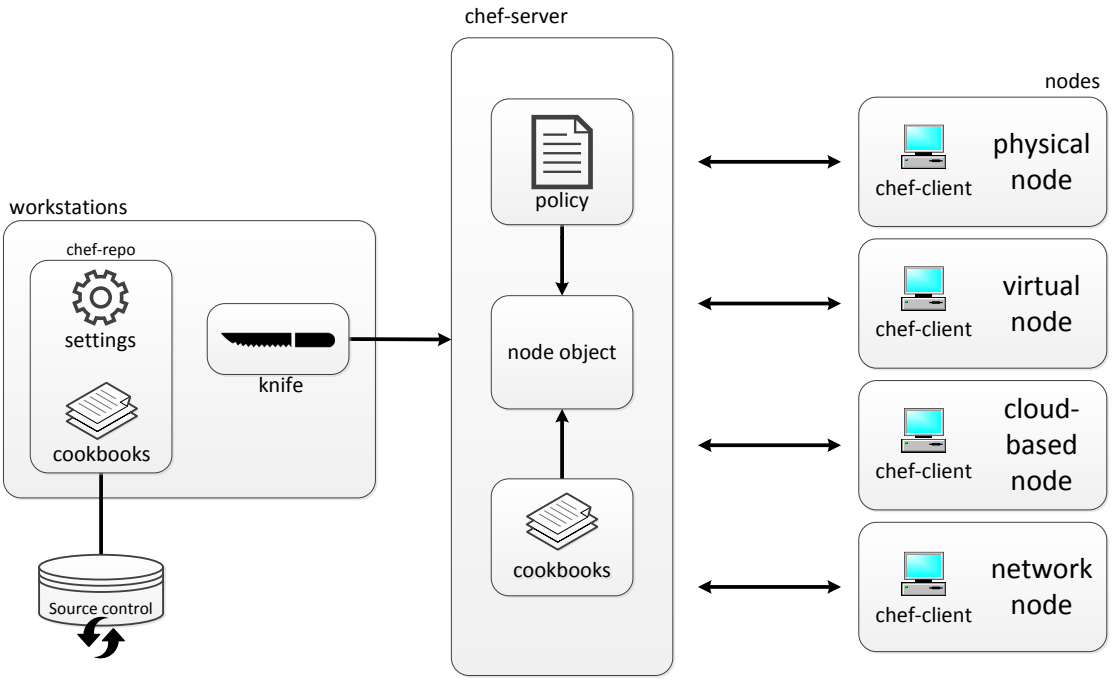


**Abbildung 2.3.:** Chef im Client/Server-Modus [che]

### 2.2.3. Juju

Die Software Juju von Canonical [can] ist ein Werkzeug zur Orchestrierung von Diensten. Mit Juju lassen sich einfach und schnell Dienste bereitstellen und verwalten. Ähnlich wie eine Programminstallation über die Paketverwaltung in Ubuntu [ubua] (und anderen linuxbasierten Systemen) können mit Juju Dienste installiert werden. Die hierfür verwendeten Komponenten werden *Charms* genannt. Dies sind Softwarekomponenten bestehend aus einem oder mehreren Diensten sowie allen Abhängigkeiten und notwendigen Konfigurationen. Juju Charms können auf Cloud Diensten wie Amazon Web Services, Microsoft Azure, OpenStack, HP Cloud [hpc] oder in Kombination mit Ubuntu MAAS [maa] bereitgestellt werden. [KB13]

Die Bereitstellung eines lauffähigen Wordpress-Blogs [wor] inklusive einer Datenbank erfolgt beispielsweise mit den in Listing 2.1 gezeigten Befehlen direkt über die Kommandozeile.

**Listing 2.1** Installation eines Wordpress-Blogs mittels Juju

```
juju deploy wordpress
juju deploy mysql
juju add-relation wordpress mysql
juju expose wordpress
```

Abbildung 2.4 zeigt, wie mit Hilfe von Juju Dienste wie z.B. Wordpress, Apache Tomcat [tom], Apache Hadoop [had], ceph [cep], Apache HTTP Server [apab], Apache Cassandra [cas] oder MySQL [mys] auf Gastbetriebssystemen in der Cloud installiert und adminstriert werden können. Die Gastbetriebssysteme können sowohl in einer Private Cloud als auch in einer Public Cloud gehosted sein, wobei z.B. Openstack, Kernel-based Virtual Machines (KVMs) [kvm], LXC - Linux Containers [lxc] oder Ubuntu Server [ubub] verwendet werden können. Die Juju-Software ist entweder mit einer grafischen Benutzeroberfläche oder als Kommandozeilenprogramm verwendbar. [juja]

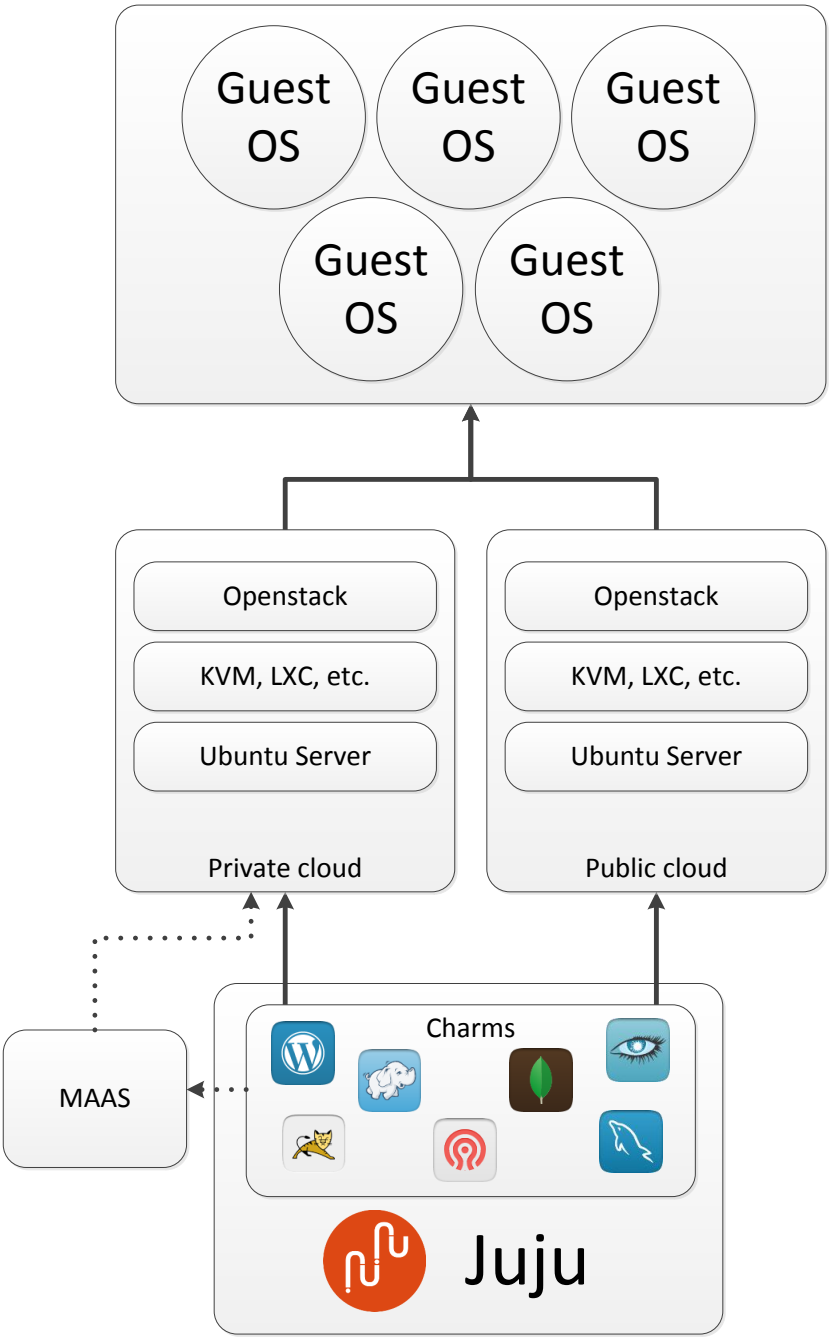


**Abbildung 2.4.:** Funktionsweise von Juju [juja]

## 2.3. DevOps

Die Prinzipien der agilen Softwareentwicklung wurden im Jahr 2001 im *Manifesto For Agile Software Development* [agi] formuliert. Sie zeigen eine Alternative zum traditionellen, schwergewichtigen Ansatz auf, Software zu entwickeln. Gefordert wird ein Wandel der Unternehmenskultur, der mit den vier Statements „Menschen und Miteinander sind wichtiger als Prozesse und Werkzeuge“, „Funktionierende Software ist wichtiger als eine verständliche Dokumentation“, „Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen“ und „Anpassungen an Veränderungen sind wichtiger als streng einem Plan zu folgen“ beschrieben wird. 12 Prinzipien konkretisieren diese Forderung, wobei das wichtigste dieser Prinzipien die Zufriedenstellung der Kunden durch eine frühzeitige und kontinuierliche Lieferung von guter Software ist.

Agile Softwareentwicklung wird heute in einem Großteil der Unternehmen eingesetzt. Vor allem die Verbesserung der Reaktionsfähigkeit auf sich ändernde Anforderungen und die Erhöhung der Transparenz im Entwicklungsprozess sorgen dafür, dass agile Softwareentwicklung zumeist auf positive Resonanz der Beteiligten stößt. Die mit Abstand am häufigsten verwendeten Methoden sind *Scrum* [Coh09] bzw. Varianten von Scrum, gefolgt von *Kanban* [Sha11] bzw. Varianten von Kanban. [Ver13]

Traditionell hatten Betriebsabteilungen vergleichsweise viel Zeit für das Management von Stabilität, Risiko und Performanz der von ihnen betreuten Anwendungen. Planung und Verwaltung der benötigten Hardware waren davon losgelöste Prozesse. Der aktuelle Trend Richtung Cloud Computing und Virtualisierung sowie die weitreichende Adaption von agiler Softwareentwicklung und der damit verbundenen häufigen, schnellen *Software-Releases* stellt die Betriebsabteilungen vor neue Herausforderungen. Automatisierung von zuvor manuell ausgeführten Aufgaben bei Deployment und Betrieb von Software, wie in Abschnitt 2.2 beschrieben, stellt einen technischen Ansatz dar, diesen Herausforderungen entgegenzutreten. [Azo11]

Die Konzepte der DevOps-Bewegung beziehen sich jedoch nicht nur auf die technischen Aspekte, sondern es wird vielmehr eine ganzheitliche Sicht auf die aktuellen Situationen in Softwareentwicklungs- und Bereitstellungsprozessen angelegt, indem vor allem die Prozesse an die neuen Anforderungen angepasst werden. Die unterschiedlichen Zielsetzungen der Unternehmensbereiche Softwareentwicklung und Softwarebetrieb werden als grundlegendes Problem identifiziert. Hauptaufgabe der Softwareentwicklung ist es, Änderungen zu planen, zu implementieren und vorzunehmen (z.B. *Bug-Fixes* oder neue *Features*). Im Gegensatz dazu ist der Betrieb darauf bedacht, Konfigurationen von Soft- und Hardware möglichst unverändert zu lassen sowie Einschränkungen möglichst restriktiv zu handhaben, so dass eine sichere und zuverlässige Bereitstellung von Anwendungen gewährleistet ist (Abbildung 2.5). Genauso wie durch die agile Softwareentwicklung die Bereiche Programmieren und Testen in neue, interdisziplinäre Teams (Entwicklung) mit gemeinsamen Zielen zusammengeführt worden sind, sollen nun durch DevOps die Bereiche Entwicklung und Betrieb zusammengeführt werden. [Hü12]

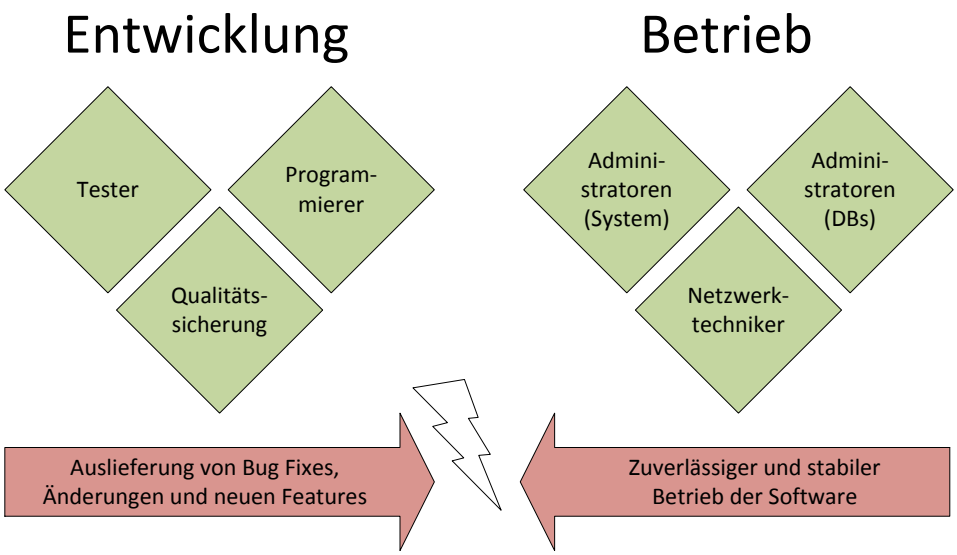


**Abbildung 2.5.:** Änderungen vs. Stabilität [Hü12]

In [Smi11] wird der Begriff DevOps als ein neuer Ansatz für die Bereitstellung von IT-Dienstleistungen definiert, dessen Wurzeln in der agilen Softwareentwicklung ausgemacht werden. Zielsetzung ist eine enge Zusammenarbeit von Entwicklung und Betrieb, um unternehmerische Ziele (z.B. zuverlässige Bereitstellung einer Webseite) zu erreichen. So werden mit Hilfe des Cloud Computings sowie Automatisierung in Deployment und Management von Anwendungen insbesondere Skalierbarkeit und hohe Verfügbarkeit von Web-Anwendungen erreicht. Für DevOps existieren (noch) keine konkreten Regeln, Standards oder Frameworks, wie z.B. der *IT Infrastructure Library* (ITIL) [iti] oder der *Capability Maturity Model Integration* (CMMI) [cmm]. Vielmehr handelt es sich um eine Sammlung von Empfehlungen und grob formulierten Prinzipien. Zwischen Entwicklung und Betrieb soll eine Kultur des Zusammenarbeitens und Kommunizierens etabliert werden, was in der Praxis vor allem für Realease Management-Prozesse wichtig ist. Zum anderen sollen Konzepte wie Infrastructure as Code als Grundlage für verschiedene Möglichkeiten der Automatisierung im *Application Lifecycle Management* (ACM) vorangetrieben werden.

In [JH11] werden die Herausforderungen, denen die DevOps-Bewegung entgegenzutreten versucht, beschrieben. Auf der einen Seite soll auf sich verändernde Marktsituationen und Anforderungen möglichst flexibel und schnell reagiert werden können. Auf der anderen Seite wird bei gleichzeitig stetig steigender Komplexität sowie Heterogenität der Softwaresysteme ein zuverlässiger und störungsfreier Betrieb erwartet. Es wird vorgeschlagen, interdisziplinäre Teams aus Entwicklern, Testern und Betriebspersonal zu bilden. Diese Teams sind jeweils einem Produkt zugeordnet und betreuen den gesamten Produktlebenszyklus. Hierbei werden die Vorgänge *Building*, *Testing* und *Deployment* von Software nach Möglichkeit automatisiert. Vor allem das Deployment wird durch Virtualisierung, Cloud Computing sowie Configuration Management Tools wie Puppet oder Chef deutlich vereinfacht

im Vergleich zur manuellen Ausführung entsprechender Vorgänge. Die gemeinsame Nutzung über Abteilungsgrenzen hinweg von Techniken und Werkzeugen aus der Softwareentwicklung sowie von Technologien zum Management von IT-Umgebungen und -Infrastruktur stellt ein grundlegendes Konzept von DevOps dar. Ermöglicht werden sollen Deployments auf Abruf (*Self-Service Deployments*) bei denen die Anwendung und die benötigte Version der Umgebung automatisch zur Verfügung gestellt werden.

[Fel13] argumentiert, dass der Einsatz von DevOps für Firmen aus dem IT-Bereich unerlässlich ist, sowohl für *Software-StartUps*, deren Erfolg maßgeblich von der Qualität der Entwicklung und des Betriebs der Software abhängt, als auch für große Unternehmen wie Etsy [ets], Netflix [net], Facebook [fac], Amazon [awsa], Twitter [twi] oder Google [goo], die trotz einer extremen Anzahl an täglich stattfindenden Deployments ein hohes Maß an Stabilität, Zuverlässigkeit und Sicherheit der Software garantieren müssen. DevOps adressiert das Problem, dass Softwareprojekte ständig größer und komplexer werden und dadurch die Störungsanfälligkeit beim Deployment und während des Betriebs steigen. Eine konkreter Lösungsansatz für dieses Problem wird darin gesehen, die zu entwickelnde Softwareanwendung von Anfang an in Einheit mit der zugehörigen Umgebung zu entwickeln und zu testen. Hierfür ist die Bereitstellung von (virtualisierten) Produktumgebungen für den Softwareentwicklungsprozess erforderlich, die außer der zu entwickelnden Anwendung alles enthält, was benötigt wird (z.B. Datenbanken, Betriebssystem, Netzwerk oder eine Virtualisierungsschicht). Dies kann vergleichsweise einfach durch Starten einer Instanz einer virtuellen Maschine erfolgen oder als ein komplexer Prozess mittels eines automatisierten *Buildsystems*, das die Umgebung von Grund auf neu erstellt. Somit wird die agile Softwareentwicklung, die als Ausgabe fertigen Code liefert, dahingehend erweitert, dass nun fertiger Code in Kombination mit einer portablen Umgebung, in der dieser Code läuft, ausgegeben wird. Neben dem Einsatz von DevOps-Werkzeugen müssen die Abteilungen Softwareentwicklung und Softwarebetrieb hierfür eng zusammenarbeiten und Fachwissen in beide Richtungen transferieren. Entwickler müssen Fähigkeiten und Verantwortlichkeiten von Mitarbeitern des Softwarebetriebs übernehmen, Mitarbeiter des Softwarebetriebs müssen Fähigkeiten und Verantwortlichkeiten von Entwicklern übernehmen.

Die Anwendung von DevOps-Prinzipien setzt weder den Einsatz von Virtualisierung noch den von Cloud Computing voraus. Jedoch ist festzustellen, dass eine Nutzung dieser Technologien viele Vorteile mit sich bringt und für ein auf hohe Performanz ausgelegtes Development-, Release- und Deploymentsystem unverzichtbar ist. [Lou12]

## 2.4. NoOps und AppOps

Der Begriff *NoOps* (No Operations) beschreibt die Weiterentwicklung des DevOps-Paradigmas hin zu einer Situation, in der Betriebsaspekte für Entwickler weitestgehend transparent sind. Deployment, Skalierung und Verwaltung der Entwicklerartefakte werden mittels Diensten durchgeführt, so dass ein Entwickler sich auf die Implementierung der Anwendungslogik konzentrieren kann anstatt sich mit Betriebsspezifika auseinandersetzen zu müssen. Systeme zur Automatisierung (z.B. CloudFoundry [cloe]) stehen für den gesamten Lebenszyklus einer Anwendung zur Verfügung, wodurch eine manuelle Administration durch Betriebspersonal entfällt. [LC12]

Als Alternative zu dem Begriff NoOps wird häufig gefordert, den Begriff *AppOps* (Application Operations) zu verwenden, da dieser besser beschreibe, dass bei PaaS-Lösungen nur noch die Anwendung selbst verwaltet werden muss und nicht mehr die (virtualisierte) Infrastruktur [Fin14].

[Har12] beschreibt, dass DevOps-Konzepte vor allem auf IaaS-Dienste zugreifen. Es wird also mit virtualisierter Infrastruktur gearbeitet. Tätigkeiten wie Provisioning, Konfiguration, Änderung oder Verwaltung der Ressourcen obliegen dem Benutzer. NoOps hingegen setzt eine Abstraktionsschicht weiter oben an, indem auf PaaS-Dienste zurückgegriffen wird, wie z.B. auf Dienste für die Verwaltung von Netzwerken und virtueller Hardware oder auf Dienste wie Provisioning, Lastverteilung oder Sicherheit.

In [Coc12] wird über die aktuelle Anwendung von NoOps bei Netflix [net] berichtet. Eine konsequente Ausrichtung der Softwarearchitektur auf das Cloud Computing basierend auf *NoSQL* [Cat11] war der erste Schritt. Einige wenige Mitarbeiter aus Bereichen des IT-Betriebs wurden in Entwicklerteams transferiert, die auf Linux basierende Images für virtuelle Maschinen erstellen (z.B. *Amazon Machine Image* (AMI) [lin]). Die Integration von Entwicklertools (Perforce [per] als Versionskontrolle, Ivy [apac] als Build-Management-Werkzeug, Jenkins [jen] als Continuous Integration-Werkzeug und Artifaktory [art] als Repository für Binärdateien) zu einer durchgehenden *Tool-Chain* ermöglichen die automatisierte Erzeugung von vollständigen AMIs für den Betrieb von Diensten. Mittels AWS Autoscale Groups [amaa] können hierbei identische Instanzen von Services erzeugt werden. Jedes Entwicklerteam kann über ein Webportal seine Software verteilen und in Echtzeit testen, wobei mit Hilfe der Tool-Chain dabei ein Großteil der Arbeit automatisiert wird. Die Software bei Netflix unterliegt keiner zentralen Kontrolle, vielmehr ist jedes Team selbst verantwortlich für die eigene Software, für deren Abhängigkeiten sowie für die Verwaltung der entsprechenden *AWS Security Groups* [AWSd].

Für die folgenden Ausführungen in der vorliegenden Arbeit ist eine Unterscheidung zwischen DevOps und NoOps nicht notwendig. Vielmehr kann NoOps als Spezialfall von DevOps angesehen werden, auf die sich die vorgestellten Konzepte gleichermaßen anwenden lassen.

## 2.5. Verwandte Arbeiten

In der Abhandlung *Pattern Based SOA Deployment* [AEK+07] wird ein Ansatz vorgestellt, mit dessen Hilfe die Konfiguration der Hosting-Infrastruktur von Diensten in einer SOA-Umgebung erleichtert werden soll. Hierbei kommen modellbasierte *Deployment-Patterns* zum Einsatz, mit denen nichtfunktionale Eigenschaften von Diensten abstrakt und strukturiert formuliert werden können. Im Gegensatz zu der häufig üblichen Vorgehensweise, Deployments anhand von informellen Dokumentationen durchzuführen, die diese Vorgänge langsam, teuer und fehleranfällig machen, erlangt man auf diesem Wege insbesondere die Vorteile, dass Deployment-Patterns wiederverwendbar und zusammensetzbar sind und iterativ verbessert werden können. Zudem wird ein Algorithmus vorgestellt, mit dem sich diese Deployment-Patterns automatisiert in verschiedenen verteilten Umgebungen für die Bereitstellung von Diensten instanziieren lassen. Die vorgestellten Konzepte lassen sich mit der vorliegenden Arbeit verbinden, indem die in Abschnitt 3.2 beschriebene Wissensdatenbank zur systematischen Organisation von Deployment-Patterns genutzt wird. Eine Auswahl an Patterns aus

der Wissensdatenbank kann dann dazu verwendet werden, Systemkonfigurationen entsprechend der zu beachtenden Anforderungen und Einschränkungen durchzuführen.

Ein Ansatz, der ebenfalls das Konzept von Patterns verwendet, wird in der Abhandlung *Pattern-based Deployment Service for Next Generation Clouds* [LSS+13] vorgestellt. Mittels einer deklarativen, XML-basierten DSL werden Patterns formuliert, die komplexe Deployment-Szenarien, Entitäten, Beziehungen und Dienste beschreiben. Patterns solcher Art dienen als Eingabe für einen Dienst, mit dessen Hilfe das Deployment von Anwendungen in Cloud-Umgebungen durchgeführt wird. Dieser Ansatz integriert die Konzepte von *System Patterns*, *Multi-Cloud* und unterschiedlichen etablierten Vorgehensweisen aus der Praxis, wie z.B. die Verwendung von *RESTful Services* und *Configuration Management Tools*. Genauso wie das Konzept der zuvor genannten Deployment-Patterns aus [AEK+07] lässt sich auch dieser Ansatz mit der vorliegenden Arbeit verbinden.

Die Abhandlung *WS-Policy4MASC - A WS-Policy Extension Used in the MASC Middleware* [TEM07] zeigt mit einer Erweiterung von *WS-Policy* in Kombination mit der Middleware *Manageable and Adaptive Service Composition* (MASC) eine Möglichkeit auf, Web Services zu überwachen und dynamisch an Änderungen der Laufzeitumgebung anzupassen. Hierfür wird von *WS-Policy extension* Gebrauch gemacht, d.h. es werden durch *XML Schemata* definierte neue Typen von *WS-Policy assertions* eingeführt. *Goal policy assertions* spezifizieren hierbei Anforderungen und Garantien, die während der normalen Systemlaufzeit gelten (z.B. *Die Antwortzeit einer bestimmten Komponente soll kleiner sein als 1 Sekunde*). *Action policy assertions* spezifizieren Aktionen, die unter bestimmten Voraussetzungen ausgeführt werden (Beispiele hierfür sind das Hinzufügen, Entfernen, Ersetzen, Überspringen und *Retrying* von Subprozessen oder Aktivitäten). Für die Spezifizierung von monetären Aspekten werden *utility policy assertions* verwendet und mit *meta-policy assertions* wird spezifiziert, welche *action policy assertions* gegenseitige Alternativen sind und welche Strategie beim Auftreten von Konflikten angewendet wird. Kompositionen aus Web Services können verständlich mittels *WSDL* [wsd], *WS-BPEL* [wsb] und *WS-Policy4MASC* beschrieben werden, wobei eine Menge von neuen Regeln für das Management (Monitoring und Steuerung) von Web Services zur Verfügung stehen. Dieser Ansatz setzt jedoch zum einen die Verwendung von Web Services voraus und ist zum anderen nicht integrierbar mit den in Abschnitt 2.2 vorgestellten Werkzeugen, so dass bestehende Spezifikationen für diese Werkzeuge nicht wiederverwendet werden können.

# 3. Vorbereitung des Softwarebetriebs aus Entwicklersicht

In diesem Kapitel werden die entwickelten Konzepte für die Vorbereitung des Softwarebetriebs aus Entwicklersicht vorgestellt. Zunächst wird in Abschnitt 3.1 erklärt, an welchen Stellen Softwareentwickler bei der Integration von automatisierten Release-Prozessen und Infrastructure as Code besser unterstützt werden können. Im Anschluss wird in Abschnitt 3.2 eine Taxonomie für Implementierungen von Softwarekomponenten vorgestellt. Abschnitt 3.3 stellt eine mögliche Struktur einer Spezifikation vor, die als Eingabe für ein Werkzeug für die Bereitstellung von Infrastrucure as Code dient. Sowohl die Taxonomie als auch die Spezifikation werden im weiteren Verlauf dieser Arbeit verwendet. In Abschnitt 3.4 wird ein beispielhaftes Szenario für den Einsatz der vorgestellten Konzepte erstellt, bevor in Abschnitt 3.5 alle wichtigen Anforderungen an diese Konzepte aufgeführt werden. Schließlich wird in Abschnitt 3.6 ein Lösungsansatz präsentiert, der alle gestellten Anforderungen erfüllt.

## 3.1. Forschungsaufgaben

Die Entwicklung von Software auf der einen Seite und die automatisierten Release-Prozesse mittels Infrastructure as Code auf der anderen Seiten stellen zwei unterschiedliche Domänen dar, die jeweils eigene Konzepte, Strukturen, Eigenheiten, Semantiken und Syntaxen besitzen. So erfolgt die Implementierung der Anwendungslogik mittels Programmiersprachen (C, C++, Java, Go, …), Skriptsprachen (Javascript, PHP, …) oder Domain Specific Languages (SQL, reguläre Ausdrücke, …). Dies stellt die Domäne der Softwareentwickler dar, die mit diesen Sprachen arbeiten und deren Funktionsweise daher oft bis ins kleinste Detail kennen. Im Gegensatz dazu erfolgen die automatisierten Release-Prozesse zumeist über spezielle Werkzeuge, deren Wurzeln in der Administration und Verwaltung des Softwarebetriebs liegen, über Konfigurationseinstellungen verschiedenster beteiligter Komponenten sowie über *Shell Scripts*.

Aktuelle Werkzeuge aus dem DevOps-Bereich (Chef, Puppet, Juju, …), im Folgenden *DevOps-Werkzeuge* genannt, erleichtern das Deployment von Anwendungen durch die Verwendung bereits bestehender, wiederverwendbarer Verwaltungs-Routinen für Softwareimplementierungen. So erfordert beispielsweise das Deployment des in Listing 3.1 gezeigten Juju Charms lediglich einige wenige Shell-Kommandos (vergleichbar zur Installation von lokalen Anwendungen mit Hilfe von Paketmanagern wie *aptitude* [apt] in Debian-basierten Linuxsystemen). Zur Veranschaulichung zeigt Listing 3.1 einen gekürzten Ausschnitt eines Juju-Charms aus dem Juju Charm Store [jujd] für die Installation eines Mediawikis [med]. Der Juju Charm installiert die beiden Dienste *mediawiki* und

*mysql*, welche jeweils durch eine Menge von Attributen konfiguriert und miteinander verbunden werden.

**Listing 3.1** Juju Charm für die Installation eines Mediawikis [jujb]

```
single:
  relations:
    - mediawiki:db
    - mysql:db
  series: precise
  services:
    mediawiki:
      charm: cs:precise/mediawiki-10
      options:
        debug: false
        skin: vector
    mysql:
      charm: cs:precise/mysql-28
      num_units: 1
      options:
        binlog-format: MIXED
        block-size: 5
        dataset-size: 80%
        flavor: distro
```

Für die Erstellung neuer oder die Überarbeitung bestehender Verwaltungs-Routinen bedarf es neben der Kenntnis der Funktionsweise des verwendeten DevOps-Werkzeuges insbesondere einem fundierten Wissen aus dem Bereich des Softwarebetriebs. So stellen sich eine Reihe von betriebsspezifischen Fragestellungen hinsichtlich der Anforderungen, Konfigurationen und Einschränkungen aller verwendeter Komponenten, die von Entwicklern nicht effektiv bearbeitet werden können. Es besteht also eine Lücke zwischen den beiden Domänen Softwareentwicklung und Softwarebetrieb, die es zu schließen gilt.

Für Entwickler, die neue Verwaltungsroutinen erstellen möchten, muss eine Möglichkeit geschaffen werden, dieses durch die Verwendung von Konzepten und Sprachen aus ihrer Domäne zu erreichen. Hierbei ist eine weitgehende Abstraktion der Mächtigkeit und zugleich auch der Komplexität der Konzepte und Konfigurationsmöglichkeiten des Softwarebetriebs notwendig.

Darüber hinaus existiert noch keine standardisierte Möglichkeit, während des Betriebs Konfigurationsänderungen durchzuführen bzw. automatisiert DevOps-Spezifikationen flexibel auf veränderte Anforderungen und Einschränkungen hin anzupassen.

## 3.2. Wissensdatenbank

Eine Wissensdatenbank dient als Grundlage für die im Folgenden beschriebenen Konzepte. Mittels einer Taxonomie werden hier Implementierungen klassifiziert, die für Softwarebereitstellungsprozesse nach dem DevOps-Paradigma wichtig sind.

Folgende Hauptkategorien sind in der Wissensdatenbank enthalten:

- Anbieter (Provider)
- Anwendung (Application)
- Dienstschicht (Middleware)
- Infrastruktur (Infrastructure)
- Werkzeug (DevOpsware)

Die Sub-Taxonomien der genannten Hauptkategorien werden in den Abbildungen 3.1 bis 3.5 anhand von Diagrammen mit Generalisierungen veranschaulicht. Abstrakte Entitäten werden hierbei durch Ellipsen symbolisiert, konkrete Implementierungen durch Rechtecke. Die Abbildungen zeigen beispielhaft jeweils nur einen kleinen Ausschnitt der entsprechenden Taxonomie.

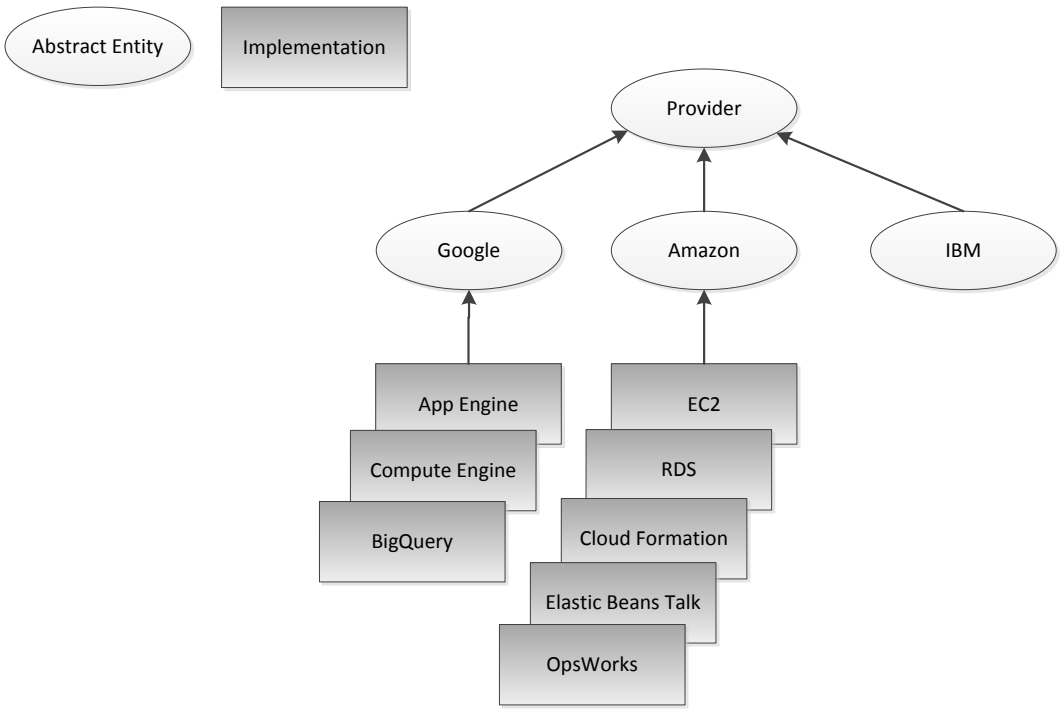


**Abbildung 3.1.:** Taxonomie der Anbieter

Die Implementierungen können verschiedene Attribute besitzen. Diese Attribute beschreiben zum einen den Wert einer Eigenschaft (z.B. beinhaltet das Attribut *version* den Wert der entsprechenden Softwareversion), zum anderen werden Implementierungen aber auch miteinander in Verbindung gesetzt (z.B. verbindet das Attribut *virtualization* eine Implementierung mit einer anderen, welche

als Virtualisierungskomponente ausgewählt ist. Folgende Attribute sind zum aktuellen Stand in der Wissensdatenbank verfügbar:

- virtualisiert durch (virtualization)
- integriert mit (integrated with)
- referenziert (references)
- ist Alternative zu (alternative to)
- Artefakt-Typ (artifact type)
- Version (version)
- gehostet auf (hosted on)
- benötigt (requires)

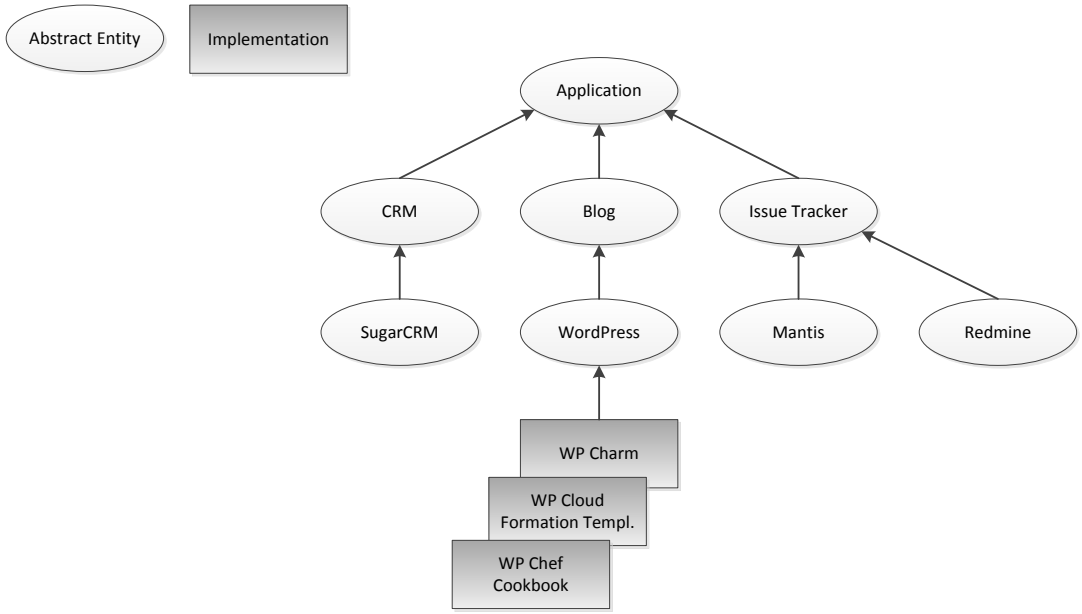


**Abbildung 3.2.:** Taxonomie der Anwendungen

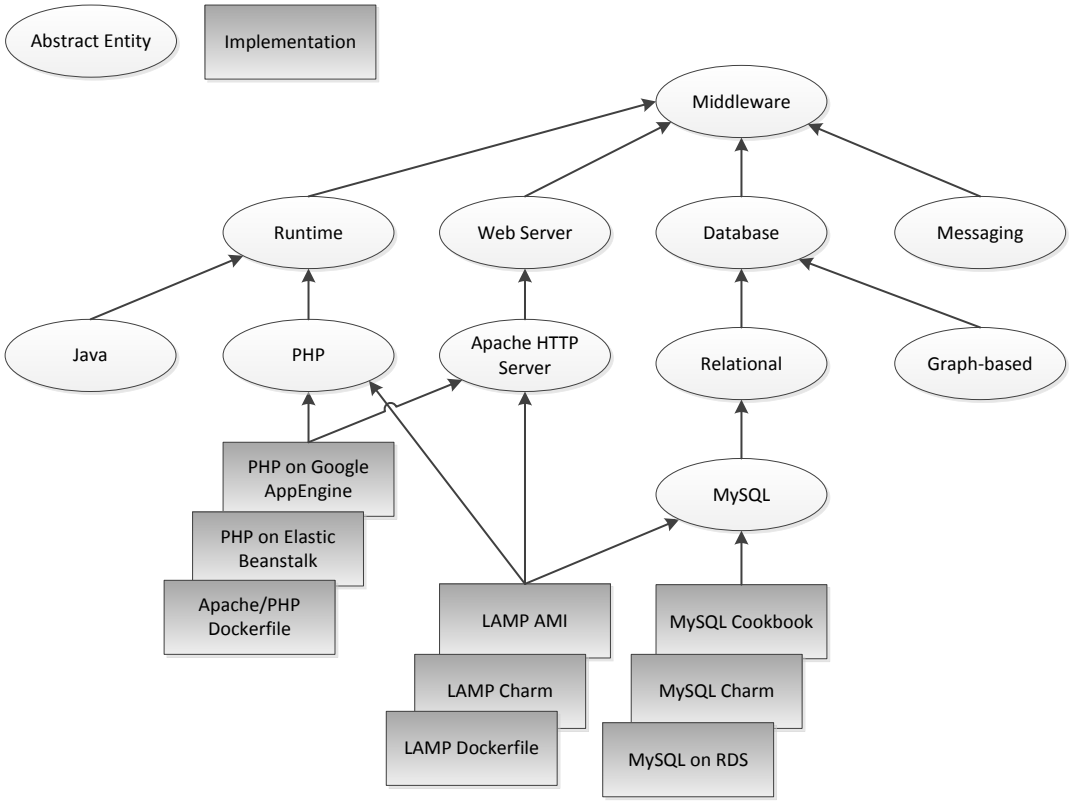


**Abbildung 3.3.:** Taxonomie der Dienstschicht

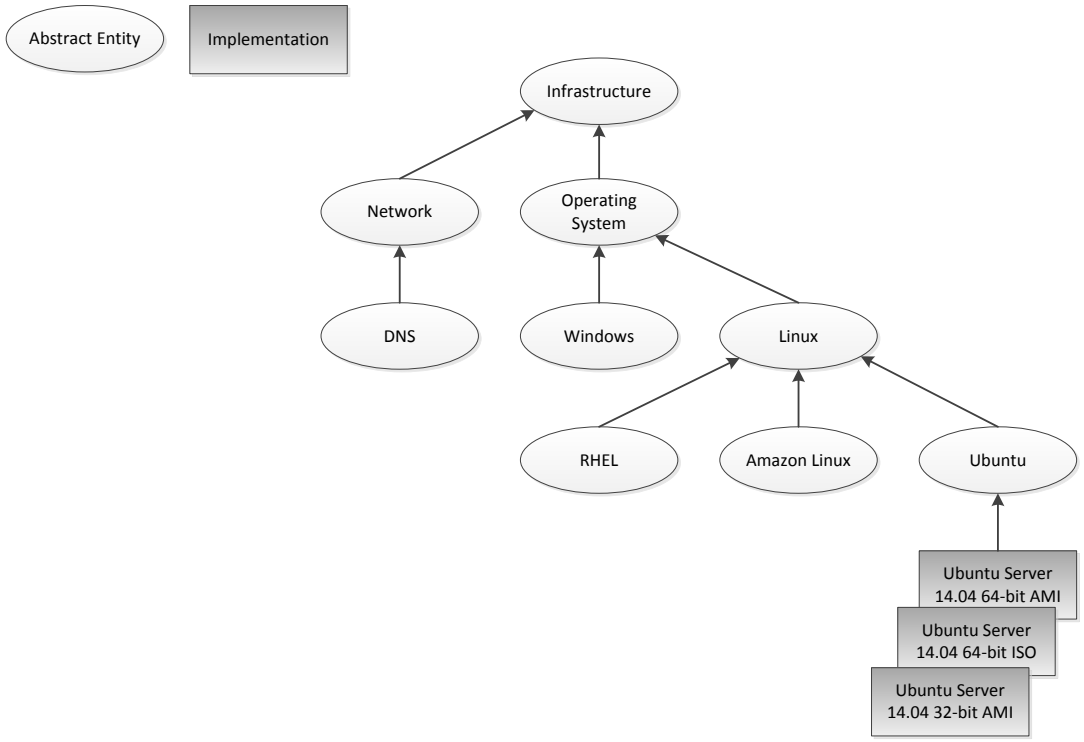


**Abbildung 3.4.:** Taxonomie der Infrastruktur

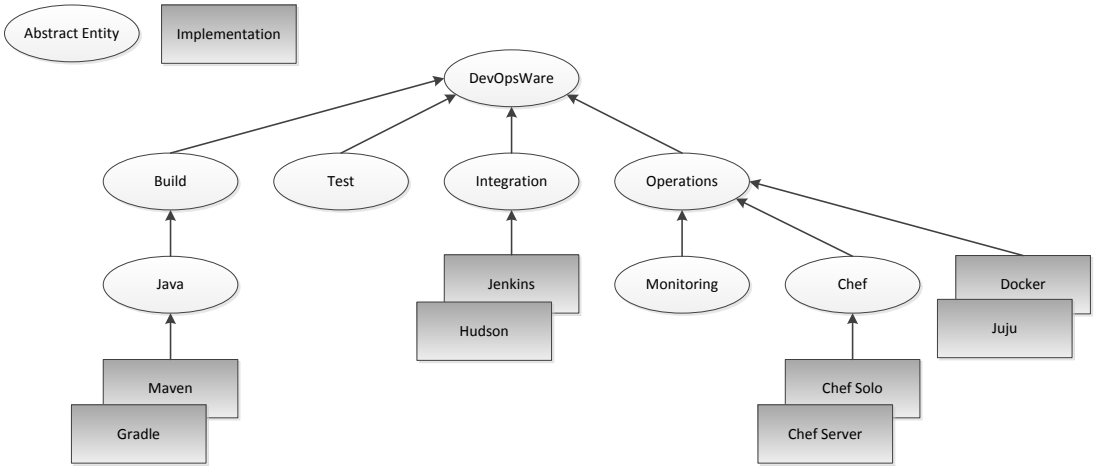


**Abbildung 3.5.:** Taxonomie der Werkzeug

## 3.3. DevOps-Spezifikation

Eine Eingabedatei für ein DevOps-Werkzeug wird im Folgenden als DevOps-Spezifikation bezeichnet. Für eine generalisierte Betrachtung solch einer Spezifikation wird an dieser Stelle eine Struktur vorgestellt, welche in dieser Arbeit verwendet wird. Das ER-Diagramm in Abbildung 3.6 zeigt den strukturellen Aufbau dieser abstrakten Spezifikation. Jede Spezifikation besitzt die *Attribute*

- Name
- Version
- Autor und
- Beschreibung

Die Attribute beinhalten Metadaten für die jeweilige Spezifikation, so dass explizit ersichtlich ist, wie die Spezifikation benannt ist, wozu genau sie dient, wer sie erstellt hat und welcher Versionsstand vorliegt. Der Name bezeichnet eine Spezifikation eindeutig.

*Operationen* in einer DevOps-Spezifikation kategorisieren unterschiedliche Schritte, die für eine Installation von Software durch die Spezifikation notwendig sind. Als Beispiele hierfür sind zu nennen

- erstellen (build)
- testen (test)
- starten (start)
- stoppen (stop)
- verteilen (deploy)
- verteilen beenden (undeploy)

Jede Operation enthält beliebig viele *Aktionen* sowie beliebig viele *Abhängigkeiten*. Aktionen sind Teiloperationen, durch dessen Ausführungen insgesamt die gewünschten Änderungen am System vorgenommen werden. Dies können z.B. einfache Shell-Kommandos oder Skripte sein, die ausgeführt werden. Genauso können aber auch fertige *Puppet Manifests*, *Chef Cookbooks* oder beliebige andere Operationen aus der vorliegenden DevOps-Spezifikation ausgeführt werden. Eine Aktion besitzt die Attribute *runner*, *config* und *comment*. Das Attribut runner ist erforderlich und bezeichnet die Technologie, mit der die Aktion ausgeführt wird. Das Attribut config beinhaltet die Nutzdaten der entsprechenden Aktion, also z.B. die tatsächlichen Shell-Kommandos, das Skript, Manifest oder Cookbook. Das optionale Attribut comment beinhaltet zusätzliche Kommentare. [Tiw00]

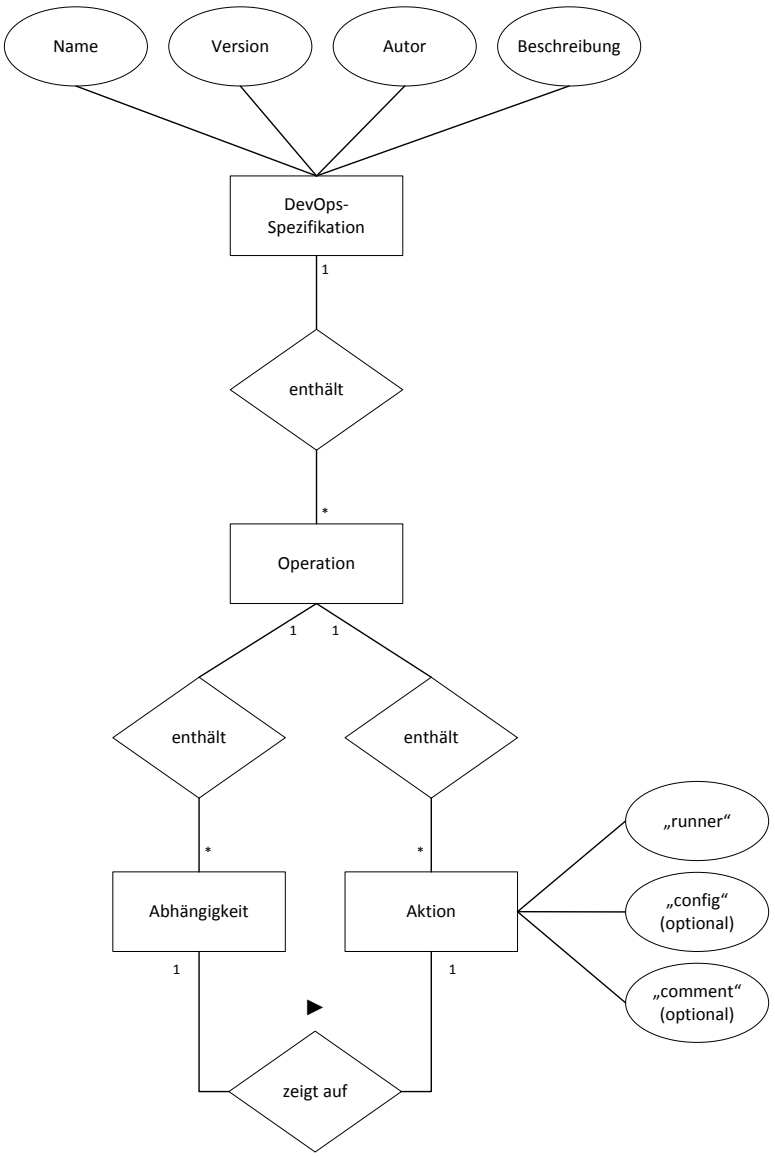


**Abbildung 3.6.:** ER-Diagramm einer DevOps-Spezifikation

## 3.4. Beispielszenario

Für eine konsistente Betrachtung wird an dieser Stelle ein Szenario eingeführt, welches im Folgenden als Beispiel dienen soll. Kern des Szenarios ist eine Installation des *Bugtracking Systems* Mantis [man]. Hierbei handelt es sich um eine Webanwendung, veröffentlicht unter der *GNU General Public License* [gpl], die für die Verwaltung von Fehlern oder als *Ticketing System* verwendet werden kann. Mantis basiert auf PHP [php] und benötigt neben einem Webserver auch eine Datenbank. Als Datenbank wird später eine MySQL-Datenbank gewählt und als Webserver ein *Apache Webserver* [apad]. Der Webserver und die Datenbank werden in (möglicherweise unterschiedlichen) Betriebssystemen ausgeführt, welche wiederum in virtuellen Maschinen laufen. Abbildung 3.7 veranschaulicht die Topologie des beispielhaft ausgewählten Szenarios.

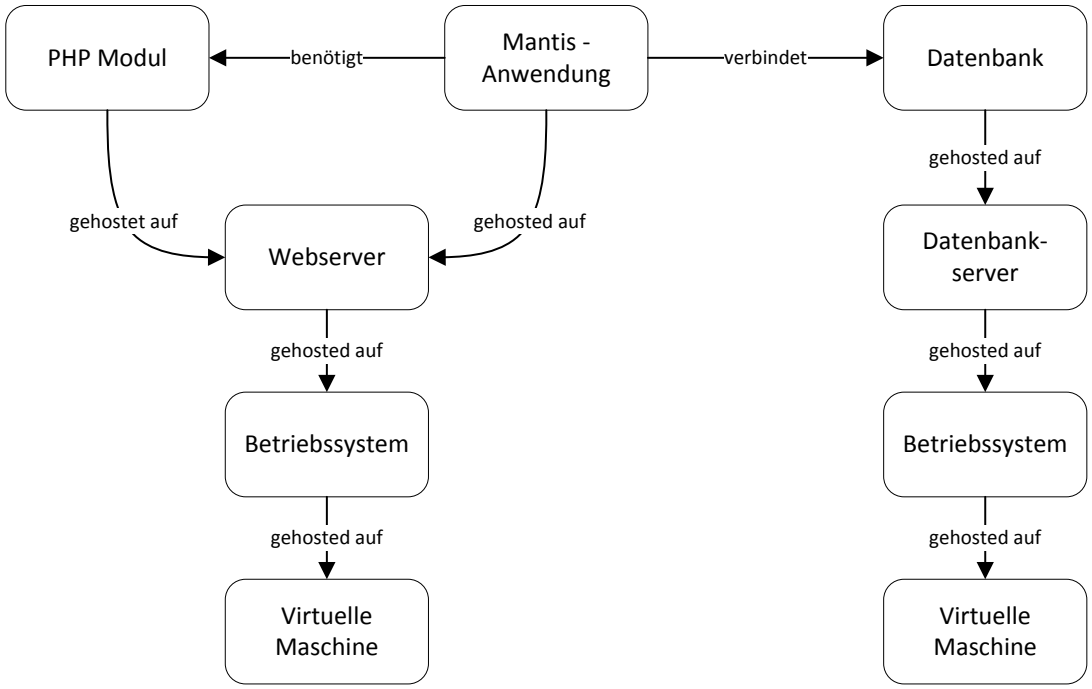


**Abbildung 3.7.:** Topologie des Beipielszenarios

Listings 3.2 und 3.3 zeigen eine unvollständige DevOps-Spezifikation entsprechend der in Abschnitt 3.3 vorgestellten Struktur, mit dem ein Deployment des Beispielszenarios durchgeführt werden kann. Die Spezifikation mit dem Namen *mantis* enthält die Operationen *build*, *start* und *deploy*. Die build-Operation wird mittels eines Shell-Kommandos ausgeführt, das über den Befehl *git clone* und einer *URL* auf ein *Git-Repository* für Mantis die Anwendung auf das Zielsystem kopiert. Die start-Operation enthält die beiden Teiloperationen *mantis* (Starten der Mantis-Anwendung) und *mysql* (Starten des MySQL-Service). Zudem ist unter *dependencies* (Abhängigkeiten) durch ein Tupel angegeben, dass

**Listing 3.2** DevOps-Spezifikation für das Beispielszenario (Teil 1)

```
{
   "name": "mantis",
   "version": "...",
   "author": "...",
   "description": "...",

   "operations": {
      "build": {
         "artifacts": {
            "get-app": {
               "runner": "command-runner",
               "comment": "git must be installed",
               "config": {
                  "command": "git clone https://github.com/mantisbt/mantisbt.git"
               }
            },
         },
      },
      "start": {
         "artifacts": {
            "mantis": {
               "runner": "command-runner",
               "config": {
                  "command": "sudo apt-get install mantis",
                  "cwd": "./mantis"
               }
            },
            "mysql": {
               "runner": "command-runner",
               "config": {
                  "command": "service mysql start"
               }
            }
         },
         "dependencies": [
            [ "mantis", "mysql" ]
         ]
      },
      "deploy": {
         "artifacts": {
            "deploy-git": {
               "runner": "chef-solo-runner",
               "config": {
                  "files": { "git.tar.gz":
                                    "http://s3.amazonaws.com/community-files.opscode.com/
                                    cookbook_versions/tarballs/6224/original/
                                    git20140318-2456-184qsad.?1395184516" },
                  "runlist": [ "recipe[git::default]" ]
               }
            },
```

**Listing 3.3** DevOps-Spezifikation für das Beispielszenario (Teil 2)

```
            "deploy-mysql": {
               "runner": "Chef Cookbook Deployer",
               "config": {
                  "name": "MySQL Server 5.x Chef Cookbook",
                                 "hosted_on_one_of": "Infrastructure/OS/Linux/Ubuntu",
                                 "version": "5.0"
               },
               "mergeconfig": false
            },
            "build-app": {
               "runner": "operation-runner",
               "config": {
                  "operation": "build"
               }
            },
            "start-app": {
               "runner": "operation-runner",
               "config": {
                  "operation": "start"
               }
            }
         },
         "sequence": [
            "deploy-git",
            "deploy-mysql",
            "build-app",
            "start-app"
         ]
      }
   }
}
```

die Teiloperation mantis von der Teiloperation mysql abhängt. Die deploy-Operation enthält die Teiloperationen *deploy-git* (Installation von Git mit Hilfe eines Chef Cookbooks und dem *chef-solo-runner*), *deploy-mysql* (Installation von MySQL), *build-app* (ruft die build-Operation auf) und *start-app* (ruft die start-app-Operation auf). Für die deploy-Operation ist eine Sequenz (*sequence*) angegeben, in der die einzelnen Teiloperationen abzuarbeiten sind. Die Teiloperation build-app erfordert z.B., dass auf dem Zielsystem eine MySQL-Datenbank installiert ist, und ist daher in der Sequenz nach der Teiloperation deploy-mysql angegeben.

## 3.5. Anforderungen

Die vorliegende Arbeit stellt Konzepte vor, die das Erstellen von DevOps-Spezifikationen für Softwareentwickler vereinfachen. Im Idealfall sollen Entwickler in die Lage versetzt werden, selbstständig und ohne direkte Unterstützung von Personal des Softwarebetriebs DevOps-Spezifikationen für die von ihnen entwickelte Software zu erstellen. Hierbei müssen neben den Anforderungen, die

seitens der Entwickler an die Anwendung und deren Umgebung gestellt werden, insbesondere globale Einschränkungen berücksichtigt werden, die seitens des Betriebs vorliegen. Einschränkungen können hierbei zum Beispiel spezielle Typen von Datenbankservern, Softwareversionen oder Konfigurationen der Betriebssysteme, auf denen die Anwendungen gehostet werden, sein. Bei diesen Einschränkungen handelt es sich um Fachwissen aus Abteilungen, die für den Softwarebetrieb zuständig sind. Entwickler haben dieses Fachwissen a priori nicht, so dass ohne weitere konzeptionelle Unterstützung eine manuelle und zeitaufwendige gemeinsame Erarbeitung bzw. ein Abgleich der Anforderungen und Einschränkungen notwendig ist. Ohne explizite Zusammenarbeit mit dem Betrieb sind Entwickler bei einem traditionellen Vorgehen also nicht in der Lage, selbstständig und kurzfristig DevOps-Spezifikationen zu erstellen. Dieses Problem soll im Folgenden durch die vorgestellten Konzepte adressiert werden. Ohne bei Entwicklern weitreichende Kenntnisse aus den Bereichen IT-Verwaltung und des IT-Betriebs vorauszusetzen, soll es für sie erleichtert werden, vollständige Deployment-Szenarien für Anwendungen eigenständig zu erstellen. Das Verfahren soll durch Automatisierung die Vorbereitung verschiedener Phasen des Softwarebetriebs verkürzen und Fehler eliminieren, die bei einer manuellen Ausführung leicht entstehen können. Hierfür müssen zum einen globale Einschränkungen (z.B. betriebliche Vorgaben für den Softwarebetrieb wie z.B. *not hosted on Amazon Web Services*) definiert werden und zum anderen Anforderungen, die eine Aktion einer DevOps-Spezifikation an die Umgebung stellt, formuliert werden können. Die Formulierung von Anforderungen und Einschränkungen muss hierbei auf einem hohen Abstraktionslevel erfolgen, so dass keine IT-Detailkenntnisse erforderlich sind. Ebenso muss die Formulierung strukturiert erfolgen können, damit eine automatisierte Auswertung ermöglicht wird. Eine Wissensdatenbank, wie in Abschnitt 3.2 vorgestellt, und eine DevOps-Spezifikation, wie in Abschnitt 3.3 definiert, stehen als Hilfsmittel zur Verfügung.

An die in Abschnitt 3.6 vorgestellten Konzepte für die Vorbereitung des Softwarebetriebs aus Entwicklersicht werden folgende Anforderungen gestellt:

**A1** Entwickler sollen bei der Erstellung von DevOps-Spezifikationen unterstützt werden. (Aktuelle Situation: Entwickler müssen DevOps-Spezifikationen manuell erstellen.)

**A2** Zu einer DevOps-Spezifikation sollen Anforderungen definiert werden können. (Aktuelle Situation: In einer DevOps-Spezifikation können Anforderungen nicht allgemein definiert werden. Z.B. ist es nicht möglich festzulegen, dass eine verwendete Software einen bestimmten Versionsstand hat.)

**A3** Es soll eine Datenbasis mit globalen Einschränkungen genutzt werden können. (Aktuelle Situation: Es existiert keine Datenbasis mit globalen Einschränkungen.)

**A4** Anforderungen und Einschränkungen sollen anhand einer vorgegebenen Struktur systematisch formuliert werden können. (Aktuelle Situation: Anforderungen und Einschränkungen werden informell ausgedrückt, z.B. durch betriebliche Anweisungen, Dokumentationen oder Standardvorgehensweisen und sind insbesondere nicht automatisiert auswertbar.)

**A5** Anforderungen und Einschränkungen sollen auf hohem Abstraktionslevel und in der Domäne der Entwickler formuliert und automatisiert ausgewertet werden können. (Aktuelle Situation: Vorliegende Anforderungen und Einschränkungen werden durch entsprechende manuelle Auswahl und Konfiguration von Implementierungen aus der Wissensdatenbank umgesetzt.)

**A6** Insbesondere auch Verbindungen von Implementierungen untereinander sollen beschrieben werden können. (Aktuelle Situation: Es lassen sich keine Abhängigkeiten zwischen verschiedenen Implementierungen ausdrücken.)

**A7** Die Ermittlung von passenden Implementierungen aus der Wissensdatenbank soll automatisiert erfolgen. (Aktuelle Situation: Die Ermittlung passender Implementierungen erfolgt manuell.)

**A8** Die Übernahme einer Implementierung aus der Taxonomie in der DevOps-Spezifikation soll automatisiert (bzw. halbautomatisiert) erfolgen können. (Aktuelle Situation: Die Übernahme einer Implementierung erfolgt manuell.)

## 3.6. Architektur

In diesem Abschnitt werden Konzepte vorgestellt, mit denen unter Beachtung der im vorangegangenen Abschnitt definierten Anforderungen der Softwarebetrieb aus Entwicklersicht vorbereitet werden kann. Zunächst wird in Unterabschnitt 3.6.1 gezeigt, wie Aktionen aus der in Abschnitt 3.3 vorgestellten DevOps-Spezifikation um zwei Attribute erweitert werden, um diese mit zusätzlichen Informationen anreichern zu können. Im Anschluss wird in Unterabschnitt 3.6.2 eine Menge von Prädikaten definiert, mit deren Hilfe Anforderungen, Einschränkungen und Abhängigkeiten von Implementierungen, die in der Wissensdatenbank (siehe Abschnitt 3.2) topologisch abgelegt sind, beschrieben werden können. In Unterabschnitt 3.6.3 wird sodann der Programmablauf erläutert und in Unterabschnitt 3.6.4 wird erklärt, wie der Algorithmus für die Auswahl von Implementierungen aus der Wissensdatenbank arbeitet. Abschließend wird in Unterabschnitt 3.6.5 beschrieben, wie die in Abschnitt 3.5 aufgestellten Anforderungen erfüllt werden.

### 3.6.1. Modifikation der DevOps-Spezifikation

Für die Formulierung von Anforderungen und Einschränkungen für das automatisierte Deployment von Anwendungen wird die DevOps-Spezifikation auf Aktionsebene um das Attribut *requirements* erweitert. Hier werden alle zu beachtenden Anforderungen hinterlegt, wobei diese so formuliert werden, dass im weiteren Verlauf eine automatisierte Auswertung möglich ist. Zusätzlich wird das weitere Attribut *resolver* eingeführt, welches beschreibt, von welchem Typ das Attribut *Anforderung* ist. Dadurch wird Flexibilität dahingehend erreicht, dass innerhalb einer DevOps-Spezifikation verschiedene Sprachen für die Beschreibung der Anforderungen verwendet werden können. Abbildung 3.8 zeigt das ER-Diagramm der erweiterten DevOps-Spezifikation.

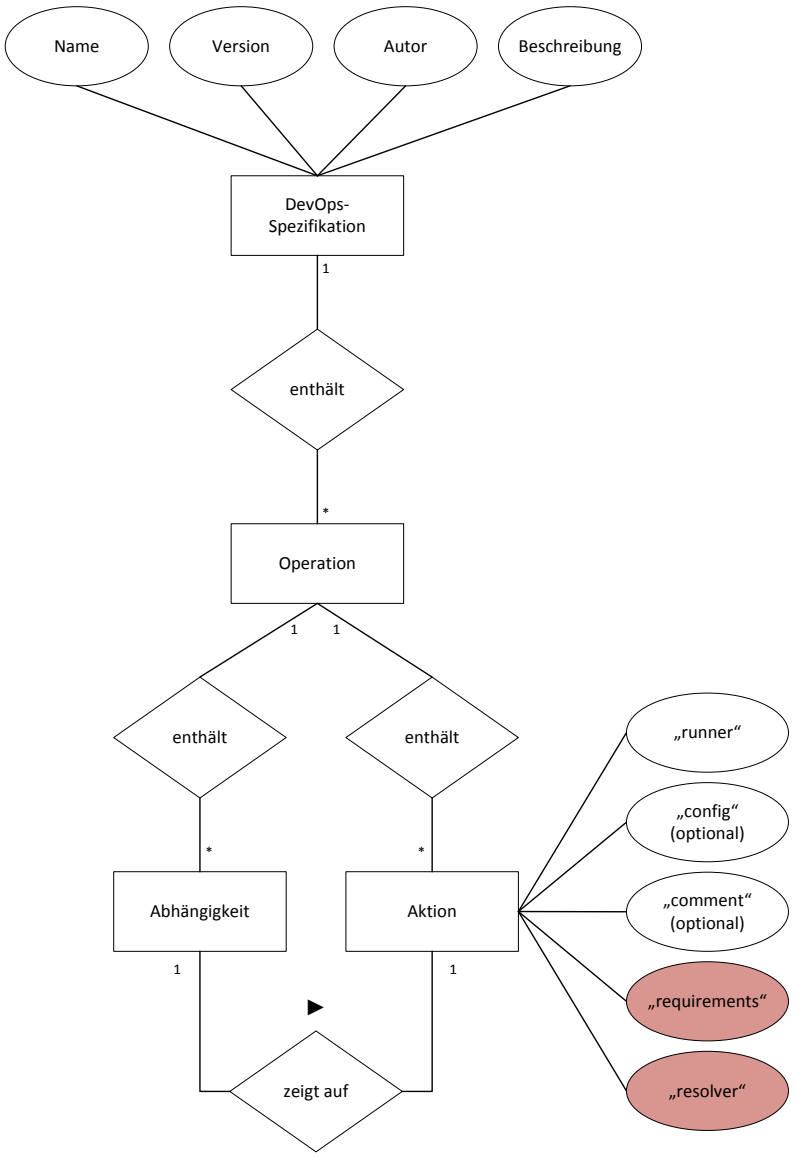


**Abbildung 3.8.:** ER-Diagramm einer DevOps-Spezifikation (erweitert)

### 3.6.2. Prädikate

Anforderungen und Einschränkungen lassen sich in der Aussagenlogik als Aussagen auffassen, denen jeweils genau ein Wahrheitswert (*TRUE* oder *FALSE*) zugeordnet wird. Atomare Aussagen lassen sich hierbei durch Junktoren (z.B. *Konjunktion*, *Disjunktion* oder *Negation*) miteinander verknüpfen. Eine Erweiterung der Aussagenlogik stellt die Prädikatenlogik dar, die zusätzlich *Quantoren*, *Funktions-*

*symbole* und *Prädikatssymbole* verwendet. Mit Hilfe der Prädikatenlogik lassen sich Sachverhalte beschreiben, die durch die Aussagenlogik alleine nicht beschreibbar sind. So lassen sich durch geeignete Prädikate z.B. bestimmte Beziehungen von Objekten untereinander ausdrücken oder es lässt sich feststellen, ob ein Objekt mit einer gewissen Eigenschaft existiert. [Sch00]

Im Folgenden wird eine Menge von Prädikaten eingeführt, um Anforderungen und Einschränkungen auszudrücken. Es wird zwischen vier Typen von Prädikaten unterschieden:

1. Prädikate, die Entitäten referenzieren und einen Parameter besitzen
2. Prädikate, die die Existenz von Eigenschaften prüfen und zwei Parameter besitzen
3. Prädikate, die den Wert einer Eigenschaft von einer Entität beschreiben und drei Parameter besitzen
4. Zusammengesetzte Prädikate

$E$ wird als die Menge aller Entitäten der Taxonomie der Wissensdatenbank (abstrakte Entitäten und Implementierungen) definiert, $P$ als die Menge aller Eigenschaften und $V$ als die Menge aller Werte, die die Eigenschaften annehmen können.

Die Namenskonvention für Prädikate wird wie folgt anhand des Beispiels $P_{fooBar}$ definiert: $foo$ gibt an, ob das Prädikat eine Entität (entity) oder eine Eigenschaft (property) einer Entität beschreibt und $Bar$ bezeichnet die Relation, die das Prädikat beschreibt (Eq (equals), Gr (greater), Le (less), Ne (not equal), Req (required), Exc (excluded) und Ex (exists)).

Ein Prädikat, das eine Entität beschreibt, besitzt immer einen Parameter (nämlich die Entität selbst). $P_{entityReq} : E \rightarrow \{TRUE, FALSE\}$ weist jedem Element aus $E$ hierbei einen booleschen Wert zu. So wird das Prädikat zu $TRUE$ ausgewertet, wenn es sich bei einer Entität um eine Implementierung handelt oder wenn von einer abstrakten Entität mindestens eine Implementierung abgeleitet wird. Für alle anderen Entitäten wird $P_{entityReq}$ zu $FALSE$ ausgewertet. Das Prädikat $P_{entityExc}$ stellt die Negation von $P_{entityReq}$ dar.

Ein Prädikat, das die Existenz einer Eigenschaft von einer Entität prüft, erfordert die zwei Parameter entity und property. $P_{propertyEx} : E \times P \rightarrow \{TRUE, FALSE\}$ weist jedem Element aus $E$ den Wert $TRUE$ zu, bei dem die ausgewählte Eigenschaft vorhanden ist. Für alle anderen Elemente wird das Prädikat zu $FALSE$ ausgewertet.

Ein Prädikat, das den Wert einer Eigenschaft von einer Entität beschreibt, erfordert die drei Parameter entity, property und value. $P_{valueBar} : E \times P \times V \rightarrow \{TRUE, FALSE\}$ weist jeder Kombination eines Elementes aus $E$ mit einer Eigenschaft und einem Wert einen booleschen Wert zu. Für alle Entitäten, welche die Eigenschaft besitzen und die angegebene Relation (Bar) erfüllen wird das Prädikat zu $TRUE$ ausgewertet. Andererseits findet eine Auswertung zu $FALSE$ statt.

Folgende Prädikate werden definiert:

- $P_{entityReq}$ (Entität notwendig)
- $P_{entityExc} = not\ P_{entityReq}$ (Entität nicht erlaubt)
- $P_{propertyEx}$ (Entität besitzt die angegebene Eigenschaft)

- $P_{valueEq}$ (Wert einer Eigenschaft von einer Entität ist gleich dem angegebenen Wert)
- $P_{valueNe}$ (Wert einer Eigenschaft von einer Entität ist nicht gleich dem angegebenem Wert)
- $P_{valueGr}$ (Wert einer Eigenschaft von einer Entität ist größer als der angegebene Wert)
- $P_{valueLe}$ (Wert einer Eigenschaft von einer Entität ist kleiner als der angegebene Wert)

Darüber hinaus werden zwei weitere, zusammengesetzte Prädikate definiert, um eine übersichtlichere Darstellung zu ermöglichen:

- $P_{valueEqGr} = P_{valueEq} \vee P_{valueGr}$ (größer oder gleich)
- $P_{valueEqLe} = P_{valueEq} \vee P_{valueLe}$ (kleiner oder gleich)

Mit Hilfe von Prädikaten lassen sich nun Eigenschaften ausdrücken, die für eine DevOps-Spezifikation gelten sollen. So könnte z.B. für das Deployment einer Mantis-Installation folgender Ausdruck verwendet werden:

$$Mantis = P_{valueEqGr}('Middleware/Database/Relational/MySQL','versions',' 5.0') \\ \wedge P_{valueEqGr}('Middleware/Runtime/PHP','versions',' 5.2.4') \\ \wedge P_{entityReq}('Middleware/WebServer')$$

Die hier vorgestellten Prädikate verstehen sich als eine erste, initiale Auswahl. Je nach Anforderungen können weitere Prädikate hinzugefügt oder bestehende entfernt werden.

### 3.6.3. Programmablauf

Abbildung 3.9 veranschaulicht den vorgeschlagenen Ablauf des Programms. Eine neue DevOps-Spezifikation dient als Haupteingabe des Verfahrens. Diese enthält entweder noch keine Implementierungen, oder die enthaltene Menge der Implementierungen ist noch unvollständig. Darüber hinaus enthält die Spezifikation auf Ebene der Aktionen Anforderungen, die erfüllt werden müssen. Sowohl Implementierungen als auch Anforderungen in der Spezifikation können bearbeitet werden, insbesondere das Löschen von vorhandenen Implementierungen und das Einfügen von neuen Anforderungen ist möglich. Neben einer Wissensdatenbank für Implementierungen, in der diese in Form einer Taxonomie strukturiert hinterlegt sind, existiert eine zweite Datenbasis mit globalen Einschränkungen. Anforderungen und Einschränkungen werden, damit eine automatisierte Auswertung erfolgen kann, mit Hilfe von Prädikaten ausgedrückt. In einem ersten Schritt werden die Anforderungen aus der Spezifikation mit den globalen Einschränkungen vereinigt. Anhand der resultierenden Prädikate aus dieser Vereinigung wird nun in einem nächsten Schritt nach einer Menge von Implementierungen in der Wissensdatenbank gesucht, die alle vorliegenden Prädikate erfüllen. Die Ergebnisse werden dem Benutzer präsentiert und es muss zwischen zwei Optionen entschieden werden:

1. eine gefundene Implementierung wird in die Spezifikation übernommen
2. eine neue Iteration wird gestartet (mit veränderten Parametern)

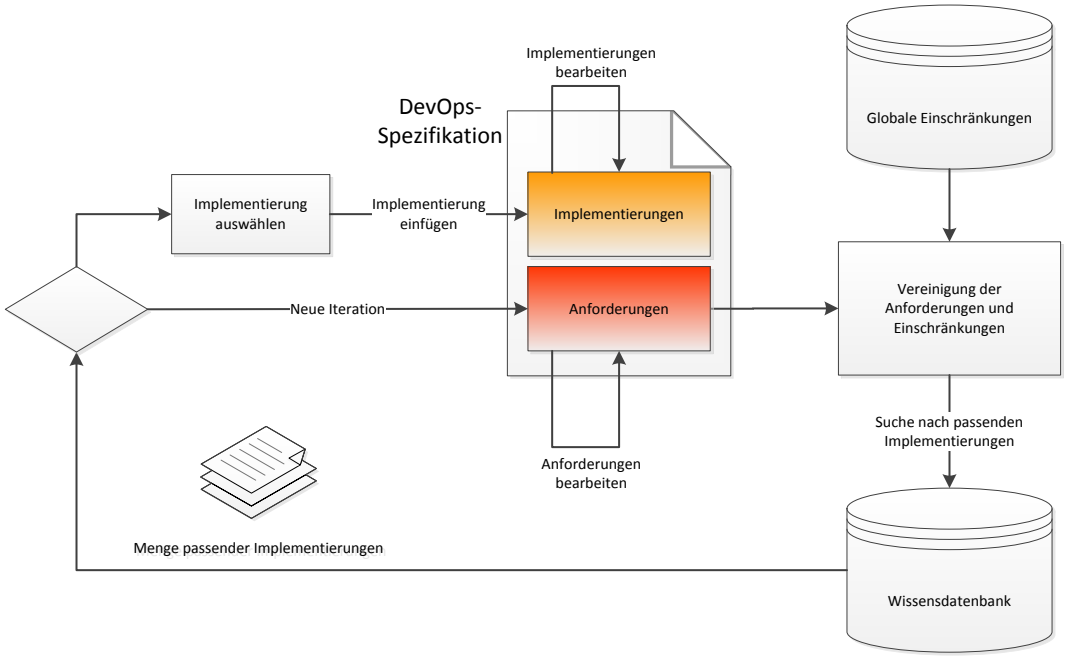


**Abbildung 3.9.:** Programmablauf der vorgeschlagenen Lösung

### 3.6.4. Auswahlalgorithmus

Anhand der WS-Policy, die als Eingabe für den Algorithmus dient, wird eine Menge von Implementierungen aus der Wissensdatenbank ermittelt, die für eine Übernahme in die DevOps-Spezifikation in Frage kommen. Hierbei wird für jede *policy alternative* aus der normalisierten WS-Policy eine Liste erstellt, die alle Referenzen auf Implementierungen enthält, die den in der Alternative enthaltenen Zusicherungen entsprechen. Dieses Vorgehen ist notwendig, da sich Zusicherungen in verschiedenen Alternativen einer WS-Policy widersprechen können und sich somit keine konsistente Menge von Implementierungen erzeugen lassen würde. Die nachfolgenden Schritte werden also in ihrer Gesamtheit separat für jede vorkommende Alternative der WS-Policy ausgeführt.

1. Ermittlung einer Menge aller Referenzen auf Implementierungen, für die die vorliegenden Zusicherungen *entityReq* in der entsprechenden Alternative zu *TRUE* ausgewertet werden
2. Hinzufügen aller Referenzen auf Implementierungen, die implizit in allen vorliegenden Zusicherungen gefordert werden (Ausnahme: *entityExc*)
3. Entfernen aller Referenzen auf Implementierungen, für die die Zusicherungen *entityExc* in der entsprechenden Alternative zu *TRUE* ausgewertet werden

4. Entfernen aller Referenzen auf Implementierungen, für die die Zusicherungen *propertyEx* in der entsprechenden Alternative zu *FALSE* ausgewertet werden
5. Entfernen aller Referenzen auf Implementierungen, für die die Zusicherungen *valueEq*, *valueNe*, *valueGr*, *valueLe*, *valueEqGr* und *valueEqLe* in der entsprechenden Alternative zu *FALSE* ausgewertet werden

Die ermittelten *i* Mengen von Referenzen auf Implementierungen der vorliegenden *i* Alternativen der WS-Policy werden nun zu einer resultierenden Menge vereinigt. Sodann kann aus dieser Menge eine Auswahl getroffen werden. Die ausgewählte Implementierung wird nachfolgend aus der Wissensdatenbank geholt und in die DevOps-Spezifikation übernommen.

### 3.6.5. Erfüllung der Anforderungen

Die in Abschnitt 3.5 definierten Anforderungen für die Vorbereitung des Softwarebetriebs aus Entwicklersicht werden durch die vorgeschlagenen Konzepte wie folgt erfüllt:

**A1** : Der Anwender wird bei der Erstellung von DevOps-Spezifikationen dahingehend unterstützt, dass die Erstellung der Spezifikation anhand von angegebenen Anforderungen automatisiert erfolgt. Somit wird die Erstellung von Spezifikationen erleichtert sowie beschleunigt.

**A2** : Zu einer DevOps-Spezifikation können durch die Erweiterung von Aktionen durch die beiden genannten Attribute Anforderungen definiert und ausgewertet werden. Diese können entweder direkt (*inline*) in der Spezifikation angegeben werden oder es kann eine Referenz auf ein externes Dokument hinterlegt werden.

**A3** : Wie in Abbildung 3.9 zu sehen ist, wird eine Datenbank eingeführt, in der eine Menge von globalen Einschränkungen abgelegt werden kann.

**A4** : Durch die Definition von Prädikaten (vgl. Unterabschnitt 3.6.2) können sowohl die Anforderungen als auch die Einschränkungen in einer strukturierten Art und Weise formuliert werden. Darüber hinaus ist eine automatisierte Auswertung dieser Prädikate in boolescher Logik möglich.

**A5** : Das für die Prädikate verwendete Abstraktionslevel ist frei wählbar. Vorgeschlagen werden Prädikate in Anlehnung an die in der Wissensdatenbank vorhandenen Attribute der Entitäten. Hierbei kann zum einen auf Existenz von Entitäten und / oder Attributen überprüft werden, zum anderen können Werte von Attributen ins Verhältnis zu einem übergebenen Referenzwert gesetzt werden. Die boolesche Auswertung der Prädikate und damit auch die entsprechend darauf ausgerichtete Formulierung derselben ist hierbei in der Domäne der Softwareentwicklung auszumachen.

**A6** : Durch die Verwendung von Prädikaten, die Werte von Eigenschaften von Entität beschreiben, können Verbindungen von verschiedenen Entitäten untereinander modelliert werden. So kann z.B. eine Entität als Alternative zu einer anderen definiert werden oder es kann beschrieben werden, dass eine Entität eine weitere Entität benötigt um zu funktionieren (wie in dem vorgestellten Beispielszenario, in dem eine Mantis-Installation das Vorhandensein einer MySQL-Installation voraussetzt).

**A7** : Die Ermittlung von passenden Implementierungen aus der Taxonomie erfolgt automatisiert. Hierfür ist im weiteren Verlauf eine Softwarekomponente zu entwerfen, welche die übergebenen Prädikate auswertet und entsprechend der Ergebnisse Implementierungen aus der Taxonomie in der Wissensdatenbank ermittelt und zurückgibt.

**A8** : Die Übernahme einer Implementierung aus der Wissensdatenbank in eine DevOps-Spezifikation erfolgt automatisiert (bzw. halbautomatisiert). Hierfür ist eine Softwarekomponente zu entwerfen, welche dem Benutzer passende Implementierungen anzeigt und auf Wunsch an den entsprechenden Stellen in der DevOps-Spezifikation einfügt.

# 4. Design und Implementierung

In diesem Kapitel wird eine prototypische Implementierung der im vorangegangenen Kapitel 3 vorgeschlagenen Konzepte entwickelt. Zunächst wird hierfür in Abschnitt 4.1 das *WS-Policy Framework* [VOH+07a] vorgestellt. In Abschnitt 4.2 werden Zusicherungen, die Teil des Frameworks sind, näher beschrieben. Für die automatisierte Verarbeitung solcher Zusicherungen wird in Abschnitt 4.3 die Software *Apache Neethi* [ws.] eingeführt. Danach wird in Abschnitt 4.4 das Design für die prototypische Implementierung der vorgeschlagenen Konzepte erarbeitet, bevor in Abschnitt 4.5 abschließend die Funktionsweise der Kommandozeilenschnittstelle erläutert wird.

Implementiert wird die Anwendung als Java-Projekt. Für die Unterstützung beim Build-Prozess wird auf das *Build-Management-Tool* [bmt] *Apache Maven* [mava] zurückgegriffen, mit dessen Hilfe folgende Abhängigkeiten aufgelöst und automatisch in das Projekt integriert werden:

- Apache Neethi für das Handling von Zusicherungen (siehe Abschnitt 4.3)
- JDOM [jdo] für das Handling von XML-Dokumenten
- JSON.simple [jsoa] für das Handling von JSON-Dokumenten [jsob]
- Maven Artifact [mavb] für das Vergleichen von Versionsnummern
- GSON [gso] für die Formatierung von JSON-Dokumenten

## 4.1. WS-Policy Framework

Die Verwendung des *WS-Policy Frameworks* [VOH+07a] als Ausdrucksmittel für die in Kapitel 3 eingeführten Prädikate ermöglicht eine standardisierte und interoperable Spezifizierung von nichtfunktionalen Eigenschaften und Anforderungen in Form von Richtlinien, die mit Web Services assoziiert werden können. Diese Richtlinien sind insbesondere zusammensetzbar und wiederverwendbar, so dass anstelle einer einzigen, monolithischen Beschreibung des Systems viele Teilbeschreibungen verwendet werden können. Diese Teilbeschreibungen sind nicht auf die Verwendung in Kombination mit *Web Service Endpoints* [wsd] beschränkt, sondern sie können ebenso auch mit XML-Dokumenten oder beliebigen zustandsbehafteten Ressourcen assoziiert werden. WS-Policy ist explizit darauf ausgelegt, erweiterbar zu sein und Richtlinien für unterschiedlichste Disziplinen und Domänen ausdrücken zu können. Mit *WS-Policy Attachment* [wsp] besteht zudem die Möglichkeit, Mengen von assoziierten WS-Policies flexibel zu verändern (Hinzufügen oder Entfernen von WS-Policies). [WCL+05]

Das *WS-Policy Framework* definiert ein Framework und Modell, mit dessen Hilfe sich Richtlinien (*policies*) für domänenspezifische Fähigkeiten, Anforderungen und Charakteristika für Systeme, die aus *Web Services* bestehen, ausdrücken lassen. Servicenutzer und Serviceanbieter haben so die Möglichkeit,

verschiedene Eigenschaften der entsprechenden Dienste in XML zu formulieren bzw. (automatisiert) auszuwerten und somit gewisse Richtlinien für den entsprechenden Dienst sicherzustellen. Eine policy besteht hierbei aus einer Menge von Alternativen (*policy alternatives*), welche wiederum aus Mengen von Zusicherungen (*policy assertions*) und geschachtelten policies bestehen. Eine Zusicherung beschreibt eine Fähigkeit, eine Anforderung oder eine andere Verhaltenseigenschaft. Sie bezieht sich entweder auf die zwischen Web Services ausgetauschten Nachrichten (*wire manifestation*) oder auf die Auswahl und die Verwendung der Web Services selbst (*no wire manifestation*). Die Darstellung einer policy als *XML Infoset* wird *policy expression* genannt.

WS-Policy definiert eine einfache Sprache bestehend aus den vier Elementen *wsp:Policy*, *wsp:All*, *wsp:ExactlyOne* und *wsp:PolicyReference* sowie den zwei Attributen *wsp:Optional* und *wsp:Ignorable*. Mit *wsp:Policy* wird eine policy ausgezeichnet. *wsp:All* zeichnet eine Menge von Zusicherungen aus, die alle gelten müssen, wohingegen *wsp:ExactlyOne* eine Menge von Zusicherungen auszeichnet, von denen genau eine gelten muss. Mit Hilfe von *wsp:PolicyReference* wird eine policy expression referenziert. *wsp:Optional* bedeutet, dass die entsprechende Zusicherung nur optional ist und *wsp:Ignorable* bedeutet, dass sie ignoriert werden darf. Mit Hilfe von WS-Policy lassen sich also über Mengen von Zusicherungen Richtlinien erstellen, in denen die Zusicherungen aussagenlogisch miteinander verknüpft sind. Die einzige Ausnahme hiervon ist die *Negation* von Ausdrücken, welche mit Sprachmitteln von WS-Policy nicht erreicht werden kann. Zu diesem Zweck wurden in der vorliegenden Arbeit für Prädikate (aus denen die Zusicherungen abgeleitet werden), die negierbar sein müssen, weitere Prädikate eingeführt, die jeweils die Negation darstellen (z.B. ist *entityExc* die Negation von *entityReq*).

Policies können in *Kompaktform* oder *Normalform* vorliegen. In der Kompaktform können alle genannten Elemente und Attribute der Sprache verwendet werden. Sie ist übersichtlicher als die Normalform, jedoch aufgrund der Unbestimmtheit des strukturellen Aufbaus schlecht für eine automatisierte Auswertung geeignet. Die Normalform hingegen besitzt einen festgelegten strukturellen Aufbau. Hier umschließt genau ein *ExactlyOne*-Operator einen oder mehrere *All*-Operatoren. Die Attribute *wsp:Optional* und *wsp:Ignorable* kommen nicht vor. Eine policy in Normalform ist im Vergleich zu einer äquivalenten policy in Kompaktform umfangreicher und unübersichtlicher, jedoch lässt sie sich gut automatisiert auswerten. Eine policy in Kompaktform kann algorithmisch in Normalform umgeformt werden. [VOH+07a]

Zur Veranschaulichung des Aufbaus einer policy in Normalform dient Abbildung 4.1. Dargestellt ist eine policy, die zwei Alternativen beinhaltet. Die erste Alternative besteht aus den Zusicherungen *AssertionA* und *AssertionB*, welche bei Auswahl dieser Alternative beide erfüllt sein müssen. Die zweite Alternative beinhaltet die Zusicherungen *AssertionA* und *AssertionC*, welche bei Auswahl dieser Alternative beide erfüllt sein müssen. [VOH+07b]

*Policy discovery*, *policy scope*, *policy subjects* und Mechanismen für *policy attachment* sind nicht Bestandteil des WS-Policy Frameworks. [VOH+07a]

Das in Abschnitt 4.4 vorgeschlagene Design nutzt nur einen Teil der Funktionalität des WS-Policy Frameworks. Hierbei findet kein Matching von Serviceanbieter und Servicenutzer anhand der als WS-Policies formulierten Richtlinien statt, wodurch die Notwendigkeit entfällt, *policy intersection* durchzuführen. Stattdessen wird die durch *policy model* und *policy expression* definierte Sprache verwendet, um Anforderungen und Einschränkungen auszudrücken und diese mit Entitäten zu

assoziieren. Insbesondere die Verwendung der Kompaktform ermöglicht hierbei eine komfortable Formulierung von Richtlinien. Diese werden nun mittels Vereinigung zu einer resultierenden Richtlinie zusammengeführt und anschließend in die leichter automatisiert auswertbare Normalform überführt.

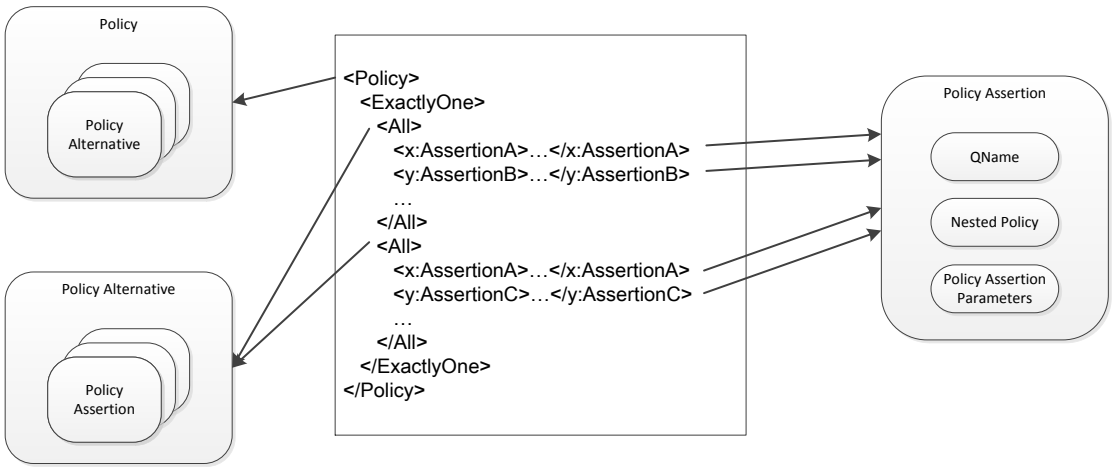


**Abbildung 4.1.:** WS-Policy in Normalform [VOH+07b]

## 4.2. Policy Assertions

Entsprechend der in Kapitel 3 eingeführten Prädikate werden im Folgenden drei verschiedene Arten von WS-Policy assertions definiert.

Abbildung 4.2 zeigt den schematischen Aufbau einer assertion. Neben einem erforderlichen *QName* [xml] kann eine Zusicherung optional geschachtelte policies sowie eine Menge von Parametern enthalten. Zur Veranschaulichung ist eine policy expression dargestellt, welche die *sp:IssuedToked Policy Assertion* verwendet. Diese hat die drei Parameter *@sp:IncludeToken*, *sp:Issuer* und *sp:RequestSecurityTokenTemplate*. Die Parameter sind bei der Verarbeitung von policies transparent und werden insbesondere durch die Operationen Normalisieren (*normalize*), Vereinigen (*merge*) und Schneiden (*intersect*) erhalten. [VOH+07b]

Listing 4.2 am Ende dieses Abschnitts zeigt die XML Schema Definitionen für WS-Policy assertions, die von den vorliegenden Prädikaten abgeleitet wurden. Die beiden assertions *entityReq* und *entityExc* referenzieren Entitäten und besitzen einen Parameter. Mit Hilfe der assertion *entityReq* wird sichergestellt, dass die im Attribut *entity* angegebene Entität vorhanden ist. Mit Hilfe der assertion *entityExc* wird sichergestellt, dass die im Attribut *entity* angegebene Entität nicht vorhanden ist (Negation von *entityReq*). Die assertion *propertyEx* prüft die Existenz einer Eigenschaft einer Entität und besitzt zwei Parameter. Mit Hilfe von *propertyEx* wird geprüft, ob die im Attribut *entity* angegebene Entität den im Attribut *property* angegebene Eigenschaft besitzt. Die vier assertions *valueEq*, *valueNe*,

*valueGr* und *valueLe* beschreiben den Wert einer Eigenschaft von einer Entität und besitzten drei Parameter. Mit *valueEq* wird für die in Attribut *entity* übergebene Entität geprüft, ob die im Attribut *property* angegebene Eigenschaft dem im Attribut *value* übergebenen Wert entspricht. *valueNe* stellt die Negation von *valueEq* dar. *valueGr* prüft für die in Attribut *entity* übergebene Entität, ob die im Attribut *property* angegebene Eigenschaft größer ist als der im Attribut *value* übergebenen Wert. *valueLe* prüft für die in Attribut *entity* übergebene Entität, ob die im Attribut *property* angegebene Eigenschaft kleiner ist als der im Attribut *value* übergebenen Wert. Die beiden assertions *valueEqGr* und *valueEqLe* prüfen, ob der Wert einer Eigenschaft einer Entität *größer gleich* bzw. *kleiner gleich* ist als der Attribut *value* übergebene Wert. Mit Ausnahme der Äquivalenz werden die Vergleichsoperatoren mittels *Maven Artifact* ausgewertet. So können neben einfachen Ganzzahlausdrücken auch Versionsnummern, die entsprechend der *Semantic Versioning Specification* (SemVer) [sem] aufgebaut sind, miteinander verglichen werden.

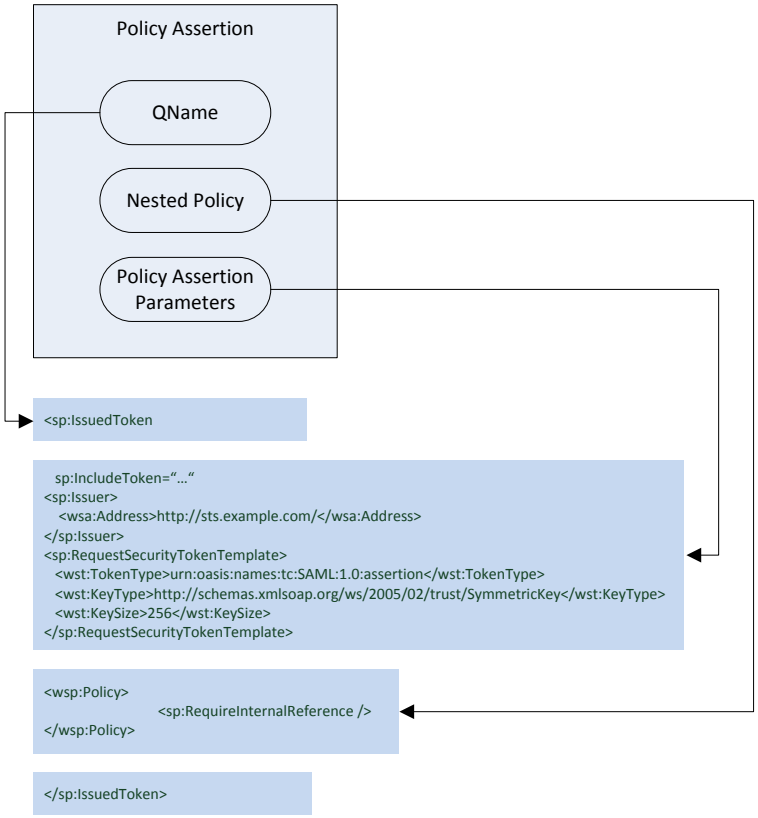


**Abbildung 4.2.:** WS-Policy Assertion [VOH+07b]

Mit Hilfe der definierten assertions lassen sich nun policies erstellen, die für Implementierungen gelten sollen, die aus der Wissensdatenbank ermittelt werden. Die policies können hierbei eine Vielzahl

von assertions enthalten, die ggf. mit Sprachmitteln von WS-Policy aussagenlogisch miteinander verknüpft sind. Für die automatisierte Auswertung der policies wird, wie im folgenden Abschnitt 4.3 ausgeführt wird, *Apache Neethi* verwendet, wobei sowohl policies in Normalform als auch policies in Kompaktform verarbeitet werden können. Listing 4.1 zeigt ein Beispiel für eine policy in Normalform zur Auswahl von Entitäten aus der Wissensdatenbank. Anhand dieser policy wird die Menge aller Linux-Implementierungen in x86-Architektur zurückgegeben, ausgenommen jedoch Amazon Linux AMIs [amab] (Alternative 1) oder es wird ein bestimmtes Ubuntu-Server Amazon Linux AMI in x64-Architektur zurückgeben (Alternative 2).

**Listing 4.1** Beispiel für eine WS-Policy zur Auswahl von Entitäten aus der Wissensdatenbank

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <wsp:ExactlyOne>
            <wsp:All>
                  <entityReq entity="Infrastructure/Operating System/Linux" />
                  <propertyEq entity="Infrastructure/Operating System" property="arch"
                       value="x86" />
                  <entityExc entity="Infrastructure/Operating System/Linux/Amazon Linux" />
            </wsp:All>
            <wsp:All>
                  <entityReq entity="Infrastructure/Operating System/Linux/Ubuntu/Ubuntu Server
                       14.04 AMI PV 64-bit US East" />
            </wsp:All>
      </wsp:ExactlyOne>
</wsp:Policy>
```

**Listing 4.2** XML Schema Definitionen für aus den Prädikaten abgeleitete WS-Policy assertions

```
<xsd:simpleType name="entityReq">
      <xs:attribute name="entity" type="xs:string"/>
</xsd:simpleType>

<xsd:simpleType name="entityExc">
      <xs:attribute name="entity" type="xs:string"/>
</xsd:simpleType>

<xsd:simpleType name="propertyEx">
      <xs:attribute name="entity" type="xs:string"/>
      <xs:attribute name="property" type="xs:string"/>
</xsd:simpleType>

<xsd:simpleType name="valueEq">
      <xs:attribute name="entity" type="xs:string"/>
      <xs:attribute name="property" type="xs:string"/>
      <xs:attribute name="value" type="xs:string"/>
</xsd:simpleType>

<xsd:simpleType name="valueNe">
      <xs:attribute name="entity" type="xs:string"/>
      <xs:attribute name="property" type="xs:string"/>
      <xs:attribute name="value" type="xs:string"/>
</xsd:simpleType>

<xsd:simpleType name="valueGr">
      <xs:attribute name="entity" type="xs:string"/>
      <xs:attribute name="property" type="xs:string"/>
      <xs:attribute name="value" type="xs:string"/>
</xsd:simpleType>

<xsd:simpleType name="valueLe">
      <xs:attribute name="entity" type="xs:string"/>
      <xs:attribute name="property" type="xs:string"/>
      <xs:attribute name="value" type="xs:string"/>
</xsd:simpleType>

<xsd:simpleType name="valueEqGr">
      <xs:attribute name="entity" type="xs:string"/>
      <xs:attribute name="property" type="xs:string"/>
      <xs:attribute name="value" type="xs:string"/>
</xsd:simpleType>

<xsd:simpleType name="valueEqLe">
      <xs:attribute name="entity" type="xs:string"/>
      <xs:attribute name="property" type="xs:string"/>
      <xs:attribute name="value" type="xs:string"/>
</xsd:simpleType>
```

## 4.3. Apache Neethi

Apache Neethi ist ein WS-Policy Framework, veröffentlicht unter der Apache 2.0-Lizenz, welches die aktuellste WS-Policy Framework-Spezifikation aus dem Jahr 2006 implementiert. Anforderungen und Fähigkeiten von Web Services können mit Hilfe einer komfortablen API formuliert und zur Laufzeit ausgewertet werden, wodurch die Verwendung von WS-Policy erleichtert wird. Domänenspezifische Typen können als assertions erweitert werden und assertions können benutzerdefiniert serialisiert bzw- deserialisiert werden. [ws.]

Mit Hilfe von Apache Neethi kann ein komfortables Handling von policies ausgeführt werden. Für Objekte vom Typ *Policy* stehen u.a. die Methoden *intersect()*, *merge()*, *normalize()* und *serialize()* zur Verfügung. Objekte, die das Interface *Assertion* implementieren, bieten ebenfalls Methoden für Normalisierung und Serialisierung sowie die beiden Attribute *optional* und *ignorable* an. Die Operatoren *All* und *ExactlyOne* stehen in Form von Klassen zur Verfügung und können für den Aufbau von policy-Alternativen verwendet werden. Darüber hinaus werden Klassen für die Konvertierung von *Streaming API for XML* (StAX) [sta] zu *Document Object Model* (DOM) [dom] und vice versa angeboten.

Im Folgenden wird nur ein Teil der Funktionalität, die das Apache Neethi Framework zur Verfügung stellt, verwendet. Anforderungen (formuliert als WS-Policy), die für eine Aktion einer DevOps-Spezifikation gelten, und vorliegende globale Einschränkungen (ebenfalls formuliert als WS-Policies) sollen gemeinsam und in strukturierter Art und Weise ausgewertet werden können. Hierfür werden zunächst alle policies zu einer äquivalenten resultierenden policy vereinigt (verwendete Funktion: `Policy.merge(Policy policy)`), so dass sich die weitere Auswertung auf ein einzelnes Dokument beschränkt. Da die einzelnen eingegebenen policies sowie die resultierende policy in Kompaktform vorliegen können, wird die resultierende policy normalisiert (verwendete Funktion: `normalize(PolicyRegistry reg, boolean deep)`. Somit liegen alle assertions strukturiert in Form einer Konjunktion von Disjunktionen vor (Normalform), die im weiteren Verlauf automatisiert verarbeitet werden können.

Von der Möglichkeit assertions, policy alternatives und policies softwareunterstützt zu erstellen bzw. zu bearbeiten wird nicht Gebrauch gemacht, da dies nicht Teil der vorliegenden Arbeit ist. Hier wäre eine weitere Integration von Apache Neethi mit der vorliegenden Arbeit zu einem späteren Zeitpunkt denkbar. Die Funktionen für policy intersection werden ebenfalls nicht verwendet, da diese aufgrund des vorgestellten Programmablaufs im folgenden Abschnitt 4.4 nicht benötigt werden.

## 4.4. Prototyp

In diesem Absatz wird das Design für eine prototypische Implementierung der vorgeschlagenen Konzepte für die Vorbereitung des Softwarebetriebs aus Entwicklersicht vorgestellt. Zunächst wird der in Unterabschnitt 3.6.3 vorgeschlagene Programmablauf um einige implementierungsspezifische Details erweitert. Im Anschluss werden Aktivitäten identifiziert, die bei Interaktionen mit dem Programm stattfinden. Die gewonnenen Erkenntnisse werden sodann genutzt, um ein UML-Klassendiagramm

der zu implementierenden Anwendung zu erstellen. Abschließend wird die Kommunikation der instanziierten Klassen untereinander mit Hilfe eines UML-Sequenzdiagramms modelliert.

### 4.4.1. Programmablauf

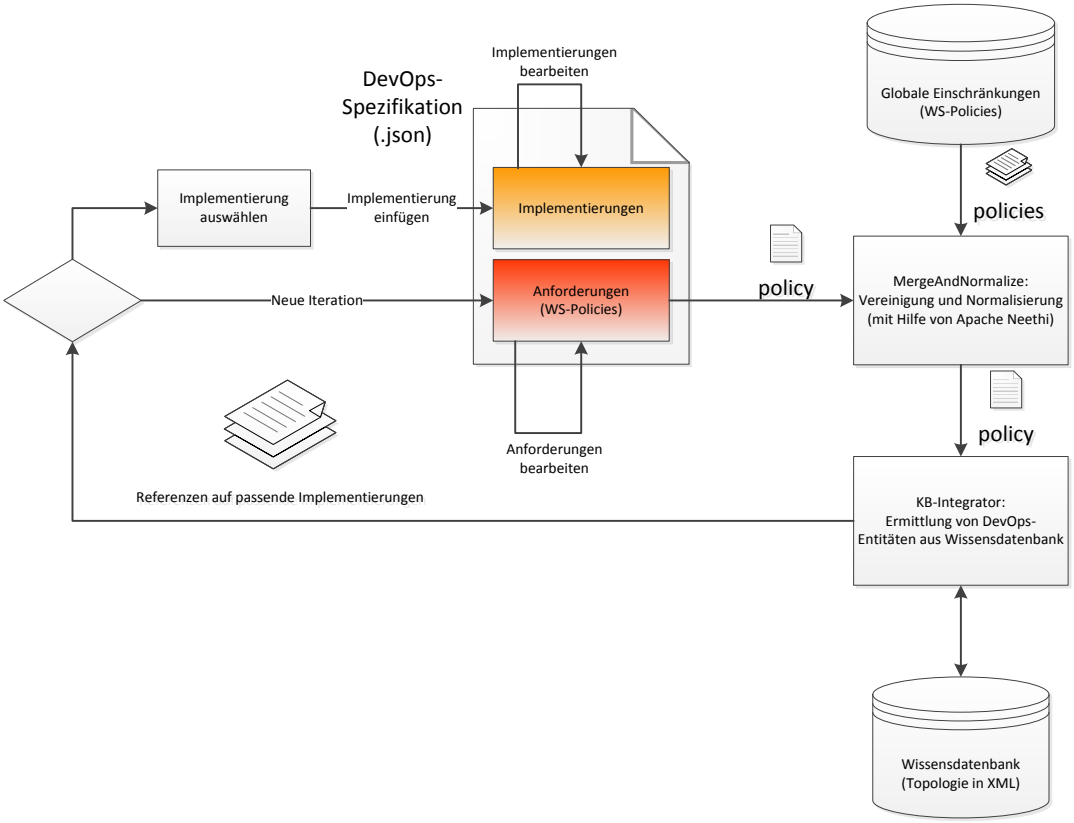


**Abbildung 4.3.:** Programmablauf der vorgeschlagenen Implementierung

Abbildung 4.3 zeigt, wie die unterschiedlichen Komponenten des Systems zusammenhängen. Ein- und Ausgabe der Anwendung ist eine DevOps-Spezifikation in der JavaScript Object Notation (JSON) [jsob] mit den zwei je Aktion enthaltenen Feldern *Anforderungen*, die als WS-Policies formuliert werden, sowie *Implementierung*. Der Zugriff auf die Spezifikation erfolgt über die Komponente *DevOpsSpecHandling*, welche Implementierungen und Anforderungen aus der Spezifikation auslesen sowie einfügen (und löschen) kann. Es existiert eine Datenbasis, in der eine Menge von global definierten Einschränkungen gespeichert ist. Die Einschränkungen sind, genauso wie die Anforderungen, als WS-Policies formuliert, so dass mit Hilfe der Komponente *MergeAndNormalize* durch Vereinigung und Normalisierung aller vorliegenden Anforderungen und Einschränkungen, eine resultierende

WS-Policy generiert werden kann. In einer weiteren Datenbank steht eine mittels XML strukturierte Topologie von Implementierungen zur Verfügung. Die Komponente *KnowledgeBaseIntegrator* ermittelt anhand der im vorangegangenen Schritt erstellten WS-Policy passende DevOps-Entitäten aus der Datenbank. Hierbei entsteht eine Mengen von Referenzen auf Implementierungen, da die Erfüllung aller Anforderungen und Einschränkungen i.A. durch viele verschiedene Kombinationen von Implementierungen erreicht werden kann. Anhand der Ergebnisse kann nun entschieden werden, ob eine Implementierung in die Spezifikation übernommen wird oder ob eine neue Iteration mit veränderten Anforderungen gestartet wird.

### 4.4.2. Aktivitäten

Anhand des in Unterabschnitt 4.4.1 vorgestellten Programmablaufs lassen sich Aktivitäten festlegen, die die grundlegenden Funktionalitäten der zu implementierenden Anwendung bilden. Sie bestehen jeweils aus verschiedenen Mengen von elementaren Aktionen, die miteinander vernetzt sind (Datenfluss oder Kontrollfluss). Die sieben definierten Aktivitäten sind:

- Lesen von Anforderungen aus einer Spezifikation
- Einfügen von Anforderungen in eine Spezifikation (vorhandene Anforderungen werden überschrieben)
- Löschen von Anforderungen aus einer Spezifikation
- Lesen einer Implementierung aus einer Spezifikation
- Einfügen einer Implementierung in eine Spezifikation (vorhandene Implementierung wird überschrieben)
- Löschen einer Implementierung aus einer Spezifikation
- Ermitteln von Implementierungen aus der Wissensdatenbank

Die Aktivitäten *Anforderungen / Implementierung Bearbeiten* (bzw. Aktualisieren) werden an dieser Stelle nicht spezifiziert, da sie durch die beiden Aktivitäten *Lesen von Anforderungen / einer Implementierung aus einer Spezifikation* und *Einfügen von Anforderungen / einer Implementierung in eine Spezifikation* und einer manuellen Bearbeitung bzw. Aktualisierung zwischen den beiden Aktivitäten bereits abgedeckt sind.

Das *Lesen von Anforderungen aus einer Spezifikation* besteht aus den Aktionen *DevOps-Spezifikation wählen*, *Operation und Aktion wählen* und *Anforderungen aus Spezifikation auslesen*.

Das *Einfügen von Anforderungen in eine Spezifikation* umfasst die Aktionen *DevOps-Spezifikation wählen*, *Einzufügende Anforderungen wählen* (WS-Policy), *Operation und Aktion wählen* und *Anforderungen in Spezifikation einfügen* in der angegebenen Reihenfolge. Anforderungen, die an der entsprechenden Stelle in der Spezifikation enthalten sind, werden überschrieben. Beim Einfügen von Anforderungen in eine Spezifikation wird auch der zugehörige *resolver* festgelegt und in die DevOps-Spezifikation eingefügt.

Das *Löschen von Anforderungen aus einer Spezifikation* umfasst die Aktionen *DevOps-Spezifikation wählen*, *Operation und Aktion wählen* und *Anforderungen aus Spezifikation löschen* in der angegebenen Reihenfolge. Hierbei wird auch der zugehörige *resolver* gelöscht.

Das *Lesen einer Implementierung aus einer Spezifikation* besteht aus den Aktionen *DevOps-Spezifikation wählen*, *Operation und Aktion wählen* und *Implementierung aus Spezifikation auslesen*.

Das *Einfügen einer Implementierung in eine Spezifikation* umfasst die Aktionen *DevOps-Spezifikation wählen*, *Einzufügende Implementierung wählen*, *Operation und Aktion wählen* und *Implementierung in Spezifikation einfügen* in der angegebenen Reihenfolge. Eine Implementierung, die an der entsprechenden Stelle in der Spezifikation enthalten ist, wird überschrieben. Beim Einfügen einer Implementierung in eine Spezifikation wird auch der zugehörige *runner* sowie ein *comment* festgelegt und in die DevOps-Spezifikation eingefügt.

Das *Löschen einer Implementierung aus einer Spezifikation* umfasst die Aktionen *DevOps-Spezifikation wählen*, *Operation und Aktion wählen* und *Implementierung aus Spezifikation entfernen* in der angegebenen Reihenfolge.

Das *Ermitteln von Implementierungen aus der Wissensdatenbank* umfasst die Aktionen *Wissensdatenbank wählen*, *Anforderungen aus Spezifikation extrahieren* (WS-Policy) und parallel dazu *Einschränkungen aus Datenbasis holen* (WS-Policies), *Vereinigung und Normalisierung der vorliegenden policies*, *Passende Implementierungen aus der Wissensdatenbank ermitteln* und *Implementierung auswählen* in der angegebenen Reihenfolge. Diese Aktivität stellt die Kernfunktionalität der Anwendung dar und ist im Vergleich zu den anderen drei Aktivitäten aufwendiger zu implementieren. Insbesondere auf die Aktion *Passende Implementierungen aus der Wissensdatenbank ermitteln* wurde bereits in Abschnitt 3.6.4 näher eingegangen.

Zur Veranschaulichung finden sich im Anhang für die genannten Aktivitäten UML-Aktivitätsdiagramme in den Abbildungen A.1 bis A.7.

### 4.4.3. Klassendiagramm

Abbildung 4.4 zeigt das Klassendiagramm für die prototypische Implementierung entsprechend dem vorgestellten Design.

**Master** Die Klasse *Master* dient als Einstiegspunkt für das Programm und stellt die komplette Funktionalität an der Kommandozeilenschnittstelle zur Verfügung (siehe Abschnitt 4.5). Hierfür werden in der einzigen enthaltenen Methode *main()* die beim Programmstart übergebenen Argumente angenommen und ausgewertet. Je nach Bedarf wird ein Objekt der Klasse *DevOpsSpecHandling* oder der Klasse *KnowledgeBaseHandling* mittels Aggregation eingebunden und verwendet.

**DevOpsSpecHandling**
Die Klasse *DevOpsSpecHandling* stellt alle Funktionen zur Verfügung, die für die Bearbeitung einer Aktion in einer DevOps-Spezifikation notwendig sind. Hierfür werden die Klassen *KnowledgeBaseHandling*, *GSON* und *json.simple* mittels Aggregation eingebunden. Die Methode *getRequirementsFromSpec()* ermittelt alle vorliegenden Anforderungen für die entsprechende Aktion und gibt diese als WS-Policy in Form eines Strings in XML-Format zurück. Über die Methode *addRequirementsToSpec()*

werden neue Anforderungen in die entsprechende Aktion eingetragen. Mittels *deleteRequirements-FromSpec()* werden die Attribute *requirements* und *resolver* aus der entsprechenden Aktion entfernt. Die Methode *getImplementationFromSpec()* ermittelt die vorliegende Implementierung der Aktion und gibt diese in Form eines Strings in XML-Format zurück. Die Methode *addImplementationToSpec()* fügt der Aktion eine Implementierung hinzu. An dieser Stelle wird dann im Programmablauf die tatsächliche Implementierung aus der Wissensdatenbank ermittelt und in die Spezifikation eingefügt (mit Hilfe der Methode *getImplementationByName()* der Klasse *KnowledgeBaseHandling*). Die Methode *deleteImplementationFromSpec()* löscht die in der Spezifikation hinterlegte Implementierung der Aktion.

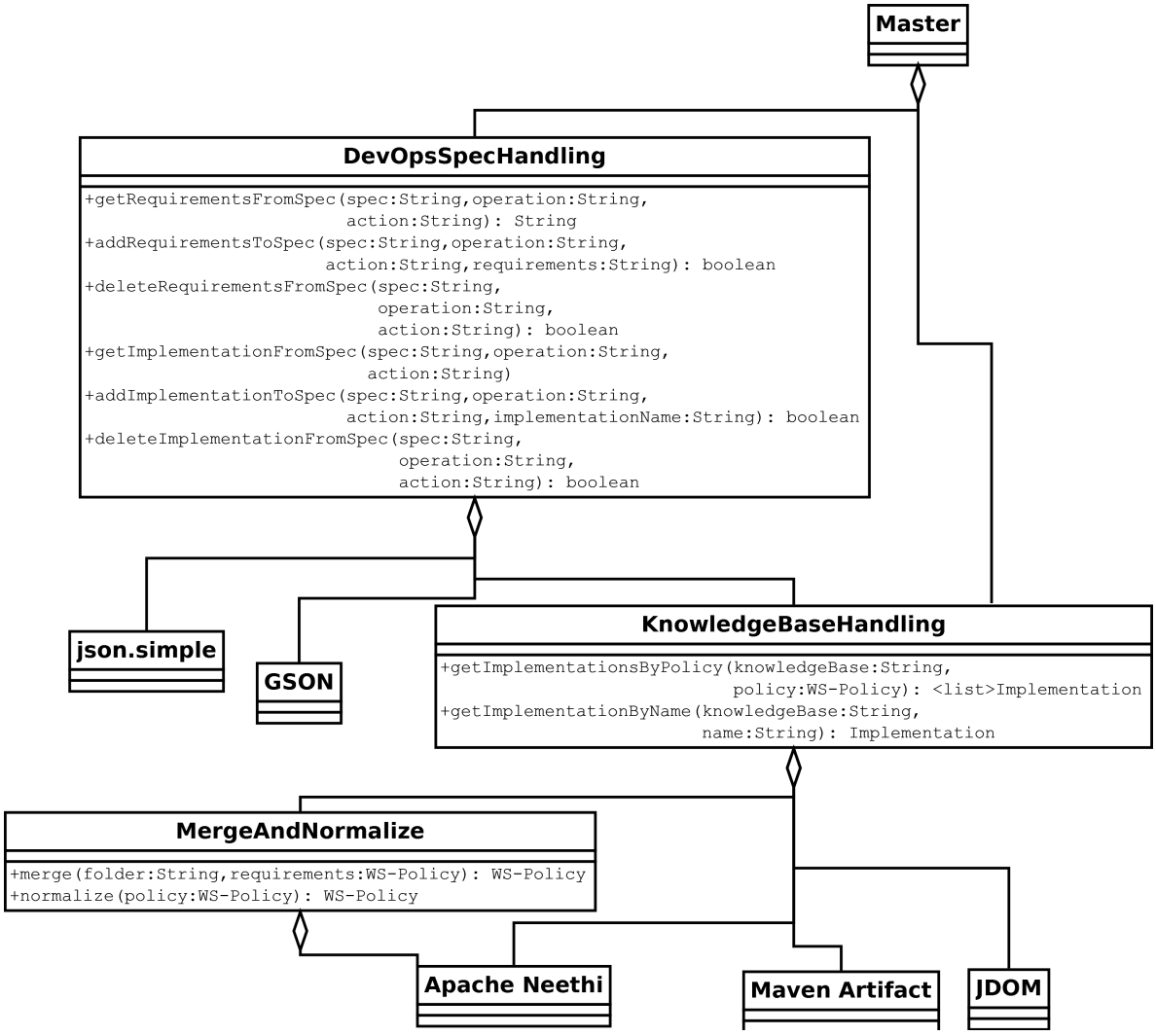


**Abbildung 4.4.:** Klassendiagramm der vorgeschlagenen Implementierung

**MergeAndNormalize**
Die Klasse *MergeAndNormalize* stellt alle Funktionen zur Verfügung, die für die Vereinigung der

Anforderungen und Einschränkungen wichtig sind. Hierfür wird die Klasse *Apache Neethi* mittels Aggregation eingebunden. Die Methode *merge()* vereinigt eine als Parameter übergebene WS-Policy (Anforderungen aus der Spezifikation) mit den globalen Einschränkungen , die als Menge von WS-Policies in Form von XML-Dateien in einem Ordner vorliegen. Alle WS-Policies, die sich in dem angegebenen Ordner befinden, werden als Einschränkungen in die resultierende WS-Policy übernommen. Die Methode *normalize()* normalisiert eine WS-Policy.

**KnowledgeBaseHandling**
Die Klasse *KnowledgeBaseHandling* stellt alle Funktionen zur Verfügung, die für die Handhabung der Wissensdatenbank wichtig sind. Hierfür werden die Klassen *Apache Neethi*, *Maven Artifact* und *JDOM* mittels Aggregation eingebunden. Die Methode *getImplementationsByPolicy()* ermittelt eine Menge von möglichen Implementierungen, die die zuvor ermittelte WS-Policy erfüllen. Die Methode *getImplementationByName()* hingegen ermittelt eine Implementierung, welche anhand des eindeutigen Namens in der Topologie identifiziert wird. Beide Methoden erwarten als erstes Argument (*knowledgeBase*) eine Referenz auf die zu verwendende Wissensdatenbank, die als XML-Dokument vorliegen muss.

### 4.4.4. Sequenzdiagramm

Das UML-Sequenzdiagramm in Abbildung 4.5 zeigt die Interaktionen der Instanzen der zuvor beschriebenen Klassen für die Ermittlung von passenden Implementierungen aus der Wissensdatenbank sowie des anschließenden Einfügens einer Implementierung in eine DevOps-Spezifikation. Das Objekt *Master* dient der Integration der weiteren Objekte und stellt die Programmfunktionalität in Form eines Kommandozeilenprogramms bereit. Zu Beginn werden die gewählten Anforderungen aus der Spezifikation ermittelt (*DevOpsSpecHandling.getRequirementsFromSpec()*), welche in Form einer WS-Policy zurückgegeben werden und im nächsten Schritt mit den WS-Policies aus der Datenbasis für die globalen Einschränkungen im Objekt *MergeAndNormalize* vereinigt werden. Das Ergebnis dieses Schrittes wird dann mittels *MergeAndNormalize.normalize()* in die Normalform gebracht, so dass anschließend mittels der Methode *KnowledgeBaseHandling.getImplementationsByPolicy()* eine Liste von (Referenzen auf) Implementierungen ermittelt werden kann, die allen vorliegenden Anforderungen und Einschränkungen genügen. Die Auswahl einer dieser Implementierungen wird schließlich mittels der Methode *DevOpsSpecHandling.addImplementationToSpec()* in die Spezifikation eingefügt.

An den mit den Ziffern 1 und 2 gekennzeichneten Stellen findet jeweils eine Benutzerinteraktion statt. An der Stelle 1 wird die Aktivität *Ermitteln von Implementierungen aus der Wissensdatenbank* ausgeführt. An der Stelle 2 wird schließlich eine der zuvor ermittelten Implementierungen ausgewählt, die im Anschluss in die Spezifikation eingefügt wird (Aktivität *Einfügen einer Implementierung in eine Spezifikation.*

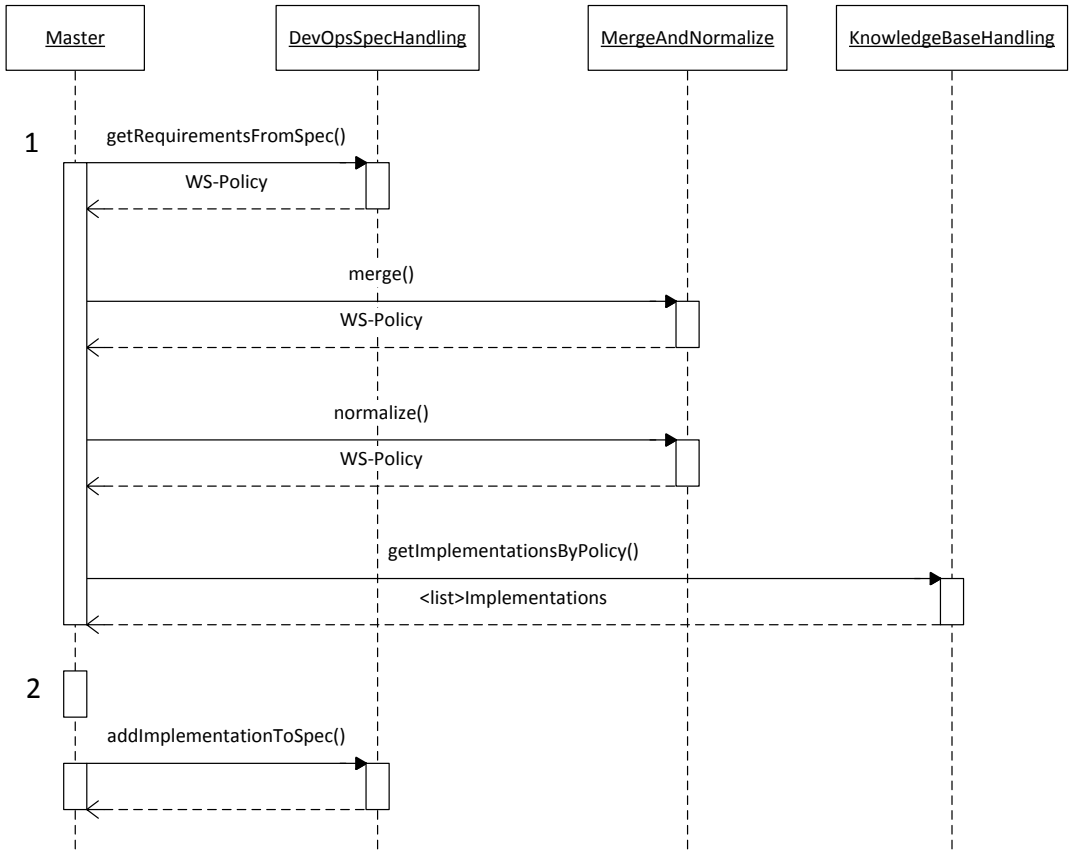


**Abbildung 4.5.:** Sequenzdiagramm: Ermitteln und einfügen einer Implementierung

## 4.5. Kommandozeilenschnittstelle

Für den Benutzer der Anwendung wird ein ausführbares, zustandsloses *Java Archive File* [jar] bereitgestellt, das die angebotenen Funktionen für den Benutzer an einer Kommandozeile verwendbar macht. Ein Aufruf der Anwendung an einer Unix-Shell findet mit dem Befehl `java -jar applicationName.jar [FUNKTION] [ARGUMENT]...` statt.

Die übergebenen Argumente bestehen aus dem Namen der Programmfunktion (`[FUNKTION]`), die verwendet werden soll, sowie einer Menge von weiteren Argumenten (`[ARGUMENT]...`), die an die angegebene Funktion übergeben werden. Die Rückgabewerte aller Funktionen werden an die Kommandozeile zurück geleitet.

Die folgenden Funktionen stehen dem Benutzer zur Verfügung (entsprechend den in Unterabschnitt 4.4.2 definierten Aktivitäten).

**getRequirementsFromSpec**
Auslesen von Anforderungen aus einer Spezifikation

| | |
|---|---|
| Argument 0: | Spezifikation, die bearbeitet wird (absoluter oder relativer Dateipfad zu einer Spezifikationsdatei in JSON-Format) |
| Argument 1: | Operation, die bearbeitet wird |
| Argument 2: | Aktion, die bearbeitet wird |
| Beispiel: | `java -jar applicationName.jar getRequirementsFromSpec /home/user1/spec.json build mantis` |

**insertRequirementsIntoSpec**
Einfügen von Anforderungen in eine Spezifikation

| | |
|---|---|
| Argument 0: | Spezifikation, die bearbeitet wird (absoluter oder relativer Dateipfad zu einer Spezifikationsdatei in JSON-Format) |
| Argument 1: | Operation, die bearbeitet wird |
| Argument 2: | Aktion, die bearbeitet wird |
| Argument 3: | Einzufügende Anforderungen (absoluter oder relativer Dateipfad zu einer WS-Policy-Datei in XML-Format) |
| Beispiel: | `java -jar applicationName.jar insertRequirementsIntoSpec /home/user1/spec.json build mantis /home/user1/requirements.xml` |

**deleteRequirementFromSpec**
Entfernen von Anforderungen aus einer Spezifikation

| | |
|---|---|
| Argument 0: | Spezifikation, die bearbeitet wird (absoluter oder relativer Dateipfad zu einer Spezifikationsdatei in JSON-Format) |
| Argument 1: | Operation, die bearbeitet wird |
| Argument 2: | Aktion, die bearbeitet wird |
| Beispiel: | `java -jar applicationName.jar deleteRequirementFromSpec /home/user1/spec.json build mantis` (macht die Ausführung des vorangegangenen Beispiels rückgängig) |

**getImplementationFromSpec**
Auslesen einer Implementierung aus einer Spezifikation

| | |
|---|---|
| Argument 0: | Spezifikation, die bearbeitet wird (absoluter oder relativer Dateipfad zu einer Spezifikationsdatei in JSON-Format) |
| Argument 1: | Operation, die bearbeitet wird |
| Argument 2: | Aktion, die bearbeitet wird |
| Beispiel: | `java -jar applicationName.jar getImplementationFromSpec /home/user1/spec.json deploy deploy-linux` |

**insertImplementationIntoSpec**
Einfügen einer Anforderung in eine Spezifikation

| | |
|---|---|
| Argument 0: | Spezifikation, die bearbeitet wird (absoluter oder relativer Dateipfad zu einer Spezifikationsdatei in JSON-Format) |
| Argument 1: | Operation, die bearbeitet wird |
| Argument 2: | Aktion, die bearbeitet wird |
| Argument 3: | Name der einzufügenden Implementierung |
| Beispiel: | `java -jar applicationName.jar insertImplementationIntoSpec /home/user1/spec.json deploy deploy-linux "Infrastructure/Operating System/Linux/RedHat/Red Hat Enterprise Linux 7.0 AMI HVM 64-bit US East"` |

**deleteImplementationFromSpec**
Entfernen einer Implementierung aus einer Spezifikation

| | |
|---|---|
| Argument 0: | Spezifikation, die bearbeitet wird (absoluter oder relativer Dateipfad zu einer Spezifikationsdatei in JSON-Format) |
| Argument 1: | Operation, die bearbeitet wird |
| Argument 2: | Aktion, die bearbeitet wird |
| Beispiel: | `java -jar applicationName.jar deleteImplementationFromSpec /home/user1/spec.json deploy deploy-linux` (macht die Ausführung des vorangegangenen Beispiels rückgängig) |

**getImplementationsFromKB**
Ermittlung von Implementierungen aus der Wissensdatenbank entsprechend den gegebenen Anforderungen und Einschränkungen

| | |
|---|---|
| Argument 0: | Spezifikation, die bearbeitet wird (absoluter oder relativer Dateipfad zu einer Spezifikationsdatei in JSON-Format) |
| Argument 1: | Operation, die bearbeitet wird |
| Argument 2: | Aktion, die bearbeitet wird |
| Argument 3: | Maximalwert für Anzahl der ermittelten Implementierungen (Integer) |
| Argument 4: | Wissensdatenbank, in der Implementierungen in Form einer Topologie hinterlegt sind (absoluter oder relativer Dateipfad zu einer Wissensdatenbank in XML-Format) |
| Argument 5: | Globale Einschränkungen (absoluter oder relativer Dateipfad zu einem Dateiordner, der eine Menge von WS-Policy-Dateien in XML-Format enthält) |
| Beispiel: | `java -jar applicationName.jar getImplementationsFromKB /home/user1/spec.json deploy deploy-linux 3 /home/user1/kb.xml /home/user1/constraints` |

Ein beispielhafter Ablauf könnte wie folgt aussehen: Zunächst wird festgelegt, welche Operation und Aktion einer DevOps-Spezifikation bearbeitet wird (diese Angaben werden unverändert allen aufgerufenen Funktionen als Argument 0 bis 2 übergeben). Mittels *getRequirementsFromSpec* kann nun inspiziert werden, welche Anforderungen für die entsprechende Aktion hinterlegt sind. Mit der Funktion *insertRequirementsIntoSpec* werden der Aktion neue Anforderungen hinzugefügt, die als WS-Policy in Form einer XML-Datei vorliegen und als Argument 3 übergeben werden. Nun wird über *getImplementationsFromKB* eine Menge von passenden Implementierungen, die den vorliegenden Anforderungen und Einschränkungen entsprechen, aus der Wissensdatenbank ermittelt. Hierbei wird mit Argument 3 die Anzahl der zurückgegebenen Implementierungen begrenzt. Mit den Argumenten 4 und 5 werden die zu verwendende Wissensdatenbank (XML-Datei) und die globalen Einschränkungen (Menge von XML-Dateien) festgelegt. Die Ergebnisse der Funktion werden an der Kommandozeile als Referenzen auf Implementierungen in der Wissensdatenbank angezeigt. Im einem abschließenden Schritt kann nun mit Hilfe der Funktion *insertImplementationIntoSpec* eine der zuvor ermittelten Implementierungen in die aktuell bearbeitete Aktion eingefügt werden.

# 5. Evaluation

In diesem Abschnitt wird die vorgestellte prototypische Implementierung anhand von drei Einsatzszenarien evaluiert. Als Eingaben werden drei verschiedene DevOps-Spezifikationen zur Verfügung gestellt (*specificationLinux.json*, *specificationMantis.json* und *specificationRedmine.json*), welche jeweils ausschließlich die Operation *deploy* beinhalten. Diese Operation wiederum beinhaltet unterschiedliche Aktionen, für die im weiteren Verlauf entsprechende Implementierungen aus der vorliegenden Wissensdatenbank ermittelt werden. Für jede dieser Aktionen stehen zudem Anforderungen und globale Einschränkungen (in den Szenarien 1.3 und 1.4) bereit.

In einem ersten Schritt werden pro Szenario die Anforderungen an die entsprechenden Stellen in die Spezifikation eingefügt. Im Anschluss werden diese automatisiert ausgewertet und mit den vorliegenden globalen Einschränkungen abgeglichen, so dass eine Auswahl von passenden Implementierungen aus der Wissensdatenbank ermöglicht wird. Abschließend wird jeweils eine der in Frage kommenden Implementierungen in die aktuell bearbeitete Aktion eingefügt.

**Szenario 1 (Linux)**

In Szenario 1 wird eine Linux-Implementierung (Ubuntu) aus der Wissensdatenbank ermittelt. Hierbei wird die Anforderung gestellt, dass das Betriebssystem eine 64-Bit-Prozessorarchitektur unterstützt, sofern eine entsprechende Implementierung in der Wissensdatenbank vorhanden ist (*wsp:Ignorable*). Anhand der WS-Policy in Listing 5.1 wird diese Anforderung definiert.

**Listing 5.1** WS-Policy für Anforderungen an Szenario 1

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:devops="my.devops.name">
      <wsp:ExactlyOne>
            <wsp:All>
                  <devops:valueEq entity="Infrastructure/Operating System/Linux/Ubuntu"
                      property="arch" value="x86-64" wsp:Ignorable="true"/>
            </wsp:All>
      </wsp:ExactlyOne>
</wsp:Policy>
```

Um zu zeigen, dass die Auswahl der Implementierung korrekt funktioniert, werden die verwendete Wissensdatenbank sowie die verwendeten globalen Einschränkungen schrittweise verändert, wodurch vier verschiedene Subszenarien simuliert werden:

1. 64-Bit-Architektur in Wissensdatenbank vorhanden + keine globale Einschränkungen
2. 64-Bit-Architektur nicht in Wissensdatenbank vorhanden + keine globale Einschränkungen

3. 64-Bit-Architektur in Wissensdatenbank vorhanden + globale Einschränkung auf 64-Bit-Architekturen für Betriebssysteme
4. 64-Bit-Architektur nicht in Wissensdatenbank vorhanden + globale Einschränkung auf 64-Bit-Architekturen für Betriebssysteme

Listing 5.2 zeigt die WS-Policy für globalen Einschränkung auf 64-Bit-Architekturen für Betriebssysteme.

**Listing 5.2** WS-Policy für globale Einschränkungen auf 64-Bit-Architekturen für Betriebssysteme für Szenario 1.3 und 1.4

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:devops="my.devops.name">
      <wsp:ExactlyOne>
            <wsp:All>
                  <devops:valueEq entity="Infrastructure/Operating System" property="arch"
                      value="x86-64"/>
            </wsp:All>
      </wsp:ExactlyOne>
</wsp:Policy>
```

Als Eingabe für Szenario 1 dient die in Listing 5.3 gezeigte Spezifikation. Die Operation *deploy* enthält die Aktion *deploy-linux*, welche weder Anforderungen noch eine Implementierung enthält.

**Listing 5.3** DevOps-Spezifikation für Szenario 1

```
[{
  "author": "Sebastian Kiesewetter",
  "description": "Evaluation Szenario 1 (Linux)",
  "name": "linux",
  "operations": {
    "deploy": {
      "artifacts": {
        "deploy-linux": {}
      }
    }
  }
}]
```

In der Wissensdatenbank liegen unterschiedliche Linux-Implementierungen mit einer 64-Bit-Architektur vor. Die (manuelle) Auswahl einer Alternative aus den ermittelten passenden Implementierungen wird an dieser Stelle auf *Ubuntu Server 14.04 AMI PV 64-bit US East* festegelegt. Die resultierende Spezifikation für Szenario 1.1 ist in Listing 5.4 dargestellt (die *requirements* sind aus Gründen der Übersichtlichkeit gekürzt dargestellt).

**Listing 5.4** Resultierende DevOps-Spezifikation für Szenario 1.1

```
[{
 "author": "Sebastian Kiesewetter",
 "description": "Evaluation Szenario 1 (Linux)",
 "name": "linux",
 "operations": {
  "deploy": {
   "artifacts": {
    "deploy-linux": {
                "runner": "AMI_Provisioner",
                "config": {
                      "name": "Ubuntu Server 14.04 AMI PV 64-bit US East",
                      "ami_id": "ami-018c9568",
                      "version": "14.04",
                      "aws_region": "us-east-1"
                },
                "comment": "auto-resolved at 2014-09-01 using WS-Policy",
                "resolver": "WS-Policy",
                "requirements": "<wsp:Policy ...
                      </wsp:Policy>"
    }
   }
  }
 }
}]
```

Für Szenario 1.2 kann keine Implementierung mit einer 64-Bit-Architektur aus der Wissensdatenbank ermittelt werden, so dass ersatzweise Implementierungen mit einer 32-Bit-Architektur ermittelt werden (manuelle Auswahl einer Alternative: *Ubuntu Server 14.04 AMI PV 32-bit US East*). Die resultierende Spezifikation für Szenario 1.2 ist in Listing 5.10 dargestellt (die *requirements* sind aus Gründen der Übersichtlichkeit gekürzt dargestellt).

**Listing 5.5** Resultierende DevOps-Spezifikation für Szenario 1.2

```
[{
  "author": "Sebastian Kiesewetter",
  "description": "Evaluation Szenario 1 (Linux)",
  "name": "linux",
  "operations": {
   "deploy": {
    "artifacts": {
     "deploy-linux": {
                 "runner": "AMI_Provisioner",
                 "config": {
                       "name": "Ubuntu Server 14.04 AMI PV 32-bit US East",
                       "ami_id": "ami-358c955c",
                       "version": "14.04",
                       "aws_region": "us-east-1"
                 },
                 "comment": "auto-resolved at 2014-09-01 using WS-Policy-Resolver",
                 "resolver": "WS-Policy",
                 "requirements": "<wsp:Policy ...
                       </wsp:Policy>"
     }
    }
   }
  }
}]
```

Die resultierende Spezifikation für Szenario 1.3 entspricht der resultierenden Spezifikation von Szenario 1.1 und wird daher an dieser Stelle nicht noch einmal dargestellt.

Für Szenario 1.4 lässt sich keine passende Implementierung aus der Wissensdatenbank ermitteln, da die globalen Anforderungen eine 64-Bit-Architektur erforderlich machen, die jedoch in der Wissensdatenbank nicht vorhanden ist.

**Szenario 2 (Mantis)**

In Szenario 2 werden Implementierungen von Middlewarekomponenten für eine DevOps-Spezifikation aus der Wissensdatenbank ermittelt, die für die Installation des *Bugtracking Systems Mantis* notwendig sind. Ausgewählt wird ein Webserver, eine Laufzeitumgebung sowie eine Datenbank (siehe Abbildung 3.7). Globale Einschränkungen liegen in diesem Szenario nicht vor.

Listing 5.6 zeigt die WS-Policy für die Anforderung, dass ein Webserver vorhanden ist. Hierfür wird der *Apache HTTP Server* explizit ausgewählt.

**Listing 5.6** WS-Policy für Anforderungen an Szenario 2 (Webserver)

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:devops="my.devops.name">
       <wsp:ExactlyOne>
             <wsp:All>
                   <devops:entityReq entity="Middleware/Web Server/Apache HTTP Server"/>
             </wsp:All>
       </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 5.7 zeigt die WS-Policy für die Anforderung, dass eine PHP-Laufzeitumgebung vorhanden ist. Im weiteren Verlauf wird hierzu die Implementierung *PHP Application Chef Cookbook* ausgewählt.

**Listing 5.7** WS-Policy für Anforderungen an Szenario 2 (Laufzeitumgebung)

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:devops="my.devops.name">
       <wsp:ExactlyOne>
             <wsp:All>
                   <devops:entityReq entity="Middleware/Runtime/PHP"/>
             </wsp:All>
       </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 5.8 zeigt die WS-Policy für die Anforderung, dass eine Datenbank vorhanden ist. Laut Dokumentation arbeitet Mantis unter anderem mit einer MySQL-Datenbank ab Version 5.0 oder mit einer PostgreSQL-Datenbank ab Version 8.0 zusammen. Die beiden genannten Möglichkeiten werden in der WS-Policy in zwei separaten Alternativen ausgedrückt. Da mit der vorliegenden Wissensdatenbank beide Alternativen bedient werden können, wird eine Liste mit passenden Implementierungen für beide Alternativen zurückgegeben, aus denen dann im weiteren Programmablauf die Auswahl *MySQL Server 5.x Chef Cookbook* getroffen wird.

**Listing 5.8** WS-Policy für Anforderungen an Szenario 2 (Datenbank)

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:devops="my.devops.name">
       <wsp:ExactlyOne>
             <wsp:All>
                   <devops:valueEqGr entity="Middleware/Database/Relational/MySQL"
                        property="provided_versions" value="5.0"/>
             </wsp:All>
             <wsp:All>
                   <devops:valueEqGr entity="Middleware/Database/Relational/PostgreSQL"
                        property="provided_versions" value="8.0"/>
             </wsp:All>
       </wsp:ExactlyOne>
</wsp:Policy>
```

Als Eingabe für Szenario 2 dient die in Listing 5.9 gezeigte Spezifikation. Die Operation *deploy* enthält die Aktionen *deploy-webserver*, *deploy-runtime* und *deploy-database*, welche weder Anforderungen noch Implementierungen enthalten.

**Listing 5.9** DevOps-Spezifikation für Szenario 2

```
[{
  "author": "Sebastian Kiesewetter",
  "description": "Evaluation Szenario 2 (Mantis)",
  "name": "mantis",
  "operations": {
    "deploy": {
      "artifacts": {
        "deploy-webserver": {},
        "deploy-runtime": {},
        "deploy-database": {}
      }
    }
  }
}]
```

Eine resultierende Spezifikation für Szenario 2, abhängig von den ausgewählten Implementierungen, ist in Listing 5.10 dargestellt (*requirements* und *resolver* sind aus Gründen der Übersichtlichkeit nicht dargestellt). Die Aktion *deploy-os-for-webserver* wurde automatisch aufgrund der Eigenschaft *hosted_on_one_of* mit dem Wert *Infrastructure/OS/Linux/Ubuntu* des eingefügten *Apache Chef Cookbooks* erstellt.

**Listing 5.10** Resultierende DevOps-Spezifikation für Szenario 2

```
[{
  "author": "Sebastian Kiesewetter",
  "description": "Evaluation Szenario 2 (Mantis)",
  "name": "mantis",
  "operations": {
    "deploy": {
      "artifacts": {
        "deploy-webserver": {
                    "runner": "Chef Cookbook Deployer",
                    "config": {
                          "name": "Apache Chef Cookbook",
                          "hosted_on_one_of": "Infrastructure/OS/Linux/Ubuntu",
                    },
                    "comment": "auto-resolved at 2014-09-01 using WS-Policy"
      },
            "deploy-os-for-webserver": {
                  "runner": "AMI_Provisioner",
                  "config": {
                        "name": "Ubuntu Server 14.04 AMI PV 64-bit US East",
                        "ami_id": "ami-018c9568",
                        "version": "14.04",
                        "aws_region": "us-east-1"
                  },
                  "comment": "auto-resolved at 2014-09-01 using WS-Policy"
      }
      "deploy-runtime": {
                  "runner": "Chef Cookbook Deployer",
                  "config": {
                        "name": "PHP Application Chef Cookbook",
                        "hosted_on_one_of": "Infrastructure/OS/Linux/Ubuntu",
                  },
                  "comment": "auto-resolved at 2014-09-01 using WS-Policy"
      },
      "deploy-database": {
                  "runner": "Chef Cookbook Deployer",
                  "config": {
                        "name": "MySQL Server 5.x Chef Cookbook",
                        "hosted_on_one_of": "Infrastructure/OS/Linux/Ubuntu",
                        "repository_url": "https://github.com/opscode-cookbooks/mysql.git",
                        "repository_type": "git",
                        "package_url": "http://s3.amazonaws.com/community-files.opscode.com/
                          cookbook_versions/tarballs/6758/original/
                          mysql20140519-29446-71fpvg.?1400550644",
                        "version": "5.0"
                  },
                  "comment": "auto-resolved at 2014-09-01 using WS-Policy"
      }
     }
   }
  }
}]
```

### Szenario 3 (Redmine)

In Szenario 3 werden Implementierungen von Middlewarekomponenten für eine DevOps-Spezifikation aus der Wissensdatenbank ermittelt, die für die Installation des *Project Management*-Werkzeugs *Redmine* notwendig sind. Ausgewählt wird eine Laufzeitumgebung sowie eine Datenbank. Globale Einschränkungen liegen in diesem Szenario nicht vor.

Listing 5.11 zeigt die WS-Policy für die Anforderung, dass eine *Ruby*-Laufzeitumgebung vorhanden ist.

**Listing 5.11** WS-Policy für Anforderungen an Szenario 3 (Laufzeitumgebung)

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:devops="my.devops.name">
       <wsp:ExactlyOne>
             <wsp:All>
                   <devops:entityReq entity="Middleware/Runtime/Ruby"/>
             </wsp:All>
       </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 5.12 zeigt die WS-Policy für die Anforderung, dass eine Datenbank vorhanden ist. Laut Dokumentation arbeitet Redmine unter anderem mit einer MySQL-Datenbank ab Version 5.0 oder mit einer PostgreSQL-Datenbank ab Version 8.2 zusammen. Die genannten beiden Möglichkeiten werden in der WS-Policy in zwei separaten Alternativen ausgedrückt. Da mit der vorliegenden Wissensdatenbank beide Alternativen bedient werden können, wird eine Liste mit passenden Implementierungen für beide Alternativen zurückgegeben, aus denen dann im weiteren Programmablauf die Auswahl *Postgres Juju Charm* getroffen wird.

**Listing 5.12** WS-Policy für Anforderungen an Szenario 3 (Datenbank)

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:devops="my.devops.name">
       <wsp:ExactlyOne>
             <wsp:All>
                   <devops:valueEqGr entity="Middleware/Database/Relational/MySQL"
                        property="provided_versions" value="5.0"/>
             </wsp:All>
             <wsp:All>
                   <devops:valueEqGr entity="Middleware/Database/Relational/PostgreSQL"
                        property="provided_versions" value="8.2"/>
             </wsp:All>
       </wsp:ExactlyOne>
</wsp:Policy>
```

Als Eingabe für Szenario 3 dient die in Listing 5.13 gezeigte Spezifikation. Die Operation *deploy* enthält die Aktionen *deploy-runtime* und *deploy-database*, welche weder Anforderungen noch Implementierungen enthalten.

**Listing 5.13** DevOps-Spezifikation für Szenario 3

```
[{
  "author": "Sebastian Kiesewetter",
  "description": "Evaluation Szenario 3 (Redmine)",
  "name": "redmine",
  "operations": {
    "deploy": {
      "artifacts": {
        "deploy-runtime": {},
        "deploy-database": {}
      }
    }
  }
}]
```

Eine resultierende Spezifikation für Szenario 3, abhängig von den ausgewählten Implementierungen bei jeweils mehreren Alternativen, ist in Listing 5.14 dargestellt (die *requirements* sind aus Gründen der Übersichtlichkeit gekürzt dargestellt).

**Listing 5.14** Resultierende DevOps-Spezifikation für Szenario 3

```
[{
  "author": "Sebastian Kiesewetter",
  "description": "Evaluation Szenario 3 (Redmine)",
  "name": "redmine",
  "operations": {
    "deploy": {
      "artifacts": {
        "deploy-runtime": {
                    "runner": "Chef Cookbook Deployer",
                    "config": {
                          "name": "Ruby Application Chef Cookbook",
                          "hosted_on_one_of": "Infrastructure/OS/Linux/Ubuntu",
                    },
                    "comment": "auto-resolved at 2014-09-01 using WS-Policy",
                    "resolver": "WS-Policy",
                    "requirements": "<wsp:Policy ...
                          </wsp:Policy>"
        },
        "deploy-database": {
                    "runner": "Juju Charm Deployer",
                    "config": {
                          "name": "Postgres Juju Charm",
                          "hosted_on_one_of": "Infrastructure/OS/Linux/Ubuntu"
                    },
                    "comment": "auto-resolved at 2014-09-01 using WS-Policy",
                    "resolver": "WS-Policy",
                    "requirements": "<wsp:Policy ...
                          </wsp:Policy>"
      }
     }
    }
  }
}]
```

### Ergebnis

Wie anhand der vorgestellten drei praxisnahen Szenarien gezeigt worden ist, funktioniert die automatisierte Auswahl von Implementierungen aus der Wissensdatenbank zuverlässig. Insbesondere die Auswertung der Anforderungen und Einschränkungen, formuliert als WS-Policies, führt zu korrekten Ergebnissen. Liegen für das Einfügen in eine Spezifikation mehrere Implementierungen vor, werden diese angezeigt und es kann (manuell) eine Auswahl getroffen werden. Die Anzahl der ermittelten möglichen Implementierungen kann begrenzt werden um bei einer umfangreichen Wissensdatenbank die Laufzeit zu begrenzen.

Für Entwickler ist somit eine neue Möglichkeit geschaffen worden, die Ansprüche an auszuliefernde Software in einer Domäne aus ihrem Fachbereich auszudrücken.

# 6. Fazit

Die vorliegende Arbeit hat Konzepte vorgestellt, mit denen der Softwarebetrieb aus Entwicklersicht vorbereitet werden kann. Es wurde gezeigt, wie sich Ansätze für die Automatisierung von Management und Bereitstellung von IT-Infrastruktur und Anwendungen mit Ausdrucksmöglichkeiten aus der Domäne der Softwareentwickler integrieren lassen. Das Ziel hierbei ist die Unterstützung von Softwareentwicklern bei der Erstellung von Spezifikationen, die als Eingabe für Infrastructure as Code-Werkzeuge dienen.

Zunächst wurde in Kapitel 2 der aktuelle Stand der Technik vorgestellt, wobei der Fokus auf den drei Paradigmen Cloud Computing, Infrastructure as Code und DevOps lag. Hierbei ist gezeigt worden, wie das Cloud Computing als Voraussetzung für die Bereitstellung von virtualisierter IT-Infrastruktur im Rahmen von Infrastructure as Code angesehen werden kann und wie dieses wiederum die Grundlage für die von DevOps empfohlenen Vorgehensweisen darstellt. Eine Auswahl von Infrastructure as Code-Werkzeugen (Chef, Juju und Puppet) für die Automatisierung von Betriebsaufgaben wurde näher untersucht und es wurde die Weiterentwicklung NoOps des DevOps-Paradigmas erläutert.

Die maßgeblichen Beiträge dieser Arbeit sind die in Kapitel 3 erarbeiteten Konzepte zur Vorbereitung des Softwarebetriebs aus Entwicklersicht. Nach einer Erläuterung, wie eine bessere Unterstützung für Softwareentwickler bei der Integration von automatisierten Release-Prozessen mittels Infrastructure as Code-Werkzeugen aussehen kann, wurde zunächst eine Taxonomie für Softwarekomponenten sowie der Aufbau von DevOps-Spezifikationen vorgestellt. Als zusätzliche Motivation wurde nachfolgend ein Beispielszenario für die Anwendung der genannten Konzepte angegeben. Die hierfür notwendigen Anpassungen der zuvor beschriebenen DevOps-Spezifikationen wurden erläutert. Sodann wurde eine Menge von Prädikaten zum Ausdrücken von Anforderungen und Einschränkungen definiert, die für die Auswahl von Implementierungen aus der Taxonomie herangezogen werden können. Der Programmablauf der vorgeschlagenen Softwarelösung im Allgemeinen, sowie der Algorithmus für die Auswahl von Implementierungen aus der Taxonomie anhand von Prädikaten im Speziellen, wurden ausführlich erläutert.

Die Umsetzung der vorgeschlagenen Konzepte erfolgte anhand einer prototypischen Implementierung in Kapitel 4. Für die Umsetzung wurde auf das WS-Policy Framework und dessen Implementierung Apache Neethi zurückgegriffen. Es wurde gezeigt, wie die Formulierung von Anforderungen und Einschränkungen an Softwarekomponenten mittels Zusicherungen in Form von WS-Policies stattfinden kann, wodurch eine automatisierte Auswertung ermöglicht wird. Es wurde argumentiert, dass die Verwendung von WS-Policy als Sprache aus der Domäne der Softwareentwickler die Auswahl, die Installation sowie die Verwaltung von Softwarekomponenten für Softwareentwickler erleichtert.

# A. Anhang

Mit Hilfe von Aktivitätsdiagrammen der UML 2 werden die notwendigen elementaren Aktionen und deren Verbindungen (Kontrollfluss) innerhalb einer Aktivität veranschaulicht. Die folgenden Abbildungen A.1 bis A.7 zeigen für die in Abschnitt 4.4 ermittelten Aktivitäten jeweils das zugehörige Aktivitätsdiagramm.

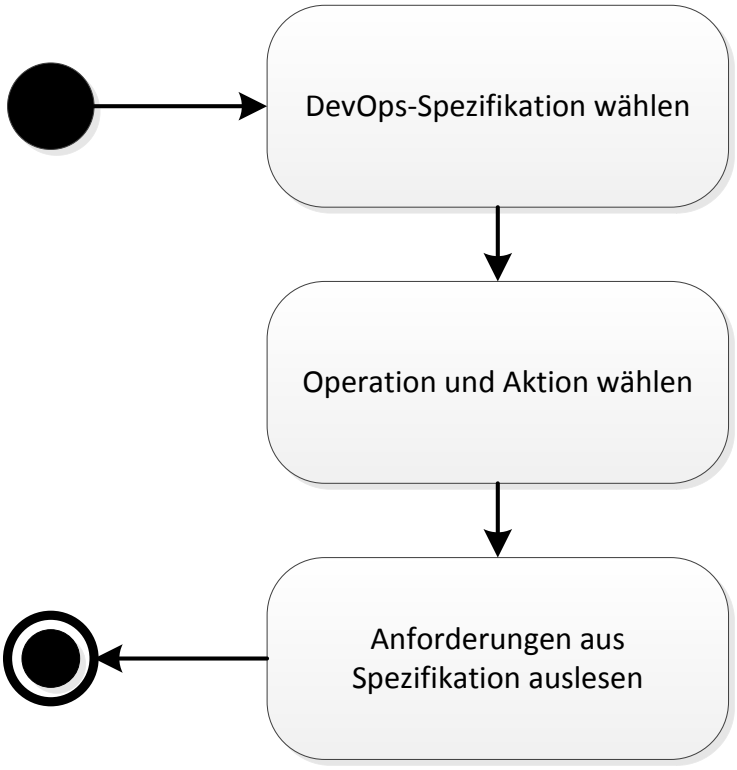


**Abbildung A.1.:** Aktivitätsdiagramm *Anforderungen lesen*

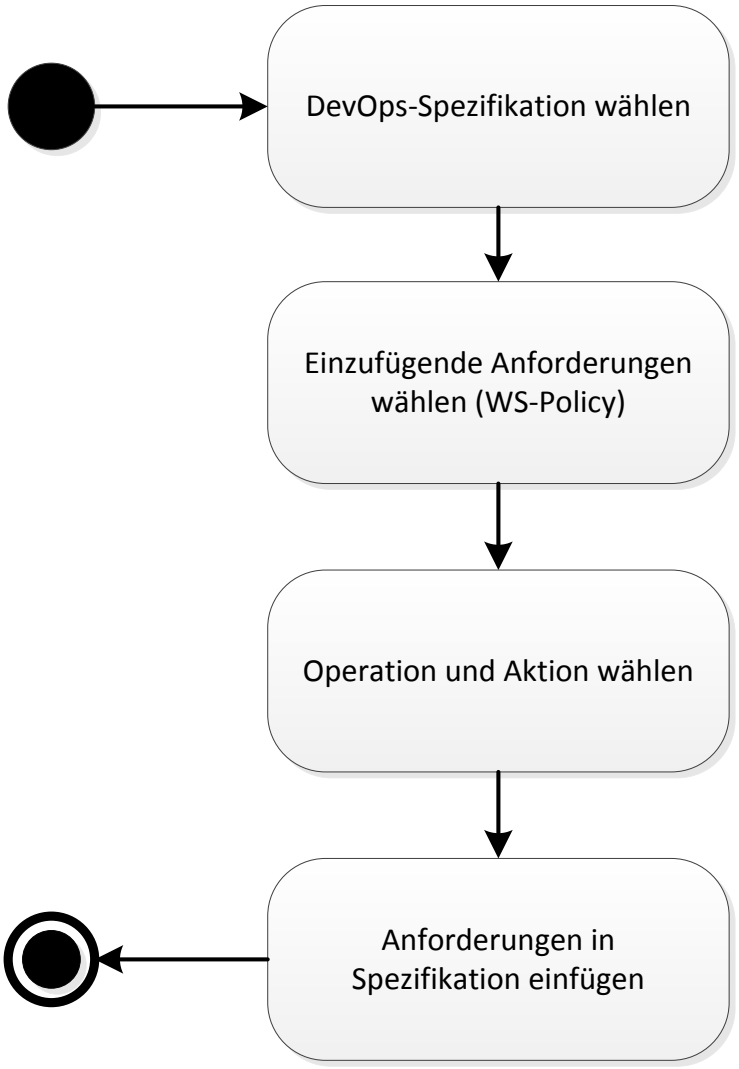


**Abbildung A.2.:** Aktivitätsdiagramm *Anforderungen einfügen*

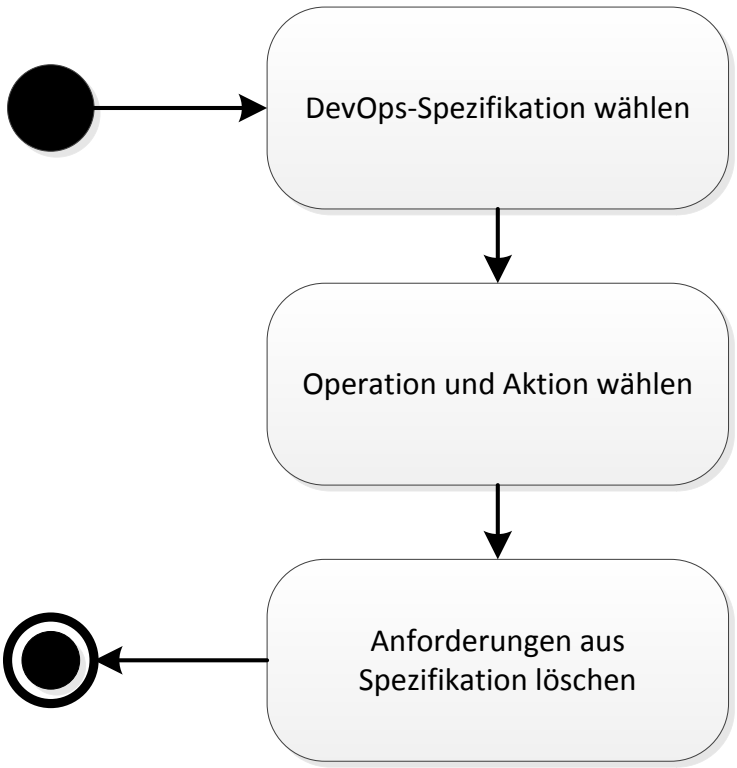


**Abbildung A.3.:** Aktivitätsdiagramm *Anforderungen löschen*

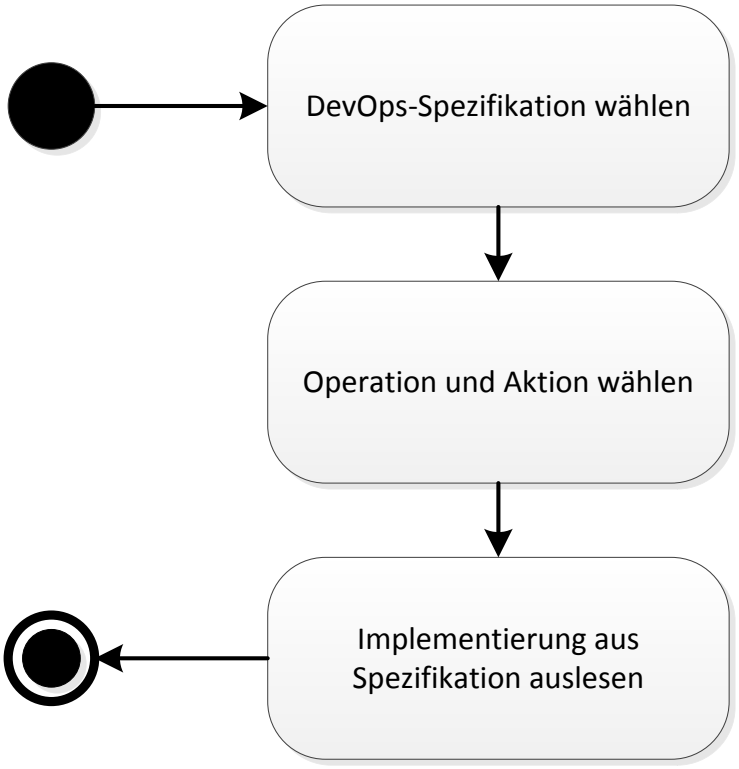


**Abbildung A.4.:** Aktivitätsdiagramm *Implementierung lesen*

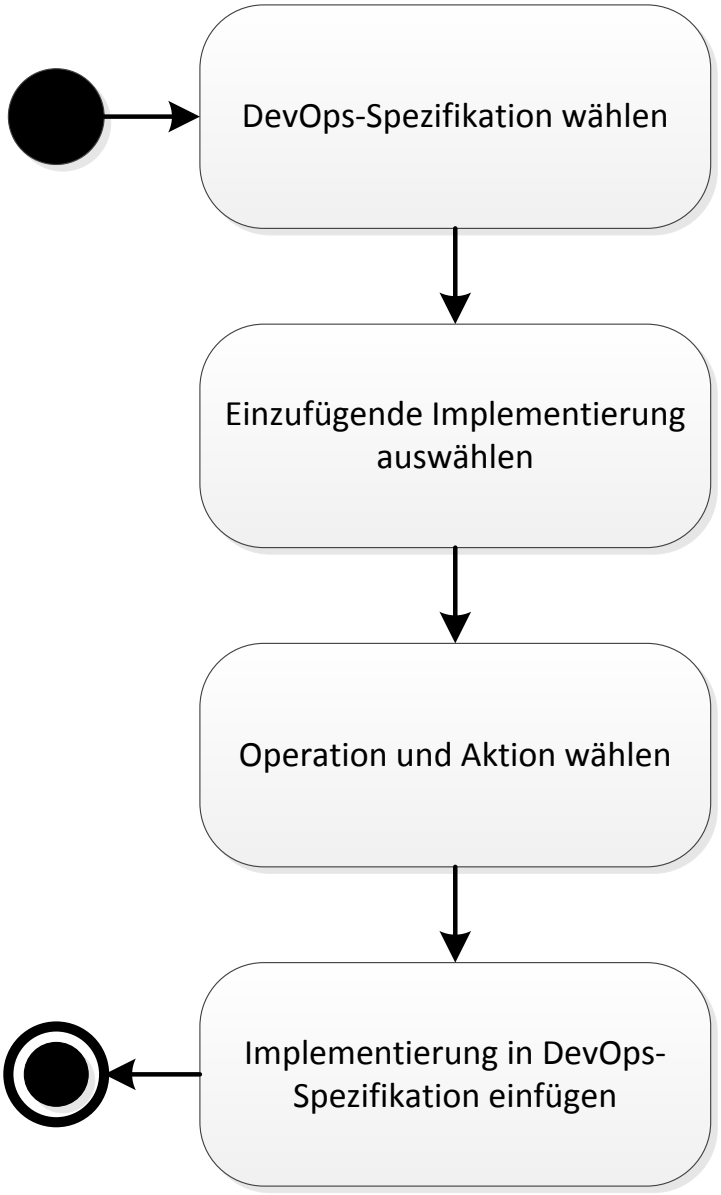


**Abbildung A.5.:** Aktivitätsdiagramm *Implementierung einfügen*

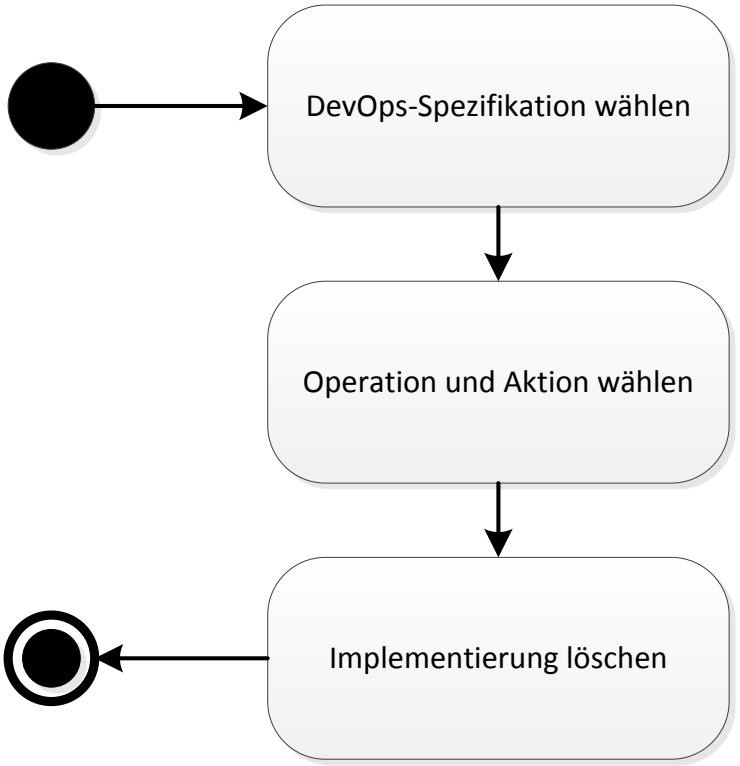


**Abbildung A.6.:** Aktivitätsdiagramm *Implementierung löschen*

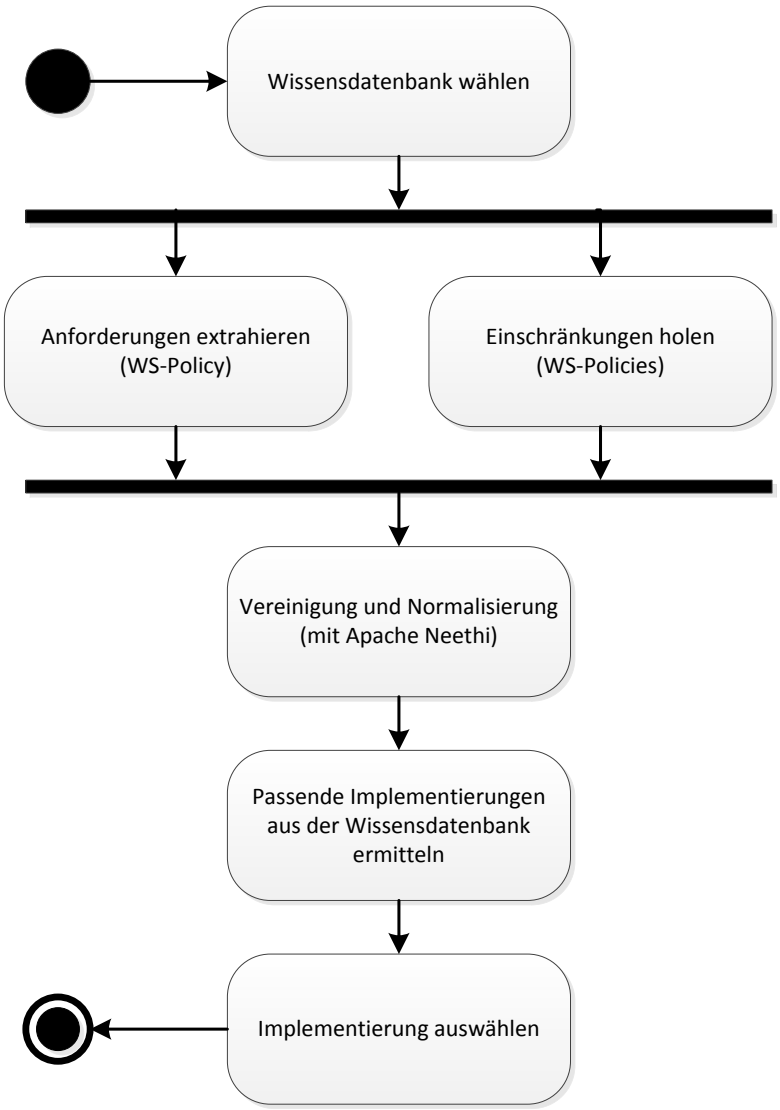


**Abbildung A.7.:** Aktivitätsdiagramm *Implementierungen aus Wissensdatenbank ermitteln*

# Literaturverzeichnis


[AEK+07] W. Arnold, T. Eilam, M. H. Kalantar, A. V. Konstantinou, A. Totok. Pattern Based SOA Deployment. In B. J. Krämer, K.-J. Lin, P. Narasimhan, Herausgeber, *ICSOC*, Band 4749 von *Lecture Notes in Computer Science*, S. 1–12. Springer, 2007. URL http://dblp.uni-trier.de/db/conf/icsoc/icsoc2007.html#ArnoldEKKT07. (Zitiert auf den Seiten 25 und 26)

[AFG+09] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technischer Bericht UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html. (Zitiert auf den Seiten 14 und 16)

[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. *Commun. ACM*, 53(4):50–58, 2010. doi:10.1145/1721654.1721672. URL http://doi.acm.org/10.1145/1721654.1721672. (Zitiert auf Seite 14)

[agi] URL http://agilemanifesto.org. (Zitiert auf Seite 22)

[amaa] URL http://aws.amazon.com/de/autoscaling. (Zitiert auf Seite 25)

[amab] URL http://aws.amazon.com/de/amazon-linux-ami/. (Zitiert auf Seite 51)

[amd] URL http://www.amd.com. (Zitiert auf Seite 17)

[AMM14] M. M. AL-Mukhtar, A. A. A. Mardan. Performance Evaluation of Private Clouds Eucalyptus versus CloudStack. *International Journal of Advanced Computer Science and Applications*, 5:108–117, 2014. (Zitiert auf Seite 17)

[ans] URL http://www.ansible.com. (Zitiert auf Seite 17)

[apaa] URL http://www.apache.org/licenses. (Zitiert auf den Seiten 16, 17 und 19)

[apab] URL http://httpd.apache.org/. (Zitiert auf Seite 20)

[apac] URL http://ant.apache.org/ivy. (Zitiert auf Seite 25)

[apad] URL http://httpd.apache.org/. (Zitiert auf Seite 35)

[apt] URL http://aptitude.alioth.debian.org/doc/en/. (Zitiert auf Seite 27)

[art] URL http://www.jfrog.com/home/v_artifactory_opensource_overview. (Zitiert auf Seite 25)

[awsa] URL http://aws.amazon.com. (Zitiert auf den Seiten 9, 15 und 24)

[awsb] URL http://aws.amazon.com/elasticbeanstalk. (Zitiert auf Seite 9)

[awsc] URL http://aws.amazon.com/ec2/. (Zitiert auf Seite 15)

[AWSd] URL http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html. (Zitiert auf Seite 25)

[Azo11] M. Azoff. DevOps: Advances in Release Management and Automation, 2011. (Zitiert auf Seite 22)

[azu] URL http://azure.microsoft.com. (Zitiert auf Seite 16)

[BBC+11] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, V. Yushprakh. Megastore: Providing Scalable, Highly Available Storage for Interactive Services. In *Proceedings of the Conference on Innovative Data system Research (CIDR)*, S. 223–234. 2011. URL http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf. (Zitiert auf Seite 16)

[BBKL14] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. *TOSCA: Portable Automated Deployment and Management of Cloud Applications*. Springer New York, 2014. (Zitiert auf Seite 17)

[bcf] URL http://bcfg2.org. (Zitiert auf Seite 17)

[bmt] URL http://en.wikipedia.org/wiki/List_of_build_automation_software. (Zitiert auf Seite 47)

[can] URL http://www.canonical.com. (Zitiert auf den Seiten 17 und 20)

[cas] URL http://cassandra.apache.org/. (Zitiert auf Seite 20)

[Cat11] R. Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39(4):12–27, 2011. doi:10.1145/1978915.1978919. URL http://doi.acm.org/10.1145/1978915.1978919. (Zitiert auf Seite 25)

[cep] URL http://ceph.com/. (Zitiert auf Seite 20)

[cfe] URL https://cfengine.com. (Zitiert auf Seite 17)

[che] URL http://docs.getchef.com/chef_quick_overview.html. (Zitiert auf den Seiten 6 und 19)

[cloa] URL https://cloud.google.com/products/cloud-dns. (Zitiert auf Seite 9)

[clob] URL http://cloud-standards.org. (Zitiert auf Seite 15)

[cloc] URL https://cloud.google.com/products/app-engine/. (Zitiert auf Seite 16)

[clod] URL http://cloudstack.apache.org/. (Zitiert auf Seite 16)

[cloe] URL http://cloudfoundry.org. (Zitiert auf Seite 24)

[cmm] URL http://cmmiinstitute.com. (Zitiert auf Seite 23)

[Coc12] A. Cockcroft. Ops, DevOps and PaaS (NoOps) at Netflix. Webblog, 2012. URL http://perfcap.blogspot.de/2012/03/ops-devops-and-noops-at-netflix.html. (Zitiert auf Seite 25)

[Coh09] M. Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 2009. ISBN 978-0321579362. (Zitiert auf Seite 22)

[com] URL https://cloud.google.com/products/compute-engine. (Zitiert auf Seite 9)

[Cor11] I. Corporation. Getting Cloud Computing Right, 2011. (Zitiert auf Seite 13)

[Deb11] P. Debois. Devops: A Software Revolution in the Making, OpeOpen Statement. *Cutter IT Journal*, 24:3–5, 2011. (Zitiert auf Seite 10)

[del] URL http://www.dell.com. (Zitiert auf Seite 17)

[dev] URL https://developers.google.com/appengine/?csw=1. (Zitiert auf Seite 9)

[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'10, S. 1–8. USENIX Association, Berkeley, CA, USA, 2010. URL http://dl.acm.org/citation.cfm?id=1924976.1924977. (Zitiert auf Seite 17)

[DMTF09] I. D. Distributed Management Task Force. Open Virtualization Format Specification. Specification, 2009. (Zitiert auf Seite 15)

[doc] URL https://www.docker.com/. (Zitiert auf Seite 10)

[dom] URL http://www.w3.org/DOM/. (Zitiert auf Seite 53)

[ets] URL https://www.etsy.com. (Zitiert auf Seite 24)

[Fab13] A. Fabbro. Linux on Azure-a Strange Place to Find a Penguin. *Linux J.*, 2013(226), 2013. URL http://dl.acm.org/citation.cfm?id=2457428.2457430. (Zitiert auf Seite 16)

[fac] URL https://www.facebook.com. (Zitiert auf Seite 24)

[Fel13] B. Feld. Why Every Company Need A DevOps Team Now, 2013. (Zitiert auf Seite 24)

[Fin14] K. Finley. NoOps, AppOps and DevOps and your Business. How do They Apply for You? Webblog, 2014. URL http://siliconangle.com/blog/2012/03/29/noops-appops-and-devops-and-your-business-how-do-they-apply-for-you/. (Zitiert auf Seite 25)

[get] URL http://www.getchef.com/chef. (Zitiert auf den Seiten 9 und 17)

[git] URL https://github.com/eucalyptus/eucalyptus/wiki. (Zitiert auf Seite 16)

[gnu] URL http://www.gnu.org/licenses/gpl-2.0.html. (Zitiert auf Seite 11)

[goo] URL https://www.google.com. (Zitiert auf Seite 24)

[gpl] URL http://www.gnu.org/copyleft/gpl.html. (Zitiert auf Seite 35)

[gso] URL https://code.google.com/p/google-gson/. (Zitiert auf Seite 47)

[Hü12] M. Hüttermann. *DevOps For Developers*. Apress, 2012. (Zitiert auf den Seiten 6, 17, 22 und 23)

[had] URL http://hadoop.apache.org/. (Zitiert auf Seite 20)

[Har12] D. Harris. Why 2013 is the year of 'NoOps' for programmers [Infographic]. Webseite, 2012. URL http://gigaom.com/2012/01/31/why-2013-is-the-year-of-noops-for-programmers-infographic/. (Zitiert auf Seite 25)

[HF10a] J. Humble, D. Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 1st Auflage, 2010. (Zitiert auf Seite 17)

[HF10b] J. Humble, D. Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison Wesley, 2010. (Zitiert auf Seite 17)

[hp.] URL http://www.hp.com. (Zitiert auf Seite 17)

[hpc] URL http://www.hpcloud.com. (Zitiert auf Seite 20)

[int] URL http://www.intel.com. (Zitiert auf Seite 17)

[iti] URL http://www.itil-officialsite.com. (Zitiert auf Seite 23)

[JA09] P. H. John Allspaw. 10 deploys per day Dev and ops cooperation at Flickr. In *Velocity Presentation*. 2009. (Zitiert auf Seite 9)

[jar] URL http://docs.oracle.com/javase/6/docs/technotes/guides/jar/index.html. (Zitiert auf Seite 60)

[jdo] URL http://www.jdom.org/. (Zitiert auf Seite 47)

[jen] URL http://jenkins-ci.org. (Zitiert auf Seite 25)

[JH11] J. M. Jez Humble. Why Enterprises Must Adopt Devops to Enable Continuous Delivery. CUTTER IT JOURNAL, 2011. (Zitiert auf Seite 23)

[jsoa] URL https://code.google.com/p/json-simple/. (Zitiert auf Seite 47)

[jsob] URL http://json.org/. (Zitiert auf den Seiten 47 und 54)

[juja] URL https://juju.ubuntu.com/deployment/. (Zitiert auf den Seiten 6, 20 und 21)

[jujb] URL http://jujucharms.com/bundle/mediawiki. (Zitiert auf den Seiten 7 und 28)

[jujc] URL https://juju.ubuntu.com. (Zitiert auf den Seiten 9 und 17)

[jujd] URL https://jujucharms.com. (Zitiert auf Seite 27)

[KB13] J. D. l. R. Kent Baxley. DepDeploy workloads with Juju and MAAS in Ubuntu 13.04, 2013. (Zitiert auf Seite 20)

[kvm] URL http://www.linux-kvm.org/page/Management_Tools. (Zitiert auf Seite 20)

[LC12] A. C. Lucas Carlson. What is NoOps anyhow? Webblog, 2012. URL http://blog.appfog.com/what-is-noops-anyhow/. (Zitiert auf Seite 24)

[Ley04] F. Leymann. The Influence of Web Services on Software: Potentials and Tasks. In P. Dadam, M. Reichert, Herausgeber, *INFORMATIK 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, Band 50 von *LNI*, S. 14–25. GI, 2004. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2004-84&engl=0. (Zitiert auf Seite 14)

[Ley09] F. Leymann. Cloud Computing: The Next Revolution in IT. In *Proc. 52th Photogrammetric Week*, S. 3–12. Wichmann Verlag, 2009. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2009-65&engl=0. (Zitiert auf den Seiten 6, 9, 14 und 15)

[lin] URL http://aws.amazon.com/amazon-linux-ami. (Zitiert auf Seite 25)

[Lou12] M. Loukides. What is DevOps? Technischer Bericht, O'Reilly Media, 2012. URL http://shop.oreilly.com/product/0636920026822.do?intcmp=il-velocity-books-what-is-devops-radar-edition. (Zitiert auf Seite 24)

[LS10] W. Lehner, K.-U. Sattler. Database as a service (DBaaS). In F. Li, M. M. Moro, S. Ghandeharizadeh, J. R. Haritsa, G. Weikum, M. J. Carey, F. Casati, E. Y. Chang, I. Manolescu, S. Mehrotra, U. Dayal, V. J. Tsotras, Herausgeber, *ICDE*, S. 1216–1217. IEEE, 2010. (Zitiert auf Seite 9)

[LSS+13] H. Lu, M. Shtern, B. Simmons, M. Smit, M. Litoiu. Pattern-based Deployment Service for Next Generation Clouds. In *IEEE 9th World Congress on Services, Cloud Cup*. 2013. (Zitiert auf Seite 26)

[lxc] URL https://linuxcontainers.org/. (Zitiert auf Seite 20)

[maa] URL https://maas.ubuntu.com. (Zitiert auf Seite 20)

[Mad10] L. Madeyski. *Test-Driven Development - An Empirical Evaluation of Agile Practice*. Springer Berlin Heidelberg, 2010. (Zitiert auf Seite 17)

[man] URL http://www.mantisbt.org. (Zitiert auf den Seiten 11 und 35)

[mava] URL http://maven.apache.org/. (Zitiert auf Seite 47)

[mavb] URL http://maven.apache.org/ref/3.2.3/maven-artifact/. (Zitiert auf Seite 47)

[med] URL https://www.mediawiki.org. (Zitiert auf Seite 27)

[MG11] P. Mell, T. Grance. The NIST Definition of Cloud Computing. Technischer Bericht 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, 2011. URL http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. (Zitiert auf den Seiten 13 und 15)

[MHT11] A. S. Michael Hogan, Fang Liu, J. Tong. NIST Cloud Computing Standards Roadmap, 2011. (Zitiert auf Seite 15)

[MLM11] D. Milojicic, I. M. Llorente, R. S. Montero. OpenNebula: A Cloud Management Tool. *IEEE Internet Computing*, 15(2):11–14, 2011. doi:http://doi.ieeecomputersociety.org/10.1109/MIC.2011.44. (Zitiert auf Seite 16)

[msc] URL http://msdn.microsoft.com/de-de/library/8bs2ecf4(v=vs.110).aspx. (Zitiert auf Seite 16)

[msn] URL http://www.microsoft.com/net. (Zitiert auf Seite 16)

[mys] URL http://www.mysql.com. (Zitiert auf den Seiten 11 und 20)

[nas] URL http://www.nasa.gov. (Zitiert auf Seite 17)

[net] URL https://www.netflix.com. (Zitiert auf den Seiten 24 und 25)

[nis] URL http://www.nist.gov. (Zitiert auf Seite 15)

[oas] URL https://www.oasis-open.org/. (Zitiert auf Seite 15)

[occ] URL http://occi-wg.org. (Zitiert auf den Seiten 15 und 16)

[ogf] URL https://www.ogf.org. (Zitiert auf Seite 16)

[OJ11] S. Ortiz Jr. The Problem with Cloud-Computing Standardization. *Computer*, 44(7):13–16, 2011. doi:10.1109/MC.2011.220. URL http://dx.doi.org/10.1109/MC.2011.220. (Zitiert auf Seite 15)

[opea] URL https://www.openstack.org. (Zitiert auf den Seiten 9 und 16)

[opeb] URL http://opennebula.org. (Zitiert auf Seite 16)

[Pan12] S. Pandey. Investigating Community, Reliability and Usability of CFEngine, Chef and Puppet, 2012. (Zitiert auf Seite 19)

[PCS+13] R. Pooley, J. Coady, C. Schneider, H. Linger, C. Barry, M. Lang. *Information Systems Development - Reflections, Challenges and New Directions*. Springer, 2013. (Zitiert auf Seite 10)

[per] URL http://www.perforce.com. (Zitiert auf Seite 25)

[php] URL http://www.php.net. (Zitiert auf Seite 35)

[pupa] URL https://docs.puppetlabs.com/learning/agent_master_basic.html. (Zitiert auf den Seiten 6 und 18)

[pupb] URL http://puppetlabs.com. (Zitiert auf den Seiten 9 und 17)

[PW11] K. Petersen, C. Wohlin. Measuring the Flow in Lean Software Development. *Softw. Pract. Exper.*, 41(9):975–996, 2011. doi:10.1002/spe.975. URL http://dx.doi.org/10.1002/spe.975. (Zitiert auf Seite 9)

[rac] URL http://www.rackspace.com. (Zitiert auf Seite 17)

[Rah12] M. J. Rahman. Investigating Configuration Management Tools Usage in Large Infrastructure, 2012. (Zitiert auf Seite 18)

[s3] URL http://aws.amazon.com/de/s3/. (Zitiert auf Seite 16)

[Sac12] M. Sacks. *Pro Website Development and Operations: Streamlining DevOps for large-scale websites*. Apress, 2012. (Zitiert auf Seite 10)

[SAE12] O. Sefraoui, M. Aissaoui, M. Eleuldj. Article: OpenStack: Toward an Open-source Solution for Cloud Computing. *International Journal of Computer Applications*, 55(3):38–42, 2012. Published by Foundation of Computer Science, New York, USA. (Zitiert auf Seite 17)

[sal] URL http://www.saltstack.com. (Zitiert auf Seite 17)

[Sch00] U. Schöning. *Logik für Informatiker*. Spektrum Akademischer Verlag, 2000. (Zitiert auf Seite 41)

[sem] URL http://semver.org/. (Zitiert auf Seite 50)

[Sha11] A. Shalloway. Demystifying Kanban, 2011. (Zitiert auf Seite 22)

[Smi11] D. M. Smith. Hype Cycle for Cloud Computing, 2011, 2011. (Zitiert auf Seite 23)

[ST10] P. Sempolinski, D. Thain. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, S. 417–426. IEEE Computer Society, Washington, DC, USA, 2010. doi:10.1109/CloudCom.2010.42. URL http://dx.doi.org/10.1109/CloudCom.2010.42. (Zitiert auf Seite 16)

[sta] URL https://jcp.org/en/jsr/detail?id=173. (Zitiert auf Seite 53)

[sur] URL https://www.surfsara.nl. (Zitiert auf Seite 16)

[tel] URL http://www.telefonica.com. (Zitiert auf Seite 16)

[TEM07] V. Tosic, A. Erradi, P. Maheshwari. WS-Policy4MASC - A WS-Policy Extension Used in the MASC Middleware. In *IEEE SCC*, S. 458–465. 2007. (Zitiert auf Seite 26)

[Tiw00] A. Tiwana. *The Knowledge Management Toolkit: Practical Techniques for Building a Knowledge Management System*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000. (Zitiert auf Seite 33)

[tom] URL http://tomcat.apache.org/. (Zitiert auf Seite 20)

[TS10] U. H. Thomas Stober. *Agile Software Development*. Springer Berlin Heidelberg, 2010. (Zitiert auf Seite 9)

[twi] URL https://twitter.com. (Zitiert auf Seite 24)

[ubua] URL http://www.ubuntu.com. (Zitiert auf Seite 20)

[ubub] URL http://www.ubuntu.com/server. (Zitiert auf Seite 20)

[vag] URL http://www.vagrantup.com. (Zitiert auf Seite 10)

[vcl] URL https://www.vmware.com/products/vcloud-suite/. (Zitiert auf Seite 16)

[Ver13] VersionOne. 7th Annual State of Agile Development Survey, 2013. URL http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf. (Zitiert auf Seite 22)

[vmw] URL http://www.vmware.com. (Zitiert auf Seite 16)

[VOH+07a] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, U. Yalçınalp. Web Services Policy 1.5 — Framework. World Wide Web Consortium, Recommendation REC-ws-policy-20070904, 2007. (Zitiert auf den Seiten 47 und 48)

[VOH+07b] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, U. Yalçınalp. Web Services Policy 1.5 — Primer. World Wide Web Consortium, Note NOTE-ws-policy-primer-20071112, 2007. (Zitiert auf den Seiten 6, 48, 49 und 50)

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. (Zitiert auf den Seiten 14 und 47)

[wor] URL http://wordpress.org. (Zitiert auf Seite 20)

[ws.] URL http://ws.apache.org/neethi/. (Zitiert auf den Seiten 47 und 53)

[ws-] URL http://www.w3.org/TR/ws-policy. (Zitiert auf Seite 15)

[wsb] URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. (Zitiert auf Seite 26)

[wsd] URL http://www.w3.org/TR/wsdl20. (Zitiert auf den Seiten 26 und 47)

[wsp] URL http://www.w3.org/TR/ws-policy-attach/. (Zitiert auf Seite 47)

[xml] URL http://www.w3.org/TR/REC-xml-names/. (Zitiert auf Seite 49)

Alle URLs wurden zuletzt am 20. 09. 2014 geprüft.

**Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift