Universität Stuttgart

Konzepte zur Verbesserung des Monitoring in industriellen Cloud-Umgebungen

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

Vorgelegt von

Mathias Mormul

aus Goslar

Hauptberichter: Prof. Dr. Bernhard Mitschang

Mitberichter: Prof. Dr. Stefan Deßloch

Tag der mündlichen Prüfung: 20.01.2022

Institut für Parallele und Verteilte Systeme

2022

Inhaltsverzeichnis

Zusammenfassung

IT-Landschaften haben sich seit der Jahrtausendwende dramatisch verändert. Neue Paradigmen wie Cloud-Computing, Industrie 4.0, und Edge-Computing führen zu disruptiven Veränderungen und tragen zu einer stetigen Zunahme der Komplexität der IT-Landschaften bei. Während zuvor starre und monolithische Strukturen vorherrschten, sind IT-Landschaften heute hochdynamisch und verteilt, was in erster Linie auf den Einsatz von Cloud Computing zurückzuführen ist. An die Stelle von physischen Servern treten virtuelle Maschinen, die innerhalb von Minuten zur Verfügung gestellt werden können und deren Kapazitäten kostengünstig an die Bedürfnisse des Unternehmens angepasst werden können. Gleichzeitig finden sich auf dem Shopfloor intelligenter Fabriken cyber-physische Systeme (CPS), die über den Zusammenschluss von Maschinen und Software-Komponenten entstehen. Die Auswertung von Sensordaten dieser CPS unterliegt oft strengen Latenzanforderungen und erfolgt daher auf Rechenressourcen, die entweder direkt mit den CPS verbunden sind oder netzwerktechnisch diesen sehr nahe liegen. Dies wird als Edge-Computing bezeichnet und führt unter Einsatz von Cloud-Technologien zu Edge-Clouds. Die Vernetzung all dieser Technologien führt zu hochkomplexen IT-Landschaften, die im Rahmen dieser Arbeit als industrielle Cloud-Umgebungen bezeichnet werden. Um der hohen Dynamik und Komplexität dieser industriellen Cloud-Umgebungen entgegenzutreten,

sind Systeme für das Monitoring unerlässlich. Es existiert eine Vielzahl solcher Systeme, die verschiedene Aspekte dieser Umgebungen überwachen. Jedoch sind diese den steigenden Latenzanforderungen und stets wachsenden Datenmengen oftmals nicht gewachsen. Ein Austausch der Systeme ist mit einem hohen Kosten- und Zeitaufwand verbunden. Zudem führt der gleichzeitige Einsatz mehrerer Monitoring-Systeme zwangsläufig zu Datensilos und erlaubt nur eine isolierte Sicht auf die jeweiligen überwachten Teilbereiche der industriellen Cloud-Umgebung. Daher werden im Rahmen dieser Dissertation Konzepte zur Verbesserung des Monitoring in industriellen Cloud-Umgebungen vorgestellt. Dazu werden zunächst individuelle Lösungen im Bereich des Cloud-Monitoring sowie des Monitoring industrieller Maschinen vorgestellt, welche bestehende Monitoring-Systeme aus der Literatur und der Praxis verbessern sollen. Im Bereich des Cloud-Monitoring werden (1) generische Agententemplates vorgestellt, die eine Abstraktionsstufe für das Modellieren von Agenten erzeugen, wodurch der Austausch von Monitoring-Systemen erleichtert werden soll. Darauf aufbauend wird (2) DEAR vorgestellt, ein Plug-in für die verteilte Auswertung von Alerting-Regeln in agentenbasierten Monitoring-Systemen. Das Ziel ist eine Reduktion des Netzwerkverkehrvolumens bzgl. der Monitoring-Daten bei gleichbleibender Qualität des Monitoring. Daraufhin wird (3) der Ansatz der Situationserkennung für das Monitoring industrieller Maschinen auf der Modellierungs- und Ausführungsebene erweitert, um eine verteilte Situationserkennung zu ermöglichen und Anforderungen bzgl. Latenz gerecht zu werden. Letztlich soll (4) ein umfassendes Kontextmodell für das Monitoring industrieller Cloud-Umgebungen ermöglicht werden. Dabei sollen alle für das Monitoring relevanten Aspekte berücksichtigt werden, um Systemadministratoren beim Monitoring zu unterstützen. Die Konzepte dieser Dissertation werden durch zugehörige Publikationen in Konferenzbeiträgen und Fachmagazinen gestützt und durch prototypische Implementierungen validiert.

Abstract

IT landscapes have changed dramatically since the turn of the millennium. New paradigms, such as cloud computing, Industry 4.0, and edge computing, lead to disruptive changes and contribute to a steady increase in the complexity of IT landscapes. While rigid and monolithic structures used to predominate, IT landscapes today are highly dynamic and distributed, which is primarily due to the use of cloud computing. Physical servers are being replaced by virtual machines that can be made available within minutes and whose capacities can be adapted to the needs of the company at low costs. At the same time, cyber-physical systems (CPS) are emerging on the shop floors of intelligent factories, which are created by combining machines and software components. The evaluation of sensor data from these CPS is often subject to strict latency requirements and is therefore performed on computing resources that are either directly connected to the CPS or very close to them in terms of network distance. This is known as edge computing and leads to edge clouds using cloud technologies. The interconnections of all these technologies leads to highly complex IT landscapes, which in the context of this work are referred to as industrial cloud environments. To counter the high dynamics and complexity of these industrial cloud environments, systems for monitoring are essential. There exists a variety of systems that monitor different aspects of these environments. However, these

systems are often not able to cope with the increasing latency requirements and ever growing data volumes. Replacing these systems is costly and time-consuming. In addition, the simultaneous use of several monitoring systems inevitably leads to data silos. This allows only an isolated view of the respective, monitored sub-areas of the industrial cloud environment. Therefore, this dissertation presents concepts to improve the monitoring of industrial cloud environments. For this purpose, individual solutions in the field of cloud monitoring as well as the monitoring of industrial machines are first presented, which are intended to improve existing monitoring systems from literature and practice. In the field of cloud monitoring, (1) generic agent templates are presented, which create an abstraction level for modeling agents, which facilitates the exchange of monitoring systems. Based on this, (2) DEAR is presented, a plug-in for the distributed evaluation of alerting rules in agent-based monitoring systems. The goal is to reduce the network traffic volume with respect to monitoring data while maintaining the same quality of monitoring. Thereupon (3) the situation detection approach for the monitoring of industrial machines is extended on the modeling and execution level to enable a distributed situation detection and to meet requirements regarding latency. Finally, (4) a comprehensive context model for monitoring of industrial cloud environments shall be created. All aspects relevant for monitoring are to be taken into account in order to support system administrators in monitoring. The concepts of this dissertation are supported by related publications in conference papers and journals and validated by prototypical implementations.

Danksagungen

Ich möchte allen danken, die mich bei der Promotion unterstützt haben. Hierfür möchte ich mich zuerst bei meinem Doktorvater Prof. Dr. Bernhard Mitschang für die fortlaufende und stets wertvolle Unterstützung bedanken. Des Weiteren gilt mein Dank Prof. Dr. Stefan Deßloch, der sich bereiterklärt hat, das Zweitgutachten für diese Dissertation zu übernehmen. Selbstverständlich bedanke ich mich für die vielen Kollegen der Abteilung AS, die bei gemütlichen Mittags- und Kaffeepausen für eine wundervolle Arbeitsatmosphäre gesorgt haben und gleichzeitig mit inhaltlichen Diskussionen mich persönlich sowie meine Forschung bereichert haben. Besonderer Dank gilt Dr. Pascal Hirmer, Dr. Christoph Stach, Dr. Christian Weber, Manuel Fritz, und Dr. Matthias Wieland, die mir stets wertvolles Feedback gegeben haben und mit denen ich meine wissenschaftlichen Veröffentlichungen verfassen durfte. Des Weiteren bedanke ich mich bei den vielen Studenten, die mich bei der Arbeit unterstützt haben. Zuletzt bedanke ich mich bei meinen Freunden und meiner Familie für deren fortlaufende Unterstützung und deren Vertrauen in mich und meine Arbeit.

Abkürzungsverzeichnis

AIOps	Artificial Intelligence for IT Operations
APM	Application Performance Monitoring
ARIMA	Autoregressive Integrated Moving Average
BET	Binary Expression Tree
CEP	Complex Event Processing
CPS	Cyber-physisches System
CPPS	Cyber-physisches Produktionssystem
CPU	Central Processing Unit
DBaaS	Database as a Service
DEAR	Distributed Evaluation of Alerting Rules
ELT	Extract Load Transform
ETL	Extract Transform Load
FaaS	Function as a Service
I/O	Input/Output
I4.0	Industrie 4.0
IaaS	Infrastructure as a Service
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
IT	Informationstechnik

ITIM IT Infrastructure Monitoring
JSON JavaScript Object Notation
LSTM Long Short Term Memory
MES Manufacturing Execution System
MQTT Message Queuing Telemetry Transport
NIST National Institute of Standards and Technology
OASIS Organization for the Advancement of Structured Information Standards
PaaS Platform as a Service
QoS Quality of Service
RAM Random Access Memory
RDBMS Relationales Datenbank-Management-System
RDF Resource Description Framework
RDFS Resource Description Framework Schema
SaaS Software as a Service
SLA Service Level Agreement
SPoA Single Point of Administration
SQL Structured Query Language
STMT Situation Templates Modeling Tool
TCP Transmission Control Protocol
TOSCA Topology and Orchestration Specification for Cloud Applications
TSDB Time Series Database
TTI Time-to-Insight
UDP User Datagram Protocol
URI Uniform Resource Identifier
VM Virtuelle Maschine
XML Extensive Markup Language

Kapitel 1

Einführung

> *Die einzige Konstante im Universum ist die Veränderung.*
>
> —*Heraklit*

1.1 Motivation

IT-Landschaften haben sich seit der Jahrtausendwende dramatisch verändert. Neue Paradigmen wie Cloud-Computing und Industrie 4.0 (I4.0) führen zu disruptiven Änderungen und tragen zu einer stetigen Zunahme der Komplexität der IT-Landschaften bei. Während zuvor starre und monolithische Strukturen vorherrschten, sind IT-Landschaften heute vor allem durch den Einsatz von Cloud-Computing sehr dynamisch und verteilt. An die Stelle von physischen Servern treten virtuelle Maschinen (VMs), die innerhalb von Minuten zur Verfügung gestellt werden können und deren Kapazitäten kosteneffizient an die Bedürfnisse des Unternehmens angepasst werden können [MLB+11]. Auf diese Weise können Unternehmen ohne hohe Kosten für Hardware hochskalierbare IT-Umgebungen erschaffen, die je nach

Bedarf schnell vergrößert oder verkleinert werden können. Die bereitgestellten Ressourcen können global verteilt werden, um Zugriffszeiten für Verbraucher rund um den Globus zu minimieren. Hierfür existieren mehrere Service-Modelle, welche die vom Anbieter bereitgestellten Cloud-Ressourcen ursprünglich in Infrastruktur (*Infrastructure as a Service (IaaS)*), Plattform (*Platform as a Service (PaaS)*, und Anwendungen *Software as a Service (SaaS)* unterteilten. Diese Cloud-Ressourcen werden dem Verbraucher über verschiedene Bereitstellungsmodelle (z. B. öffentlicher oder exklusiver Zugriff) zur Verfügung gestellt [MG11].

Über die Jahre ist eine Vielzahl weiterer, spezialisierter Service-Modelle entstanden. Hierzu zählen beispielsweise *Function as a Service (FaaS)* [Rob18], das auch unter dem Begriff *Serverless Computing* die Ausführung von Funktionen in der Cloud beschreibt oder *Database as a Service (DBaaS)* [LK18], das dem Verbraucher cloud-basierte Datenbanksysteme zur Verfügung stellt. Die weitverbreitete Nutzung von Cloud-Ressourcen hat Unternehmen zunehmend abhängiger von deren Verfügbarkeit gemacht. Um Ausfällen einzelner Cloud-Anbieter entgegenzuwirken, ist inzwischen die Verwendung von Multi-Clouds, d. h., die gleichzeitige Nutzung von Cloud-Ressourcen mehrerer Anbieter, ein übliches Vorgehen [Pet13]. Demnach verfolgen einer Umfrage zufolge im Jahr 2020 93 % der befragten Unternehmen einen Multi-Cloud-Ansatz [Wei20].

Es wird ersichtlich, dass im Vergleich zu traditionellen, monolithischen IT-Umgebungen das Cloud-Computing Unternehmen jeder Größe zu der Erzeugung hochkomplexer, verteilter Cloud-Umgebungen befähigt, die sich jederzeit schnell und dynamisch an die stets ändernden Bedürfnisse der Unternehmen anpassen können. Um diese Komplexität und Dynamik kontrolliert und effizient handhaben zu können, ist ein Monitoring (dt.: Überwachung) der Cloud-Umgebungen von essentieller Bedeutung [ABdP13]. Daher kommen Monitoring-Systeme zum Einsatz, die alle Cloud-Ressourcen überwachen und Systemadministratoren bei erkannten Problemen alarmieren, um eine zuverlässige Cloud-Umgebung zu ermöglichen. Dieser Prozess wird als *Alerting* (dt.: Alarmieren) bezeichnet. Auf Basis von vordefinierten

Alerting-Regeln können Bedingungen definiert werden (z. B. *wenn CPU-Auslastung über 90 %*), bei deren Eintreten bestimmte Aktionen ausgeführt werden (z. B. *sende E-Mail an Systemadministrator*).

Für das Monitoring werden diverse Monitoring-Systeme eingesetzt, um verschiedene Aspekte einer Cloud-Umgebung zu überwachen. Beispielsweise existieren Systeme für IT-Infrastruktur-Monitoring (ITIM) zur Überwachung der Cloud-Infrastruktur (z. B. Auslastung der VMs oder des Speichers), Application-Performance-Monitoring (APM; dt.: Performanz von Anwendungen) zur Überwachung diverser Anwendungen oder Netzwerk-Monitoring zur Überwachung des Netzwerks zwischen den Cloud-Ressourcen. In jedem dieser Segmente befinden sich mehrere Anbieter, aus denen die Verbraucher auswählen können. Dies führt unweigerlich dazu, dass Unternehmen mehrere Monitoring-Systeme parallel betreiben müssen, um eine Cloud-Umgebung vollständig zu überwachen. Studien zeigen, dass viele Unternehmen mehr als zehn Monitoring-Systeme parallel betreiben [Big19]. Diese Monitoring-Systeme müssen in die zu überwachende Cloud-Umgebung integriert werden, um detaillierte Informationen darüber erfassen zu können.

Die meisten aktuellen Monitoring-Systeme sind jedoch ursprünglich nicht für das Monitoring von Cloud-Umgebungen mit hochdynamischen und schnellen Veränderungen konzipiert worden, wie z. B. VMs, die innerhalb weniger Minuten provisioniert und wieder deprovisioniert werden können. Stattdessen wurden diese eher für traditionelle Hardware und sich langsam verändernde Umgebungen verwendet [CA15; TPD15]. Wenn sich die Geschäftsanforderungen ändern und damit eingehend Veränderungen in der Cloud-Umgebung stattfinden, könnte der Einsatz eines neuen Monitoring-Systems, das diesen Geschäftsanforderungen besser entspricht, vorteilhaft sein. Beispielsweise unterstützen in den letzten Jahren immer mehr Monitoring-Systeme fortschrittliche Analysemethoden, z.B. neuronale Netzwerke und Clustering, um einen tieferen Einblick in Cloud-Umgebungen zu ermöglichen und bspw. zuvor unbekannte Abhängigkeiten der Cloud-Ressourcen zu erkennen oder präventiv gegen potentiell auftretende Probleme vorzugehen. Der Austausch eines Monitoring-Systems wird jedoch dadurch erschwert, dass IT-Abteilungen oft unterbesetzt sind und ihnen die Ressour-

cen für neue Technologien fehlen [Bow18; Tec17]. Insbesondere in groß angelegten Cloud-Umgebungen gibt es immer wieder Ausfälle von virtuellen oder physischen Ressourcen, die behoben werden müssen und die Zeit der IT-Abteilungen stark in Anspruch nehmen. Daher ist es schwer, mit den neuesten Technologien Schritt zu halten und sie im Unternehmen einzusetzen. Dies wiederum führt zu einem Lock-in-Effekt hinsichtlich der verwendeten Monitoring-Systeme, da die zeitlichen und finanziellen Kosten den Austausch eines Monitoring-Systems verhindern.

Ein weiteres Problem des Cloud-Monitoring stellt das große Datenvolumen der Monitoring-Daten dar, das gemeinsam mit größer werdenden Cloud-Umgebungen stets anwächst und die Netzwerkbandbreite stark belastet [Har17]. Monitoring-Daten werden auf den überwachten Ressourcen, z. B. VMs, von sogenannten Agenten erfasst und üblicherweise an eine zentrale Datenbank des Monitoring-Systems gesendet [ARM+15; HW18]. Die gängige Methode zur Reduzierung der Datenmengen ist die Aggregation der Monitoring-Daten durch die Agenten, bevor diese an die zentrale Datenbank gesendet werden [ABdP13]. Allerdings führt eine Aggregation zwangsläufig zu einem Informationsverlust und einer geringeren Genauigkeit des Monitoring [TJT+18; TPD15]. Eine Alternative ist es, die Zeitabstände zu erhöhen, in denen die Monitoring-Daten von den Agenten erfasst werden. Doch während in der Vergangenheit Zeitabstände von 30 Sekunden oder mehr akzeptabel erschienen, ist heutzutage eine schnelle Reaktion auf auftretende Probleme obligatorisch [VT14; WB14].

Parallel dazu führt das Auftreten von I4.0 zu ähnlich disruptiven Folgen in der verarbeitenden Industrie, wie es mit Cloud-Computing für die IT-Welt der Fall war. Die vom Bundesministerium für Bildung und Forschung[1] ins Leben gerufene Hightech-Strategie zur Digitalisierung in der verarbeitenden Industrie [Bun20] umfasst eine Vielzahl von Projekten, die alle zum Ziel haben, die industrielle Produktion mit aktuellen Informations- und Kommunikationstechnologien auszustatten. Durch eine umfassende sensorische Überwachung werden Umgebungs- und Prozessdaten von Assets erfasst. Ein

[1] https://www.bmbf.de/

Asset wird als *„alles, was verbunden werden muss, um eine I4.0-Lösung zu erzeugen“* bezeichnet [Pla19]. Zur deutlicheren Abgrenzung zu den Cloud-Ressourcen, die in dieser Arbeit ebenfalls Ziel des Monitoring sind, wird im Rahmen dieser Arbeit der Begriff *industrielles Asset* verwendet.

Die Vernetzung von industriellen Assets und Software-Komponenten, die bspw. über das Internet kommunizieren können, führen zu einem cyber-physischen System (CPS) [Bun20]. Umfassendere komplexe Vernetzungen mehrerer CPS führen bspw. zu intelligenten Fabriken, die voll- oder teilautomatisiert produzieren können und menschliches Eingreifen auf ein Minimum reduzieren. Die Ziele umfassen bspw. eine flexiblere Produktion bis hin zur Losgröße 1 oder eine zustandsorientierte Instandhaltung der Maschinen, um diese bis zur Verschleißgrenze zu benutzen. Im Rahmen von I4.0 kommen Cloud-Technologien zum Einsatz, die für die Analyse und Verarbeitung der großen Datenmengen verwendet werden. Allerdings müssen diese aufgrund strikter Anforderungen, z. B. bzgl. Latenz, nahe an den Datenquellen, d. h., den industriellen Assets, eingesetzt werden, was im Allgemeinen als *Edge-Computing* bekannt ist [SCZ+16]. Die Kombination von Cloud-Technologien und Edge-Computing ergibt Edge-Clouds, welche die Vorteile des Cloud-Computing für industrielle Zwecke erlaubt [CHML14].

Industrie 4.0 ermöglicht durch die Einführung von CPS daher ein Monitoring der industriellen Assets, um Fehler frühzeitig zu erkennen und zu beheben oder bereits prädiktiv dagegen vorzugehen, was unter dem Begriff *Condition Monitoring* (dt.: Zustandsüberwachung) zusammengefasst wird [Kol17]. Aufgrund der hohen Heterogenität industrieller Assets und weiteren Anforderungen wie Latenz sind Cloud-Monitoring-Systeme in der Regel nicht praktikabel. Stattdessen werden maßgeschneiderte Lösungen in diesem Bereich eingesetzt, die sich analog zu ITIM- oder APM-Systemen je nach Einsatzbereich in ihrer Funktionalität unterscheiden.

Zu diesem Zweck ist an der Universität Stuttgart das Forschungsprojekt SitOPT[1] entstanden, dessen Themenbereich auch die Überwachung von I4.0-Umgebungen sein kann [WSBL15]. Dazu wird eine Situationserkennung

[1] https://www.ipvs.uni-stuttgart.de/de/abteilungen/as/forschung/projekte/SitOpt/

verwendet, um aus Sensordaten höherwertige Informationen abzuleiten, die als *Situationen* bezeichnet werden. Diese Situationen können anschließend in situationsadaptiven Workflows verwendet werden, die z. B. zur Steuerung der Produktion verwendet werden können. Je nach Situation können diese Workflows ihre Ausführung entsprechend anpassen. Die benötigten Sensordaten werden zunächst an eine zentrale Komponente weitergeleitet, was analog zum Cloud-Monitoring bei großen Datenmengen in einer hohen Belastung für das Netzwerk resultiert.

Insgesamt betrachtet müssen sowohl Cloud-Umgebungen als auch industrielle Assets überwacht werden, was im Rahmen dieser Arbeit als *Monitoring industrieller Cloud-Umgebungen* bezeichnet wird. Als Resultat dieses Monitoring werden daher eine Vielzahl verschiedener Monitoring-Systeme eingesetzt. Zusätzlich existieren für gewöhnlich weitere Systeme, die wertvolle und relevante Daten für das Verwalten der Cloud-Ressourcen und industriellen Assets haben. In Geräte-Management-Systemen sind Details zu den industriellen Assets hinterlegt. Cloud-Management-Plattformen enthalten statische Informationen zu den VMs wie deren Erstellungsdatum oder Event-Informationen wie z. B. die Anweisung zum Abschalten einer VM. Service-Level-Agreement (SLA)-Management-Systeme enthalten die vertraglich vereinbarten Regelungen für gebuchte Cloud-Ressourcen. Der Verbraucher kann diese Informationen in Kombination mit dem Monitoring nutzen, um bspw. zu überprüfen, ob die Cloud-Ressourcen den SLAs entsprechen, um bei einer entsprechenden Verletzung der SLAs eine Kostenminderung zu erhalten. Der Cloud-Anbieter hingegen nutzt dies, um sicherzustellen, dass es nicht zu einer solchen Verletzung kommt und erhebt frühzeitig Maßnahmen dagegen. Die Summe all dieser Systeme zum Monitoring und Management führt zu einer Vielzahl von Datensilos, die von verschiedenen Abteilungen verwaltet werden, unter denen oftmals wenig bis gar keine Kommunikation stattfindet [NKFW19]. Eine umfassende Sicht, die einen Gesamtüberblick über die industrielle Cloud-Umgebung liefern würde, wird somit stark erschwert.

Zusammenfassend werden im Rahmen der vorliegenden Dissertation Konzept zur Verbesserung des Monitoring in industriellen Cloud-Umgebungen

vorgestellt. Hierfür werden zunächst individuelle Lösungen im Bereich des Cloud-Monitoring und der Situationserkennung und anschließend ein domänenübergreifendes Kontextmodell für die Verwaltung der industriellen Cloud-Umgebungen präsentiert. Die Beiträge dieser Dissertation wurden auf internationalen Konferenzen veröffentlicht. Eine Übersicht der Publikationen befindet sich am Ende des Dokuments (S. 175-176).

1.2 Forschungsfragen und Ziele der Dissertation

Aus dem vorigen Abschnitt ergeben sich die folgenden Forschungsfragen:

- Wie kann die Gefahr eines Lock-ins bei Monitoring-Systemen reduziert werden?
- Wie können existierende Monitoring-Systeme verbessert werden, um den steigenden Anforderungen nach niedrigerem Netzwerkverkehrvolumen, geringer Adminstrationskomplexität, und hoher Genauigkeit der Monitoring-Daten gerecht zu werden?
- Wie können neue Konzepte wie Edge-Clouds effizient für die Situationserkennung verwendet werden?
- Wie kann eine umfassende Sicht auf eine industrielle Cloud-Umgebung ermöglicht werden?

Diese Fragestellungen werden durch die vorliegende Dissertation behandelt und resultieren in den vier im Folgenden aufgeführten Zielen.

1.2.1 Ziel 1: Gefahr von Lock-ins bei agentenbasierten Monitoring-Systemen verringern

Der Vorteil von Agenten ist, dass sie detaillierte Informationen über die überwachte Ressource sammeln können. Allerdings ist dies mit der Installation, Wartung und Konfiguration einer Vielzahl von Agenten verbunden. In der Regel besitzt jedes Monitoring-System eigene Agenten, die für die Datenerfassung zuständig sind. Der Austausch eines Monitoring-Systems

führt daher zwangsläufig zum Austausch der zugehörigen Agenten, die in der industriellen Cloud-Umgebung integriert sind. Die jeweiligen Agenten unterscheiden sich bspw. in der Programmiersprache oder den unterstützten Funktionalitäten. Ein Austausch des Monitoring-Systems führt daher nicht nur zur Neuinstallation und -konfiguration der neuen Agenten, sondern auch dazu, dass Systemadministratoren im schlimmsten Fall neue Programmiersprachen erlernen muss, um Agenten korrekt zu konfigurieren und diese möglicherweise nicht über alle benötigten Funktionalitäten, wie z. B. Aggregation, verfügen.

Das erste Ziel ist daher, eine Abstraktionsstufe bei der Modellierung von Agenten einzuführen, um generische Agenten modellieren zu können, ohne technische Details kennen zu müssen. Abhängig von dem gewünschten Monitoring-System werden die generischen Agenten automatisch in die entsprechenden lösungsspezifischen Agenten transformiert und mit standardkonformen Technologien auf der industriellen Cloud-Umgebung installiert.

1.2.2 Ziel 2: Akkurate Analyse in agentenbasierten Monitoring-Systemen bei niedrigem Netzwerkverkehrvolumen

Eine Hauptaufgabe von Monitoring-Systemen ist das Alerting, das bei erkannten Problemen in der industriellen Cloud-Umgebung die zuständigen Systemadministratoren alarmiert [Lig12c]. Hierfür werden die erfassten Monitoring-Daten an zentraler Stelle mittels vordefinierter Alerting-Regeln analysiert. Eine Aggregation der Monitoring-Daten zur Reduktion des hohen Netzwerkverkehrvolumens auf den Agenten führt allerdings zu einem Informationsverlust und reduziert somit die Qualität des Alerting [TPD15].

Das zweite Ziel beschreibt eine Auswertung von Alerting-Regeln auf den überwachten VMs, um das Netzwerkverkehrvolumen gering zu halten. Alle benötigten Prozesse, wie z. B. Anpassungen der Agenten, sollen automatisiert erfolgen. Die Verwaltung der Alerting-Regeln soll weiterhin zentralisiert bleiben, um keine erhöhte Administrationskomplexität zu erzeugen. Es wird

darauf geachtet, dass der Lösungsansatz auf einer Vielzahl von Monitoring-Systemen anwendbar ist, um nicht nur einzelne Monitoring-Systeme zu verbessern, was die Gefahr von Lock-ins wiederum erhöhen könnte.

1.2.3 Ziel 3: Analyse verschiedener Ausführungsumgebungen und verteilte Ausführung der Situationserkennung

Durch I4.0 sind neue Möglichkeiten für das Monitoring industrieller Assets entstanden und durch die Einführung von Cloud-Technologien verschiedene Ausführungsumgebungen, die für eine Situationserkennung verwendet werden können. Die Erkennung von Situationen basiert auf Situationstemplates [HHL+10], die dahingehend erweitert wurden, um Situationen modellieren und erkennen zu können [HWS+16].

Das Ziel ist es, zunächst die verschiedenen Ausführungsumgebungen für einen Einsatz der Situationserkennung unter Beachtung industrieller Anforderungen zu analysieren. Im nächsten Schritt wird eine verteilte Situationserkennung präsentiert, für die sowohl auf der Modellierungsebene von Situationstemplates als auch auf der Ausführungsebene entsprechende Änderungen vorgenommen werden.

1.2.4 Ziel 4: Umfassende Sichtweise auf eine industrielle Cloud-Umgebung ermöglichen

Die Informationen zu einer industriellen Cloud-Umgebung werden von einer Vielzahl verschiedener Systeme überwacht und verwaltet. Monitoring-Daten werden z. B. von ITIM-, APM-, und Netzwerk-Monitoring-Systemen erfasst und Daten zur Verwaltung der Cloud-Ressourcen sind bspw. in Cloud-Management-Systemen hinterlegt.

Das vierte Ziel dieser Arbeit ist es, ein Kontextmodell für das Monitoring industrieller Cloud-Umgebungen zu erstellen, um alle relevanten Informationen darzustellen, die im Rahmen des Monitoring benötigt werden. Dieses

Kontextmodell soll unter Anderem als Grundlage für die Modellierung von Topologien industrieller Cloud-Umgebungen dienen und ein effizienteres Monitoring ermöglichen.

1.3 Herausforderungen und Stand verwandter Arbeiten

In diesem Abschnitt werden die grundlegenden Herausforderungen vorgestellt, welche dieser Arbeit zugrunde liegen. Diese lassen sich in (i) Lock-in-Effekte bzgl. Technologien oder Anbieter, (ii) Hohes Netzwerkverkehrvolumen bei Cloud-Monitoring, (iii) Hohe Latenz und Netzwerkverkehrvolumen bei Situationserkennung, und (iv) Isolierte Datenhaltung unterscheiden. Zusätzlich wird betrachtet, wie verwandte Arbeiten mit diesen Herausforderungen umgehen.

1.3.1 Lock-in-Effekte an Technologien oder Anbieter

Lock-in-Effekte an bestimmte Technologien oder Anbieter sind nichts Ungewöhnliches und existieren in allen Bereichen. Im Cloud-Computing wird der Vendor-Lock-in (dt.: Herstellerbindung) als eine zeit- und kostenintensive Migration von Cloud-Ressourcen zu anderen Cloud-Anbietern [OST14] charakterisiert. Analog dazu wird der Vendor-Lock-in im Cloud-Monitoring als ein zeit- und kostenintensiver Austausch eines Monitoring-Systems und dazugehöriger Agenten charakterisiert.

Je nach Domäne existieren verschiedene Ansätze, um dem Lock-in-Effekt entgegenzuwirken. Im Cloud-Computing wird bspw. eine Standardisierung der Schnittstellen bei allen Anbietern als mögliche Lösung vorgeschlagen [OST14]. Ein gebräuchliches Mittel, das in mehreren Domänen Ansatz findet, ist die Transformation von abstrakten Modellen auf konkrete ausführbare Implementierungen. Hierbei wird zunächst bei der Modellierung von technischen Details abstrahiert und Domänennutzer sind in der Lage, Modelle auf einem hohen Abstraktionsniveau zu erstellen, ohne technische Kenntnisse über deren Realisierung zu benötigen. Die Verwendung eines einzigen abstrakten Modells führt zu Generizität und verringert die Gefahr eines

Lock-ins, da es nicht von bestimmten Technologien oder Anbietern abhängig ist. Dieser Ansatz wird in mehreren verwandten Arbeiten verfolgt [EEKS11; FBB+14; Hir18; RSM14], wurde allerdings bisher nicht für die Domäne des Cloud-Monitoring angewandt.

1.3.2 Wachsendes Datenvolumen von Monitoring-Daten

Durch die Verwendung von Cloud-Computing ist es selbst klein- und mittelständischen Unternehmen möglich geworden, große und komplexe Cloud-Umgebungen zu erstellen, die detailliert und über alle Schichten der Service-Modelle (Infrastruktur, Plattform, Anwendungen) hinweg überwacht werden müssen. Die Monitoring-Systeme, die hierfür eingesetzt werden, besitzen in der Regel eine zentralisierte Architektur [ARM+15; HW18]. Das bedeutet, dass die Agenten alle erfassten Monitoring-Daten zunächst an eine zentrale Datenbank des Monitoring-Systems weiterleiten, die üblicherweise auf einem separaten Monitoring-Server liegt. Dies führt zu großen Datenmengen, die das Netzwerk stark belasten und zu Engpässen auf dem Monitoring-Server führen [Har17].

Die in der Praxis übliche Methode ist die Aggregation von Monitoring-Daten auf dem Agenten, um das Datenvolumen zu reduzieren [ABdP13]. Dies führt jedoch unweigerlich zu einem Informationsverlust und damit zu einer sinkenden Genauigkeit des Monitoring [TPD15]. Eine Alternative sind dezentralisierte Monitoring-Architekturen, die jedoch aufgrund einer hohen Administrationskomplexität in der Regel nicht praktikabel sind [WB14]. In der Literatur werden daher hybride Architekturen vorgeschlagen, die eine Auswertung der Monitoring-Daten direkt auf oder nahe an den überwachten Ressourcen erlauben und weiterhin eine zentralisierte Architektur vorsehen [HW18; KEW+10; SWWM10; WST+11]. Diese Ansätze haben gemein, dass die Analysen händisch definiert werden müssen, was ebenso die Administrationskomplexität erhöht.

Gleichermaßen ist die Erstellung neuer Monitoring-Systeme wie in [TPD15] oder [KEW+10] nicht zielführend, da auf diese Weise erneut ein Lock-in entstehen würde.

1.3.3 Hohe Latenz und Netzwerkverkehrvolumen bei Situationserkennung

Der Grundgedanke bei der Modellierung und Erkennung von Situationen liegt bei der Abstrahierung von den vielen, verschiedenen Kontextdaten, die zur Erkennung der Situation beitragen. Auf diese Weise können situationsadaptive Workflows erstellt, die z. B. zur Steuerung der Produktion verwendet werden können, um je nach Situation ihre Ausführung entsprechend anzupassen [WSBL15]. Würden diese Workflows stattdessen direkt die zugrundeliegenden Kontextdaten erfassen, müssten diese jeweils einzeln im Workflow modelliert werden und somit die Modellierungskomplexität stark erhöhen [WSBL15].

Um den wachsenden Anforderungen bzgl. Latenz und den wie im Cloud-Monitoring stets wachsenden Datenmengen [LCW08] entgegenzutreten, sind neue Ausführungsumgebungen wie z. B. Edge-Clouds für die Situationserkennung zu beachten, um die Auswertung nahe an den Datenquellen zu ermöglichen. Dadurch werden die ansonsten unvermeidlichen, hohen Netzwerklatenzen umgangen und ermöglichen eine zeitgerechte Bereitstellung der erkannten Situationen.

Jedoch ist eine ausschließlich auf der Edge-Cloud ausgeführte Situationserkennung je nach Anwendungsfall möglicherweise nicht möglich, z. B. wenn dort nicht alle benötigten Datenquellen erreichbar sind. Daher ist es notwendig, verschiedene Ausführungsumgebungen und deren Auswirkungen auf die Situationserkennung zu analysieren, und letztlich eine Verteilung der Situationserkennung auf die unterschiedlichen Umgebungen zu ermöglichen.

Es existieren verschiedene Ansätze zur verteilten Situationserkennung, die auf Complex-Event-Processing (CEP) [SKPR10; SMP09], Ontologien [FZYZ08], oder Machine-Learning [ASRH13] beruhen. Ontologien und Machine-Learning-Ansätze sind aufgrund der strikten Latenzanforderungen nicht geeignet [ASRH13; FZYZ08]. Bei verteilten CEP-basierten Ansätzen werden hingegen die unterschiedlichen Aspekte der verschiedenen Ausführungsumgebungen wie Sicherheit oder Kosten nicht beachtet und des Weiteren entsteht hierbei abermals ein technologischer Lock-in, der im Rahmen der vorliegenden Arbeit nicht zielführend ist.

1.3.4 Isolierte Betrachtung der Ressourcen

Bereits die Vielzahl verwendeter Systeme zum Monitoring der Cloud-Ressourcen führt zu einer großen Anzahl von Datensilos, die von verschiedenen Abteilungen verwaltet werden, unter denen oftmals wenig bis gar keine Kommunikation stattfindet [NKFW19]. Hinzu kommen sowohl Systeme für das Monitoring industrieller Assets als auch Management-Systeme zur Verwaltung der industriellen Cloud-Umgebung. Auf diese Weise wird eine umfassende Sichtweise auf die überwachte industrielle Cloud-Umgebung, in welcher die verschiedenen Abhängigkeiten aller Ressourcen zueinander ersichtlich sind, erschwert.

Im Bereich des Cloud-Monitoring wurde von Gartner [BMC20] daher der Begriff AIOps (Artificial Intelligence in IT Operations; dt.: Künstliche Intelligenz in IT-Operations[1]) eingeführt. AIOps beschreibt unter anderem die Konsolidierung aller relevanten Monitoring-Daten und sieht als Voraussetzung dafür ein Kontextmodell vor.

Allerdings stellen weder Gartner noch Monitoring-Systeme, die ihrer Aussage nach AIOps unterstützen, ein solches Modell vor. Im Rahmen dieser Dissertation ist zudem die ausschließliche Betrachtung der Domäne Cloud-Monitoring nicht zielführend, da auch das Monitoring industrieller Assets in ein solches Kontextmodell miteinbezogen werden muss.

1.4 Beiträge der Dissertation

Im Folgenden werden die Beiträge dieser Dissertation beschrieben, welche die in Abschnitt 1.2 definierten Forschungsfragen beantworten und Ziele erfüllen. Von jedem Beitrag wird ein Ziel erfüllt. Gemeinsam bilden diese Beiträge eine Lösung für das Monitoring industrieller Cloud-Umgebungen. Aufgrund unterschiedlicher Anforderungen und Voraussetzungen werden zunächst individuelle Lösungen für agentenbasierte Monitoring-Systeme

[1] IT-Operations ist der übergreifende Begriff für das Management einer IT-Umgebung. Monitoring stellt neben weiteren Aufgaben wie Vertragsabwicklung mit Software-Anbietern einen Teil der IT-Operations dar.

sowie für die Situationserkennung vorgestellt. Abschließend werden alle relevanten Aspekte des Monitoring industrieller Cloud-Umgebungen in einem Kontextmodell abgebildet, um eine umfassende Sicht auf eine überwachte industrielle Cloud-Umgebung zu ermöglichen. Die jeweiligen Beiträge wurden auf internationalen Konferenzen und Fachmagazinen veröffentlicht. Eine Übersicht der Publikationen befindet sich am Ende des Dokuments (S. 175-176). Die im Rahmen dieser Arbeit entstandene Software ist auf GitHub[1] verfügbar.

1.4.1 Generische Agententemplates

Mit dem ersten Beitrag soll erreicht werden, die Gefahr eines Vendor-Lock-ins bei agentenbasierten Monitoring-Systemen zu verringern. Dies soll dadurch ermöglicht werden, dass der Austausch eines Monitoring-Systems erleichtert werden soll. Wie zuvor beschrieben, stellen Monitoring-Systeme üblicherweise ihre eigenen Agenten bereit, die sich in der Programmiersprache und Funktionalität voneinander unterscheiden. Eine Neuinstallation und -konfiguration neuer Agenten ist daher mit einem erheblichen Zeit- und Kostenaufwand verbunden.

Daher werden in diesem Beitrag *generische Agententemplates* vorgestellt, die eine Abstraktionsstufe bei der Modellierung von Agenten darstellen. Dadurch wird die Modellierung von abstrakten Agenten ermöglicht, ohne technische Details der lösungsspezifischen Agenten erlernen zu müssen. Stattdessen werden das grundlegende Verhalten und die Funktionen eines Agenten modelliert. Anschließend können diese abstrakten Agenten in beliebige lösungsspezifische Agenten der gewünschten Monitoring-Systeme transformiert werden. Um die Skalierbarkeit für Installation und Wartung von Agenten zu erhöhen, wird im Rahmen eines Lebenszyklus von Agenten die automatische Bereitstellung und dynamische Anpassung von Agenten vorgestellt. Die Inhalte dieses Beitrags wurden auf der *International Conference on Business Information Systems (BIS)* veröffentlicht [MHSM20a].

[1] https://github.com/

1.4.2 Verteilte Auswertung von Alerting-Regeln

Die bisherigen Ansätze aus Praxis und Literatur führen im Umgang mit steigenden Datenmengen durch das Monitoring zu Kompromissen, welche zur Reduktion des Netzwerkverkehrvolumens die Genauigkeit des Monitoring verringern oder die Komplexität der Administration erhöhen.

In diesem Beitrag wird daher *DEAR* (Distributed Evaluation of Alerting Rules; dt.: Verteilte Auswertung von Alerting-Rules) vorgestellt []. Hierbei wird die Auswertung der Alerting-Regeln auf die Agenten verteilt, sodass auf die Übermittlung feingranularer Monitoring-Daten an den Monitoring-Server verzichtet werden kann. Somit sinkt das Netzwerkverkehrvolumen trotz einer hohen Genauigkeit der Monitoring-Daten in Bezug auf das Alerting. Die Definition und Verwaltung der Alerting-Regeln verbleibt an zentraler Stelle, damit die Administrationskomplexität nicht erhöht wird. Die Verteilung sowie benötigte Anpassungen an Agenten werden automatisiert ausgeführt. Die Inhalte dieses Beitrags wurden auf der *IEEE International Conference On Cloud Computing (CLOUD)* veröffentlicht [MHSM20b].

1.4.3 Verteilte Situationserkennung

Neue Ausführungsumgebungen wie Edge-Clouds ermöglichen es, den strikten Latenzanforderungen in I4.0 gerecht zu werden und ein zeitgerechtes Monitoring industrieller Assets zu ermöglichen. Weitere Anforderungen (z. B. Sicherheit) als auch Einschränkungen (Datenquellen sind verteilt) führen zu unterschiedlichen Ausführungsumgebungen, die für eine Situationserkennung verwendet werden können.

In diesem Beitrag werden zunächst die Anforderungen an die Situationserkennung erfasst und anhand dieser Anforderungen verschiedene Ausführungsumgebungen analysiert. Um eine verteilte Situationserkennung zu ermöglichen, werden bestehende Ansätze zur Situationserkennung erweitert, was sowohl die Modellierung von Situationen als auch die Ausführung der Situationserkennung betrifft. Die Inhalte dieses Beitrags wurden auf der

International Conference on Smart Cities, Systems, Devices and Technologies (SMART) [MHWM18] und im *International Journal On Advances in Intelligent Systems* [MHWM19] veröffentlicht.

1.4.4 Kontextmodell für das Monitoring industrieller Cloud-Umgebungen

Das Monitoring und Verwalten einer industriellen Cloud-Umgebung erfolgt über eine Vielzahl von Monitoring- und Management-Systemen, deren Daten logisch nicht miteinander verknüpft vorliegen. Dadurch wird eine umfassende Sichtweise auf die industrielle Cloud-Umgebung erschwert.

Daher wird im vierten Beitrag ein Kontextmodell für das Monitoring industrieller Cloud-Umgebungen vorgestellt, dass die überwachten Ressourcen, die Monitoring- und Management-Daten zur Beschreibung dieser Ressourcen, und die jeweiligen Systeme zur Verwaltung bzw. Erfassung dieser Daten enthält. Des Weiteren werden auf den Daten aufbauende Analysen und daraus resultierende Aktionsmöglichkeiten, wie z. B. Alerting, im Kontextmodell modelliert. Die Inhalte dieses Beitrags wurden in *Computer Science - Research and Development* [MHWM17] und auf dem *Workshop on Context Modeling and Activity Recognition (CoMoRea)* [MS20] veröffentlicht.

1.5 Aufbau der Arbeit

Die restliche Dissertation ist wie folgt strukturiert: In Kapitel 2 werden die Grundlagen dieser Arbeit vorgestellt. Diese umfassen Grundlagen zu Cloud-Computing, Industrie 4.0 und Monitoring. Kapitel 3 liefert anhand einer beispielhaften industriellen Cloud-Umgebung einen Überblick der Beiträge dieser Dissertation. Diese werden in den darauffolgenden Kapiteln Kapitel 4, Kapitel 5, Kapitel 6, und Kapitel 7 detailliert vorgestellt. Schließlich fasst Kapitel 8 die Inhalte dieser Arbeit zusammen und liefert einen Ausblick für zukünftige Arbeiten zu den jeweiligen Beiträgen dieser Dissertation.

Kapitel 2

Grundlagen

2.1 Cloud-Computing

Cloud-Computing erlaubt durch die Auslagerung von IT-Infrastruktur an Drittanbieter eine bedarfsgerechte Bereitstellung von Ressourcen, die optimal auf die Bedürfnisse des Verbrauchers zugeschnitten werden können. Dies wird durch eine Virtualisierung physischer Rechenressourcen ermöglicht. Im Folgenden werden die Charakteristika von Cloud-Computing basierend auf der Definition des National Institute of Standards and Technology (NIST) [MG11] vorgestellt:

- Bedarfsabhängige Selbstbedienung: Der Verbraucher kann eigenständig, d. h., ohne menschliche Interaktionen mit dem Cloud-Anbieter, nach Bedarf weitere Ressourcen wie VMs oder Netzwerkspeicher buchen. Daraus folgt, dass dieser Prozess automatisiert erfolgen kann und dem Verbraucher zu jeder Zeit die auf seinen Bedarf zugeschnittenen Ressourcen verfügbar gemacht werden.
- Breiter Netzwerkzugriff: Der Zugriff auf die Ressourcen wird über ein Netzwerk ermöglicht, welches auf standardkonformen Mechanismen

basiert. Somit soll ein Zugriff durch heterogene Systeme, z. B. leichtgewichtige Geräte wie Mobiltelefone sowie schwergewichtige Systeme wie PCs, ermöglicht werden.

- Ressourcen-Pooling: Der Cloud-Anbieter bündelt seine Ressourcen, um basierend auf einem Multi-Tenant-Modell mehrere Verbraucher damit zu versorgen. Jedem Verbraucher können hierbei je nach Bedarf mehr oder weniger physische sowie virtuelle Ressourcen dynamisch hinzugefügt oder wieder entfernt werden. Der Verbraucher kann grob entscheiden, wo sich die tatsächlichen physischen Ressourcen befinden sollen, z. B. in welchem Land, ohne jedoch exakte Positionsangaben zu kennen.
- Dynamische Elastizität: Die Ressourcen können manuell oder automatisch dynamisch hinzugefügt oder wieder entfernt werden. Auf diese Weise können die Ressourcen bei Bedarfsspitzen schnell hinzugefügt werden und bei zu geringer Auslastung wieder freigegeben werden, um Kosten zu sparen. Für den Verbraucher scheint es, als gäbe es unendlich viele Ressourcen, welche zu jeder Zeit in beliebiger Menge gebucht werden können.
- Automatisierte Messungen: Cloud-Systeme überwachen die Ressourcennutzung, um diese automatisch zu optimieren. Hierfür wird ein Monitoring eingesetzt, das sowohl den Anbieter als auch den Verbraucher über die verwendeten Cloud-Ressourcen benachrichtigt.

Des Weiteren definiert NIST die drei Service-Modelle *Software-as-a Service (SaaS)*, *Platform-as-a-Service (PaaS)*, und *Infrastructure-as-a-Service (IaaS)*.

- IaaS: Das IaaS-Modell stellt dem Verbraucher Infrastruktur-Ressourcen wie Speicher, Netzwerk und Rechenressourcen bereit, und der Verbraucher entscheidet frei über die Wahl von Betriebssystem und installierten Plattformen und Anwendungen. Der Verbraucher hat keine Kontrolle über die zugrundeliegende Cloud-Infrastruktur, dafür aber Kontrolle über Betriebssysteme, Speicher und Anwendungen, und eine beschränkte Kontrolle über Netzwerkkomponenten wie Firewalls.

- PaaS: Beim PaaS-Modell wird dem Verbraucher die Möglichkeit zur Erstellung von Anwendungen durch diverse Programmiersprachen, Bibliotheken, Werkzeuge und Services geboten, welche anschließend auf der Cloud-Infrastruktur bereitgestellt werden können. Der Verbraucher hat keine Kontrolle über die zugrundeliegende Infrastruktur, die im IaaS-Modell vorhanden ist. Dafür kann der Verbraucher frei über die bereitgestellten Anwendungen entscheiden und hat möglicherweise Zugriff auf Konfigurationseigenschaften der bereitgestellten Plattform.
- SaaS: In SaaS erhält der Verbraucher Zugriff auf Anwendungen des Anbieters, die auf der Cloud-Infrastruktur bereitgestellt werden. Der Zugriff auf die Anwendungen wird über verschiedene Systeme und Schnittstellen ermöglicht, z. B. über einen Web-Browser. Der Verbraucher kontrolliert weder die zugrundeliegende Cloud-Infrastruktur noch das verwendete Betriebssystem. Es ist unter Umständen möglich, dass der Verbraucher bestimmte Konfigurationseigenschaften der bereitgestellten Anwendung kontrollieren kann.

Im Laufe der Zeit sind weitere Service-Modelle erschienen, darunter bspw. *Function-as-a-Service (FaaS)* [Rob18] und *Database-as-a-Service (DBaaS)* [LK18], die dem Verbraucher das Ausführen von Funktionen (auch *Serverless Computing) genannt* bzw. den Zugriff auf einen Datenspeicher in der Cloud ermöglichen. Diese und weitere Unterteilungen sind im Rahmen dieser Arbeit nicht relevant und werden nicht näher erörtert.

Zuletzt unterscheidet NIST vier verschiedene Bereitstellungsmodelle der Cloud, die sich in der Art der Zugriffsrechte voneinander unterscheiden:

- Public-Cloud: Eine Public-Cloud (dt.: öffentliche Cloud) ist der breiten Öffentlichkeit zugänglich und kann von einem Unternehmen oder akademischen sowie staatlichen Organisationen verwaltet und betrieben werden. Der Betrieb der Cloud findet am Standort des Anbieters statt.
- Private-Cloud: Die Private-Cloud (dt.: private Cloud) wird exklusiv von einer einzigen Organisation verwendet, die aus mehrere Verbrauchern bestehen kann. Die Cloud kann von der Organisation selbst, einem

Drittanbieter, oder einer Kombination von beidem verwaltet und betrieben werden. Der Betrieb der Cloud kann am Standort des Betreibers oder außerhalb stattfinden.

- Community-Cloud: Die Community-Cloud (dt.: Gemeinschafts-Cloud) besitzt die gleichen Eigenschaften wie die Private-Cloud. Der einzige Unterschied liegt darin, dass statt einer einzelnen Organisation mehrere Organisationen, die gemeinsame Ziele haben, Zugriff auf die Cloud haben. Demnach kann auch die Community-Cloud von einer oder mehreren der Organisation, einem Drittanbieter, oder einer Kombination von beidem verwaltet und betrieben werden. Der Betrieb der Cloud kann am Standort des Betreibers oder außerhalb stattfinden.
- Hybrid-Cloud: Eine Hybrid-Cloud (dt.: hybride Cloud) ist eine Zusammensetzung aus zwei oder mehreren der bereits genannten Bereitstellungsmodelle. Die jeweiligen Clouds verbleiben unabhängig voneinander, werden aber über standardisierte oder proprietäre Technologien miteinander verbunden, um eine Portabilität von Daten und Anwendungen zwischen den einzelnen Clouds zu ermöglichen.

Innerhalb der letzten Jahre sind zusätzlich zwei weitere Ausprägungen hinzugekommen, die für diese Arbeit von Relevanz sind: *Multi-Clouds* und *Edge-Clouds*.

- Multi-Cloud: Eine Multi-Cloud ähnelt der Hybrid-Cloud in der Hinsicht, dass mehrere Clouds parallel genutzt werden. Im Fokus von Multi-Clouds liegt jedoch die Absicht, mehrere unterschiedliche Cloud-Anbieter zu verwenden, um einen Vendor Lock-in und beim Ausfall einzelner Cloud-Anbieter einen Datenverlust zu vermeiden. Die Verwendung von Multi-Clouds hat sich zu einer weit verbreiteten Strategie entwickelt, um das Beste aus den verschiedenen Bereitstellungsmodellen sowie den verschiedenen Cloud-Anbietern herauszuholen [Pet13].
- Edge-Cloud: Edge-Clouds sind aus den Herausforderungen der Industrie 4.0 (s. Abschnitt 2.2), wie z. B. geringe Latenzzeiten und Ausfallsicherheit, entstanden [CHML14]. Diese bauen auf dem Konzept des

Edge Computing auf, das eine Verarbeitung von Daten nahe an den Datenquellen vorsieht [SCZ+16]. In Edge-Clouds wird Edge-Computing mit Cloud-Technologien angereichert.

2.2 Industrie 4.0

Mit Industrie 4.0 (I4.0) wird eine vom Bundesministerium für Bildung und Forschung[1] ins Leben gerufene Hightech-Strategie zur Digitalisierung in der verarbeitenden Industrie bezeichnet [Bun20]. I4.0 umfasst eine Vielzahl von Projekten, die alle zum Ziel haben, die industrielle Produktion mit aktuellen Informations- und Kommunikationstechnologien auszustatten, um z. B. eine flexiblere Produktion bis hin zur Losgröße 1 zu ermöglichen. Hierfür wird unter anderem auch ein datengetriebener Ansatz verfolgt. Als Grundlage dient daher eine umfassende sensorische Erfassung der Umgebungsdaten, die z. B. mithilfe analytischer Prozesse für Anwendungsfälle wie Predictive-Maintenance (dt.: prädiktive Wartung) eingesetzt werden können. Jedes Asset muss daher mit Sensoren ausgestattet werden. Ein *Asset* wird als *„alles, was verbunden werden muss, um eine I4.0-Lösung zu erzeugen“* bezeichnet [Pla19]. Zur deutlicheren Abgrenzung zu den anderen Ressourcen, die in dieser Arbeit Ziel des Monitoring sind, wie z. B. VMs, wird im Rahmen dieser Arbeit der Begriff *industrielles Asset* verwendet, um den Bezug zu I4.0 zu verdeutlichen.

Die Vernetzung von industriellen Assets und Software-Komponenten, die über bspw. das Internet kommunizieren können, führen zu einem cyber-physischen System (CPS) [Bun20]. Umfassendere komplexe Vernetzungen dieser CPS führen zu cyber-physischen Produktionssystemen (CPPS), mithilfe derer intelligente Fabriken erstellt werden können, die voll- oder teilautomatisiert produzieren können und menschliches Eingreifen auf ein Minimum reduzieren.

Im Rahmen des Projekts IC4F[2] [Stu21] wurde eine industrielle Referenzarchitektur entwickelt, um klein- und mittelständische Unternehmen dabei

[1] https://www.bmbf.de/
[2] https://ic4f.de/

zu unterstützen, eine Infrastruktur bestehend aus industriellen Assets und digitalen Komponenten nach dem Stand der Technik zu modellieren. Hierbei war der Einsatz von Cloud-Technologien (vgl. Edge-Cloud) unumgänglich. Um diese IT-Umgebungen zu beschreiben, in denen industrielle Assets mit Clouds interagieren, wird im Rahmen dieser Arbeit der Begriff *industrielle Cloud-Umgebung* eingeführt.

2.3 Monitoring

Das Verb *monitor* aus dem Englischen wird definiert als *„sorgfältig beobachten und prüfen einer Situation für eine bestimmte Zeit, um etwas darüber zu erfahren“* [Cam21]. Der Begriff des Monitoring wird daher in einer Vielzahl von Domänen wie Biologie, Psychologie und Bauwesen verwendet und besitzt dort jeweils unterschiedliche Ausprägungen. Das Monitoring industrieller IT-Umgebungen erfordert eine technische Ausprägung dieser Definition. Im Rahmen dieser Arbeit wird daher die Definition nach Ligus [Lig12a] verwendet. Der Begriff wird wie folgt definiert: *„Monitoring ist der Prozess der Überwachung der Existenz und des Ausmaßes von Zustandsänderungen und des Datenflusses in einem System. Die Überwachung zielt darauf ab, Fehler zu identifizieren und bei ihrer späteren Beseitigung zu helfen. Die bei der Überwachung von Informationssystemen verwendeten Techniken überschneiden sich mit den Bereichen Echtzeitverarbeitung, Statistik und Datenanalyse. Eine Reihe von Softwarekomponenten, die für die Datenerfassung, ihre Verarbeitung und Präsentation verwendet werden, wird als Monitoring-System bezeichnet“* [Lig12a].

Für das Monitoring industrieller Assets muss zusätzlich die in der verarbeitenden Industrie verwendete Definition des *Condition-Monitoring* (dt.: Zustandsüberwachung) beachtet werden. Hierbei liegt der Fokus auf dem Monitoring von Maschinen und ihrer Prozesse und umfasst dabei Aufgaben wie Fehler(früh)erkennung, Fehlerdiagnose, Trendanalyse und Prognose [Kol17]. Dies stellt daher einen speziellen Anwendungsfall des Monitoring dar, der bestimmte Ziele verfolgt, wie z. B. eine zustandsorientierte Instand-

haltung der Maschinen. Die Definition nach Ligus ist generischer und umfasst die Definition des Condition-Monitoring und wird daher im weiteren Verlauf dieser Arbeit als Begriffsdefinition für *Monitoring* verwendet.

Im Folgenden werden unterschiedliche Ausprägungen von Monitoring-Systemen beschrieben. Diese unterscheiden sich vor allem in der Architektur, der Art der Erfassung von Monitoring-Daten, und dem speziellen Anwendungsbereich der Monitoring-Systeme.

2.3.1 Architektur

Architekturen von Monitoring-Systemen können allgemein in zentralisiert und dezentralisiert unterschieden werden [WB14]. In zentralisierten Monitoring-Systemen existiert eine zentrale Komponente – der Monitoring-Server – an den alle erfassten Monitoring-Daten gesendet und in einer zentralen Datenbank gespeichert werden. In der Regel wird auf dem Monitoring-Server eine zusätzliche Komponente zur Visualisierung verwendet. Die für das Monitoring zuständigen Systemadministratoren können diese Komponente zur Visualisierung der Monitoring-Daten und zur Administration des Monitoring verwenden. Da nur diese eine Schnittstelle zu Administrationszwecken verwendet wird, spricht man auch von einem Single-Point-of-Administration (SPoA; dt.: einzelner Administrationspunkt).

Bei großen Mengen an Monitoring-Daten kann eine zentrale Komponente jedoch zu einem Engpass bzgl. der Netzwerkbandbreite, Speicherung der Daten, oder Verarbeitungsgeschwindigkeit führen. Daher existieren als eine Ausprägung zentralisierter Monitoring-Architekturen N-tier-Architekturen. N-tier-Architekturen bestehen aus einer hierarchischen Baumstruktur der Tiefe n mit einem zentralen Monitoring-Server als Wurzel und den überwachten Ressourcen als Blätter des Baums. Die dazwischen liegenden Knoten (für $n \geq 3$) stellen vollständige, autonome Monitoring-Systeme dar, die als *zentralisierte* Server für ein Subset der darunter im Baum liegenden überwachten Ressourcen gelten. Alle diese zwischengeschalteten Monitoring-Systeme senden die erhaltenen Daten zu ihrem Elternknoten weiter, bis diese am

Wurzelknoten, d. h. dem eigentlichen zentralen Monitoring-Server, ankommen. Auf diesen Monitoring-Server greifen die Systemadministratoren zu, um die Monitoring-Daten vollständig zu erfassen.

In vollständig dezentralisierten Monitoring-Architekturen existiert keine zentrale Komponente mehr, d. h., auf jeder überwachten Ressource existiert ein Monitoring-System, das mit den anderen Monitoring-Systemen kommunizieren kann. Die oben beschriebenen Engpässe werden dadurch beseitigt. Allerdings existiert in dezentralen Monitoring-Systemen kein SPoA mehr, was die Anzahl zu administrierender Monitoring-Systeme erhöht und daher einen erhöhten Aufwand für Systemadministratoren darstellt.

2.3.2 Erfassung von Monitoring-Daten

Bei der Erfassung von Monitoring-Daten unterteilen sich Monitoring-Systeme in agentenlos und agentenbasiert [Lig12b]. Ligus definiert einen Agenten als *„Software-Prozess, der Monitoring-Daten sammelt und sie an ein Monitoring-System meldet“* [Lig12c] und betrachtet agentenbasiertes Monitoring als das standardmäßige Vorgehen. Diese Agenten sind zu unterscheiden von Agenten im Kontext von Multi-Agenten-Systemen nach der Definition von Woolridge [Woo99], welche autonome Entscheidungen treffen, um gemeinsam definierte Ziele zu erreichen.

Beim agentenbasierten Monitoring muss zunächst auf jeder zu überwachenden Ressource, z. B. einer VM, ein Agent installiert werden, der daraufhin detaillierte Informationen der Ressource, z. B. CPU-Auslastung, an das Monitoring-System weiterleitet. Für jeden Agenten ist über seine Konfiguration definiert, (i) welche Metriken er überwachen soll, (ii) wie oft er für diese Metriken Messwerte sammeln soll, und (iii) wohin er die erfassten Monitoring-Daten weiterleiten soll. Zusätzlich besitzt ein Agent oftmals Funktionalitäten wie das Aggregieren und Filtern von Monitoring-Daten, um eine Reduktion des Netzwerkverkehrvolumens zu erreichen [ABdP13].

Agentenloses Monitoring verzichtet auf den Einsatz von Agenten und nutzt bspw. das Simple Network Management Protocol (SNMP; dt.: Einfaches Netzwerkverwaltungsprotokoll), um Netzwerkgeräte zu überwachen. Der

Vorteil agentenloser Monitoring-Systeme liegt darin, dass die Installation und Wartung von Agenten entfällt. Hingegen ist die Erfassung detaillierter Informationen der überwachten Ressourcen oftmals nicht möglich. Allerdings ist agentenloses Monitoring auch nicht als Alternative zu agentenbasiertem Monitoring zu betrachten, sondern kann parallel verwendet werden, z. B. wenn agentenbasiertes Monitoring aufgrund von Restriktionen bzgl. der Installation zusätzlicher Software nicht möglich ist.

Unabhängig von der Art der Erfassung existieren verschiedene Arten von Monitoring-Daten. Diese können z. B. numerische Daten in Form von Zeitreihen (bspw. CPU-Auslastung) oder alpha-numerische Daten in Form von unstrukturiertem Text (bspw. Log-Dateien von Anwendungen) sein.

Je nach Ursprung der Monitoring-Daten werden Monitoring-Systeme in verschiedene Gruppen mit unterschiedlichen Fokussen unterteilt. Dazu gehören bspw. IT-Infrastruktur-Monitoring (ITIM)-Systeme, Application-Performance-Monitoring (APM)-Systeme oder Netzwerk-Monitoring-Systeme. ITIM-Systeme fokussieren sich demnach auf das Monitoring der IT-Infrastruktur, z. B. von virtuellen Maschinen und APM-Systeme auf das Monitoring von Anwendungen.

2.3.3 Alerting

Unabhängig davon, welche Monitoring-Architektur, welche Art der Erfassung oder welchen Fokus ein Monitoring-System hat, ist Alerting ein wesentlicher Bestandteil [Lig12c]. Es zählt zu den Hauptaufgaben eines Monitoring-Systems und beschreibt die automatische Benachrichtigung der Systemadministratoren, wenn Probleme in der überwachten Umgebung erkannt werden. Beispielsweise wird ein Systemadministrator alarmiert, wenn die CPU-Auslastung einer VM über 90 % beträgt. Hierfür werden in dem Alerting-Framework des Monitoring-Systems Alerting-Regeln definiert, die aus einer Regelbedingung und einer Aktion bestehen. Bei Verletzung der Regel wird demnach die entsprechende Aktion ausgeführt. Das eben beschriebene Beispiel würde daher zu einer Regel wie in Listing 2.1 führen.

```
IF VM.CPU > 90 % THEN send email to SysAdmin
```

Listing 2.1: Beispielhafte Alerting-Regel

Diese Alerting-Regel wird kontinuierlich auf neu eintreffenden Monitoring-Daten ausgewertet und ermöglicht somit eine passive Überwachung, d. h., in diesem Beispiel müssen Systemadministratoren nicht dauerhaft aktiv die CPU-Auslastungen der VMs im Auge behalten. Die Benachrichtigung der Systemadministratoren wird als *Alert* bezeichnet, der Informationen zum erkannten Problem – dem *Event* – beinhaltet. Alternativ können zunächst die erkannten Events an ein Event-Management-System weitergeleitet werden, welche auftretende Events aus mehreren Monitoring-Systemen konsolidieren, analysieren und über die entsprchenden Reaktionen darauf entscheiden.

2.4 Situationserkennung

In diesem Abschnitt wird die Überwachung industrieller Umgebungen mittels einer Situationserkennung vorgestellt. Die in der vorliegenden Arbeit verwendete Methode stammt aus dem Forschungprojekt SitOPT [Uni21] und basiert auf Situationstemplates [HHL+10]. Die Situationstemplates wurden von Hirmer et al. [HWS+16] erweitert, um damit Situationen modellieren und erkennen zu können. Die Situationserkennung funktioniert auf der Basis von Sensordaten, die über Sensoren der industriellen Assets erfasst werden und kann somit als Bestandteil des Condition-Monitoring betrachtet werden. Für jedes industrielle Asset kann ein entsprechendes Situationstemplate modelliert werden, das unerwünschte Situationen für das Asset modelliert.

In Abbildung 2.1 ist ein beispielhaftes Situationstemplate für das Monitoring einer Bohrmaschine zu sehen. Ein Situationstemplate besteht aus Kontext-, Bedingungs-, Operations-, und Situationsknoten. Das industrielle Asset sendet kontinuierlich Sensordaten an die Situationserkennung, die das Situationstemplate auswertet. Im Kontextknoten werden die Sensordaten (z. B. zur Werkzeugabnutzung der Bohrmaschine und zum vorhandenen Material) modelliert und bei der Ausführung erfasst. Im Bedingungsknoten werden Bedingungen an die einzelnen Sensorwerte gestellt. Beispielsweise

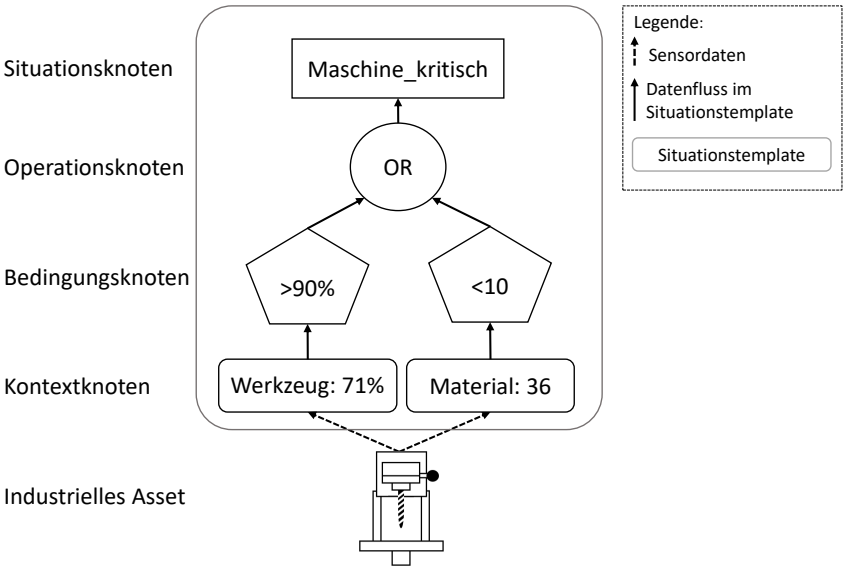

Abbildung 2.1: Situationstemplate zur Überwachung eines industriellen Assets

soll die Werkzeugabnutzung nicht über 90 % steigen und das verfügbare Material darf nicht unter zehn sinken. Mittels Operationsknoten können die Bedingungsknoten über logische Operatoren *AND, OR, XOR* verknüpft werden. Es können mehrere *Schichten* an Operationsknoten vorhanden sein, die sowohl Bedingungsknoten als auch Operationsknoten miteinander verknüpfen. Ein letzter Operationsknoten fasst alle darunterliegenden Knoten zusammen und leitet das Ergebnis des logischen Ausdrucks an den Situationsknoten. Im Situationsknoten wird die erkannte Situation als Boolscher Wert (*wahr* oder *falsch*) dargestellt. Auf diese Weise können beliebige logische Ausdrücke definiert werden, die den Alerting-Regeln aus dem vorherigen Abschnitt ähneln.

Die Situationstemplates liegen nach der Modellierung als XML-Dateien vor. Ein XML-Schema wurde erstellt, um modellierte Situationstemplates auf Vali-

dität prüfen zu können. Zur Ausführung der Situationserkennung können verschiedene Streaming-basierte Systeme wie z. B. Complex-Event-Processing (CEP)-Systeme zum Einsatz kommen. Hierfür werden die Situationstemplates in eine entsprechende, ausführbare Repräsentation überführt (z. B. in CEP-Queries). Erkannte Situationen können anschließend von situationsadaptiven Anwendungen (z. B. Workflows) verwendet werden, um entsprechend darauf zu reagieren [WSBL15]. Weitere zur Situationserkennung benötigten Systeme wie die Sensorregistrierung und -verwaltung [HWBM16a] wurden ebenso im SitOPT-Projekt behandelt, werden im Rahmen dieser Arbeit allerdings nicht weiter betrachtet.

Kapitel 3

Übersicht der Beiträge

In diesem Kapitel wird eine Übersicht der Beiträge der vorliegenden Arbeit gegeben. Die Komplexität des Monitoring hat sich in diesem Jahrzehnt aufgrund neuer Technologien wie Cloud-Computing, Edge-Computing, und I4.0 stark erhöht. Dies kann verschiedene Herausforderungen zu Folge haben, wie in Kapitel 1 beschrieben. Beispielsweise können (i) Lock-ins hinsichtlich der verwendeten Monitoring-Systeme entstehen, (ii) die bei der Überwachung entstehenden hochvoluminösen Monitoring-Daten die Netzwerkbandbreite stark belasten, und (iii) der Einsatz mehrerer Monitoring-Systeme zu Datensilos führen, die eine umfassende Sicht auf die überwachte Umgebung erschweren. Um diesen Herausforderungen entgegenzutreten, sollen bestehende Ansätze, die für das Monitoring verwendet werden, verbessert werden. Die Heterogenität der überwachten Ressourcen, d. h. industrielle Assets und Cloud-Umgebungen, erfordern jedoch aufgrund verschiedener Anforderungen an ihre Überwachung unterschiedliche Lösungsansätze. Daher befasst sich diese Dissertation mit den folgenden zwei Bereichen: (1) dem Monitoring von Cloud-Umgebungen auf Basis agentenbasierter Monitoring-Systeme und (2) dem Monitoring industrieller Assets mittels Situationserkennung

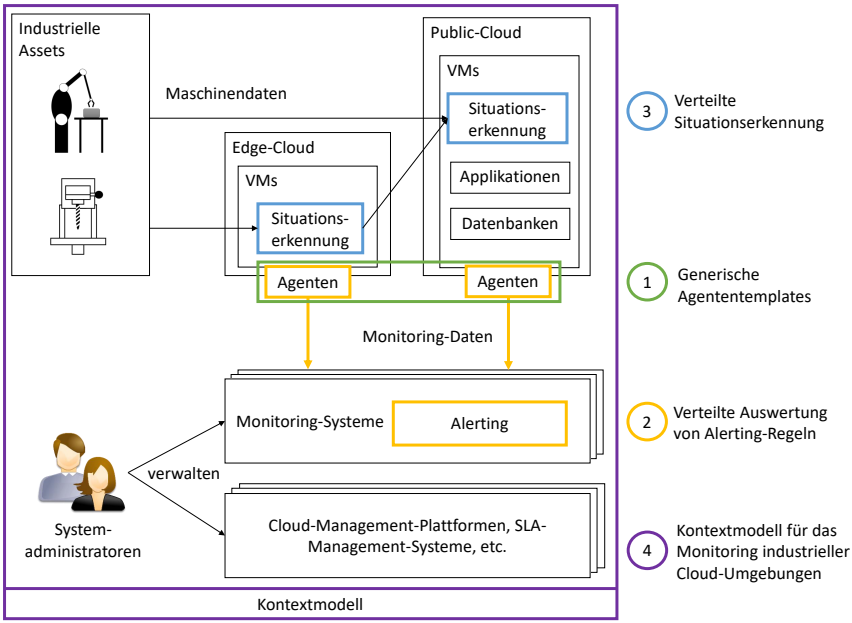

Abbildung 3.1: Übersicht der Beiträge dieser Arbeit

auf Basis von Situationstemplates. Um eine umfassende Übersicht auf eine überwachte industrielle Cloud-Umgebung zu ermöglichen, werden zuletzt alle relevanten Aspekte Daten in einem Kontextmodell abgebildet.

In Abbildung 3.1 werden die Beiträge dieser Dissertation innerhalb einer typischen IT-Architektur für das Monitoring industrieller Assets und Cloud-Umgebungen dargestellt, wie sie in aktuellen Forschungsprojekten mit mehreren Industriepartnern, wie bspw. IC4F [Stu21], üblich ist. Die farbig markierten Bereiche stellen die Beiträge dieser Dissertation dar. Links oben in der Abbildung beginnt das Monitoring industrieller Assets auf dem Shopfloor einer Fabrik mittels einer Situationserkennung (blau markiert). Hierfür werden die Maschinendaten zunächst an eine geeignete Ausführungsumgebung, z. B. eine Public- oder Edge-Cloud, gesendet, um Störfälle wie z. B. eine zu hohe Drehzahl einer Bohrmaschine, entsprechend

zu erkennen und darauf zu reagieren. Die Situationserkennung wird innerhalb der VMs der jeweiligen Clouds ausgeführt und muss ebenso auf Ausfall überwacht werden. Daher werden alle virtuellen Maschinen der Public- und Edge-Clouds sowie darauf installierte Applikationen, Datenbanken, etc., fortlaufend von Agenten überwacht (grün markiert), welche die gesammelten Monitoring-Daten an Monitoring-Systeme weiterleiten. Daraufhin können Systemadministratoren auf die Daten zugreifen und sich einen Überblick über den Zustand der Cloud-Umgebung verschaffen. Darüber hinaus sorgen Alerting-Funktionen der Monitoring-Systeme für eine automatische Benachrichtigung der Systemadministratoren im Falle von unerwünschten Zuständen in der Cloud-Umgebung (gelb markiert), z. B. eine zu hohe CPU-Auslastung bei den VMs. Parallel hierzu sind weitere Systeme wie z. B. Cloud-Management-Plattformen zum Management von Private- oder Hybrid-Clouds oder Service-Level-Agreement-Systeme im Einsatz, die wertvolle Informationen zur Cloud-Umgebung liefern. Das umfassende Kontextmodell (lila markiert) ist zunächst nicht enthalten und ein neuer Beitrag dieser Dissertation, während die restlichen Beiträge bestehende Ansätze nutzen, verbessern oder erweitern.

Diese Dissertation umfasst folgende vier Beiträge (s. Abbildung 3.1):

B1: Generische Agententemplates

B2: Verteilte Auswertung von Alerting-Regeln

B3: Verteilte Situationserkennung

B4: Kontextmodell für das Monitoring industrieller Cloud-Umgebungen

Der erste **Beitrag B1** stellt generische Agententemplates vor, um die Gefahr eines Vendor Lock-ins bei Monitoring-Systemen zu verringern. Der Beitrag fokussiert sich hierbei auf die Agenten von Monitoring-Systemen und damit auf das Monitoring der Cloud-Umgebung. Der Grundgedanke ist es, den Austausch eines Monitoring-Systems zu erleichtern. Die Gründe hierfür können vielfältig sein, bspw. soll ein neues Monitoring-System verwendet werden, das Machine-Learning-Konzepte anwendet, um zuvor unbekannte Verhaltensweisen der Cloud-Umgebung zu entdecken. Jedoch

sind Monitoring-Systeme über deren Agenten in der zu überwachenden IT-Umgebung integriert und ein Austausch kann sehr zeitaufwändig und fehleranfällig sein. Daraus resultieren auch hohe Kosten, was in Summe die Möglichkeiten vieler IT-Abteilungen übersteigt. Um den Austausch von Monitoring-Systemen zu unterstützen, werden (1) generische Agententemplates (s. Abbildung 3.1 (grün)) vorgestellt, um eine Abstraktionsstufe zur Modellierung von Agenten zu erschaffen. Statt für jedes neu verwendete Monitoring-System entsprechende Agenten zu modellieren, wird stattdessen ein Agententemplate ein einziges Mal abstrakt und generisch modelliert. Durch von Domänenexperten der jeweiligen Monitoring-Systeme bereitgestellte Transformationslogik können die abstrakten Agenten in ausführbare, lösungsspezifische Agenten des gewählten Monitoring-Systems umgewandelt werden. Der Modellierungsprozess wird durch die Einführung einer Agenten-Pipeline, die alle relevanten Komponenten eines Agenten inkludiert, unterstützt. Des Weiteren wird ein erweiterter Lebenszyklus für Agenten vorgestellt, um eine fortlaufende Anpassung an heutige, dynamische Cloud-Umgebungen zu ermöglichen.

Auf Basis des ersten Beitrags wird in **Beitrag B2** *DEAR* vorgestellt – ein Plug-in für Monitoring-Systeme zur verteilten Auswertung von Alerting-Regeln (en.: **D**istributed **E**valuation of **A**lerting **R**ules). DEAR ist dafür verantwortlich, auf die im Monitoring-System definierten Alerting-Regeln zuzugreifen und diese anschließend so zu transformieren, dass diese direkt auf den VMs evaluiert werden können. Entgegen dezentralisierter Monitoring-Ansätze soll weiterhin ein SPoA erhalten bleiben, um die Management-Komplexität nicht zu erhöhen. Da DEAR auf generischen Agententemplates basiert, entsteht auch mit DEAR ein technologie- und lösungsunabhängiger Ansatz, der eine Vielzahl von Monitoring-Systemen unterstützt.

Aufgrund der hohen Heterogenität industrieller Assets, z. B. verschiedene Sensorik und Schwellwerte für die Problemerkennung, haben sich bisher keine allgemein verwendbaren Systeme etabliert wie es beim Monitoring im Bereich des Cloud-Computing der Fall ist. Daher werden im Rahmen dieser Arbeit für das Monitoring industrieller Assets Situationstemplates (s. Kapitel 2) verwendet, die individuell hinsichtlich der Eigenschaften der

jeweiligen Assets wie z. B. Drehzahl einer Bohrmaschine, modelliert werden können. In **Beitrag B3** wird der bestehende Ansatz der Situationserkennung mittels Situationstemplates erweitert, um eine Verteilung der Situationserkennung zu ermöglichen. Hier werden die Vorteile des Edge-Computing zu Nutze gemacht, um eine Situationserkennung nahe an den industriellen Assets stattfinden zu lassen und somit eine niedrigere Latenz und eine Reduzierung des Netzwerkverkehrs zwischen dem Shopfloor einer Fabrik und der Public-Cloud zu ermöglichen. Im Vergleich zu **Beitrag B2** ist es allerdings nicht immer möglich bzw. nicht immer vorteilhaft, die Situationserkennung ausschließlich auf der Edge, d. h., so nahe wie möglich an den Datenquellen, auszuführen, z. B. weil nicht alle Datenquellen lokal zugreifbar sind oder keine passende Infrastruktur auf der Edge-Cloud bzw. überhaupt keine Edge-Cloud vorhanden ist. Daher werden die Vor- und Nachteile verschiedener Ausführungsstrategien auf Edge- und Public-Cloud analysiert. Als Grundlage für die Verteilung der Situationserkennung werden die Modellierung und Ausführung von Situationstemplates und der Situationserkennung adaptiert.

Der vierte **Beitrag B4** befasst sich mit der umfassenden Betrachtung einer überwachten industriellen Cloud-Umgebung. Eine Vielzahl verwendeter Monitoring- und Management-Systeme resultiert in Datensilos, die von verschiedenen Abteilungen verwaltet werden, unter denen oftmals wenig bis gar keine Kommunikation stattfindet [NKFW19]. Wie und ob diese Daten miteinander in Beziehung zueinander stehen, ist daher oftmals unbekannt und erschwert bspw. die Suche nach der Fehlerursache eines aufgetretenen Problems. Daher wird in diesem Beitrag ein Kontextmodell erstellt, welches die überwachten Ressourcen, die Monitoring- und Management-Daten zur Beschreibung dieser Ressourcen, und die jeweiligen Systeme zur Verwaltung bzw. Erfassung dieser Daten enthält. Des Weiteren werden auf den Daten aufbauende Analysen und daraus resultierende Aktionsmöglichkeiten, wie z. B. Alerting, ebenfalls im Kontextmodell berücksichtigt. Auf diese Weise kann beispielsweise eine Topologie der überwachten industriellen Cloud-Umgebung modelliert werden, die Systemadministratoren einen tieferen Einblick liefern kann und somit das Monitoring verbessern kann.

Gemeinsam ergeben die **Beiträge B1-B4** ein umfassendes Monitoring industrieller Cloud-Umgebungen. Beginnend beim Shopfloor werden industrielle Assets einer Fabrik mittels einer Situationserkennung auf Edge- und Public-Cloud überwacht. Die VMs dieser Clouds werden wiederum mittels agentenbasierten Monitoring überwacht. Um niedrige Latenzen und geringen Netzwerkverkehr beim Monitoring zu ermöglichen, werden bestehende Ansätze, d. h., (i) agentenbasiertes Monitoring und (2) Situationserkennung auf Basis von Situationstemplates, erweitert. Somit wird eine verteilte Ausführung ermöglicht, sodass die für das Monitoring benötigten Daten möglichst nahe an der Datenquelle ausgewertet werden können. Aufgrund der Vielzahl verwendbarer Monitoring-Systeme, wird mit **Beitrag B1** & **B2** ein möglichst technologie- und lösungsunabhängiger Ansatz verfolgt, um auf eine Vielzahl aktueller Monitoring-Systeme eingesetzt werden zu können. Mit **Beitrag B3** hingegen wird die Situationserkennung als spezialisierte Lösung für das Monitoring industrieller Assets eingesetzt und erweitert. Dennoch wird darauf geachtet, mittels der Verwendung von Situationstemplates eine technologieunabhängige Modellierung zu ermöglichen und verschiedene Technologien für die Situationserkennung (z. B. CEP) verwenden zu können. **Beitrag B4** erlaubt eine umfassende Sicht auf eine überwachte industrielle Cloud-Umgebung, indem alle relevanten Aspekte in einem Kontextmodell erfasst werden, das Systemadministratoren beim Monitoring dieser Umgebungen unterstützen soll.

Kapitel 4

Generische Agententemplates

Die neu gewonnene Flexibilität, die durch den Einsatz von Cloud-Computing erreicht wird, stellt neue Herausforderungen an das Management und Monitoring von IT-Umgebungen. Das Hauptproblem liegt hierbei, dass die meisten aktuell verwendeten Monitoring-Systeme ursprünglich nicht für den Einsatz in hochdynamischen Umgebungen konzipiert wurden. So sind beispielsweise virtuelle Maschinen (VMs), die innerhalb von Minuten zur Verfügung stehen und ebenso wieder abgeschaltet werden können, nicht vorgesehen. Stattdessen wurden diese Monitoring-Systeme für traditionelle Hardware in sich langsam ändernden IT-Umgebungen konzipiert [CA15; TPD15]. Als Beispiel dafür, welche Unzulänglichkeiten in aktuellen Systemen vorherrschen, sei das meistverwendete Open-Source-Monitoring-System Nagios [Nag21a] genannt. Wenn eine neue VM erstellt wird, muss neben der Installation eines Agenten auch eine Änderung im Backend-System von Nagios und anschließend ein Neustart des gesamten Systems durchgeführt werden [Nag21c]. Dies ist bei oftmaligem Hinzufügen beziehungsweise Ent-

fernen von VMs weder ein skalierbarer noch adäquater Prozess in dynamischen Cloud-Umgebungen. Ein weiterer Aspekt ist beispielsweise die immer weiter verbreitete Nutzung von anspruchsvollen Analysen mittels Machine-Learning-Algorithmen wie z.B. neuronalen Netzen oder anderen Verfahren wie z.B. Clustering-Algorithmen [BMC20]. Mit diesen Verfahren werden die Monitoring-Daten analysiert, um tiefere Einblicke in die IT-Umgebung zu erhalten und beispielsweise das Verhalten von VMs vorherzusagen. Diese und weitere Aspekte können dazu führen, dass für Unternehmen ein Wechsel zu einem aktuelleren Monitoring-System vorteilhaft oder sogar erforderlich sein kann.

Monitoring-Systeme sind jedoch eng in die IT-Umgebung integriert. Agenten werden auf der gesamten IT-Umgebung installiert, um alle relevanten Daten zu erfassen. Komplexe IT-Umgebungen können allerdings sehr heterogen sein, beispielsweise sind auf den VMs verschiedene Betriebssysteme (z.B. Windows oder Linux) oder unterschiedliche Anwendungen installiert (z.B. Datenbanken oder Web-Server). Agenten müssen daher an diese Heterogenität angepasst werden, indem sie entsprechend konfiguriert werden und kontinuierlich Monitoring-Daten erfassen können. Wenn ein Unternehmen sich dazu entscheidet, aufgrund der oben genannten Aspekte ein Monitoring-System auszutauschen, müssen auch alle beteiligten Agenten ausgetauscht werden, da ein Monitoring-System in der Regel eigene Agenten verwendet.

Es ergeben sich die folgenden Herausforderungen:

- Programmiersprache: Es ist oftmals der Fall, dass verschiedene Programmiersprachen für verschiedene Monitoring-Systeme und deren Agenten verwendet werden. Systemadministratoren, die für den Austausch des Monitoring-Systems verantwortlich sind, müssen daher zunächst Syntax und Semantik der neuen Programmiersprache erlernen, falls diese nicht bekannt sind, um die neuen Agenten entsprechend konfigurieren zu können.
- Funktionalität: Auch die unterstützten Funktionen, beispielsweise die Aggregation von Monitoring-Daten, von Agenten verschiedener

Monitoring-Systeme können sich unterscheiden. Systemadministratoren müssen überprüfen, ob die neuen Agenten die benötigte Funktionalität beinhalten und wie diese angewendet wird.

- Bereitstellung: Nach der Konfiguration der neuen Agenten müssen diese auf den VMs bereitgestellt werden. Dieser Prozess wird oftmals händisch durchgeführt, da Monitoring-Systeme typischerweise keine automatische Bereitstellung unterstützen. Vor allem in großen und hochdynamischen Cloud-Umgebungen ist dieser Prozess schwerfällig und nicht skalierbar.
- IT-Abteilungen: Die bisherigen Herausforderungen werden durch den derzeitigen Zustand der IT-Abteilungen, die häufig unterbesetzt sind und denen die finanziellen Mittel für neue Technologien fehlen [Bow18; Tec17], noch weiter verstärkt. Es ist daher schwer, auf dem neuesten Stand der Technik zu bleiben und diesen im Unternehmen durchzusetzen.

Diese Herausforderungen ergeben für den Austausch von Monitoring-Systemen und deren Agenten eine komplizierte und zeitaufwändige Aufgabe. Dies ist sogar der Fall, wenn sich das zugrundeliegende Verhalten der Agenten nicht ändert, z.B. welche Metrik überwacht werden soll und wie oft oder wie die Monitoring-Daten verarbeitet werden sollen. Dies kann der Fall sein, wenn die Gründe für den Austausch des Monitoring-Systems nichts mit den Agenten zu tun haben, sondern der Austausch beispielsweise aufgrund der Unterstützung von Machine-Learning-Algorithmen oder einer besseren Plattform zur Visualisierung geschieht.

Um diese Herausforderungen zu lösen, werden in diesem Kapitel **generische Agententemplates** vorgestellt, mittels jener eine Abstraktionsstufe bei der Modellierung von Agenten eingeführt wird. Anstatt jedes Mal bei einem Wechsel des Monitoring-Systems lösungsspezifische Agenten zu modellieren, müssen Systemadministratoren Agenten stattdessen nur ein einziges Mal auf eine abstrakte und generische Art modellieren. Auf diese Weise wird nur das Verhalten modelliert, d.h., welche Metrik und wie oft sie überwacht werden soll oder wie die Monitoring-Daten verarbeitet werden sollen. Diese

abstrakten Agenten werden darauffolgend in ausführbare, lösungsspezifische Agenten überführt. Hierfür wird die entsprechende Transformationslogik benötigt, die von Systemexperten bereitgestellt werden kann. Auch dies muss nur einmalig pro zu unterstützendem Monitoring-System beziehungsweise dessen Agenten geschehen. Somit wird der zeitliche und finanzielle Aufwand für den Wechsel eines Monitoring-Systems stark reduziert. Des Weiteren wird in diesem Kapitel ein erweiterter Lebenszyklus von Agenten vorgestellt, um das Management und die automatische Bereitstellung von Agenten zu unterstützen, welches in großen, hochdynamischen Cloud-Umgebungen essentiell ist. Diese Prozesse werden mittels eines webbasierten Modellierungswerkzeugs unterstützt.

Dieses Kapitel ist eine überarbeitete Version einer Publikation des Autors [MHSM20a].

Die Gliederung des restlichen Kapitels ist wie folgt: In Abschnitt 4.1 werden verwandte Arbeiten vorgestellt. Abschnitt 4.2 beinhaltet die Anforderungen an den Beitrag dieses Kapitels und eine Analyse agentenbasierter Monitoring-Systeme. In Abschnitt 4.3 werden die generischen Agententemplates sowie der erweiterte Lebenszyklus von Agenten vorgestellt. Abschließend wird in Abschnitt 4.5 eine Zusammenfassung des Kapitels gegeben.

4.1 Verwandte Arbeiten

Die Transformation von abstrakten Modellen auf konkrete ausführbare Implementierungen ist ein gebräuchliches Mittel, um von technischen Details zu abstrahieren. Folglich sind Benutzer in der Lage, Modelle auf einem hohen Abstraktionsniveau zu erstellen, ohne technische Kenntnisse über deren Realisierung zu benötigen. Die Verwendung eines einzigen abstrakten Modells führt zu Generizität und verringert die Gefahr eines Vendor Lock-ins, da es nicht von bestimmten Technologien abhängig ist. Im Folgenden werden Ansätze beschrieben, die einen ähnlichen Ansatz für die Transformation von abstrakten Modellen in konkrete Modelle anstreben.

Falkenthal et al. [FBB+14] zielen auf einen generischen Ansatz ab, um abstrakte Mustersprachen in konkrete Lösungsimplementierungen zu transformieren. Da ihr Ansatz generisch ist, konzentrieren sie sich nicht auf eine bestimmte Domäne, sondern diskutieren vielmehr, wie eine solche Transformation allgemein durchgeführt werden kann. Mit generischen Agententemplates wird dieser Ansatz auf die konkrete Domäne des Monitoring angewendet, indem eine Abbildung generischer Agenten auf konkrete, ausführbare Implementierungen lösungsspezifischer Agenten eingeführt wird.

Eilam et al. [EEKS11] stellen einen Ansatz für die modellbasierte Bereitstellung und Verwaltung von Anwendungen vor. Durch Transformationen werden Anwendungstopologien auf verschiedene Abstraktionsebenen abgebildet, um schließlich ausführbare Implementierungen zu erstellen, die eingesetzt werden können. Eilam et al. erfordern jedoch eine Vormodellierung konkreter Implementierungsartefakte, was bei generischen Agententemplates nicht notwendig ist.

Ähnliche Ansätze bezüglich der Agentenvorlagen werden von Künzle et al. [Kün+11] und Cohn et al. [CH09] vorgestellt, die artefakt-zentrierte Ansätze verwenden. In diesen Ansätzen werden Business-Artefakte erzeugt, d. h. abstrakte Darstellungen von Software-Komponenten mit dem Ziel, technische Details zu verbergen. Diese Artefakte lassen sich auf konkrete ausführbare Implementierungen abbilden, wie von Sun et al. [Sun+14] gezeigt wurde. Reimann [RSM14] stellt generische Muster für Simulations-Workflows vor, die auch auf konkrete ausführbare Implementierungen abgebildet werden, in seinem Ansatz auf Workflow-Fragmente, die in der Business Process Execution Language (BPEL) [WCL+05] bereitgestellt werden. Hirmer [Hir18] beschreibt eine anforderungsbasierte Modellierung und Ausführung von Datenflussmodellen. Er ermöglicht es, Datenverarbeitungsszenarien abstrakt zu modellieren und diese Modelle auf einer Modelltransformationsebene auf verschiedene ausführbare Repräsentationen abzubilden. Dies soll Benutzern bei der Erstellung komplexer Datenanalysen unterstützen.

Zuletzt stellen mehrere Monitoring-Systeme die Templates zur Modellierung ihrer lösungsspezifischen Agenten über grafische Oberflächen oder zum Download bereit [Mic20; Ora21; Sol20]. Diese Templates ermöglichen

allerdings nur die Modellierung der jeweils lösungsspezifischen Agenten. Im Vergleich dazu konzentriert sich dieser Ansatz auf eine generische Modellierung, um eine Vielzahl von Monitoring-Systemen zu unterstützen.

4.2 Grundlagen und Anforderungen

Unabhängig davon, welches Monitoring-System in einem Unternehmen zur Überwachung der IT-Umgebung genutzt wird, bleiben grundlegende Konzepte und Methoden bei agentenbasierten Monitoring-Systemen gleich. In Abbildung 4.1 wird anhand des Monitoring einer VM ein grober Überblick über agentenbasiertes Monitoring dargestellt. Beim Monitoring einer VM werden üblicherweise mehrere Metriken überwacht, beispielsweise die CPU- und Speicherauslastung, Disk I/O, und weitere, anhand derer Systemadministratoren (Admins) einen Eindruck über den "Gesundheitszustandëiner VM gewinnen können. Der Agent wird auf der VM installiert und erstellt regelmäßig und fortlaufend Messungen für diese Metriken und leitet die gesammelten Monitoring-Daten an ein Monitoring-System weiter, wo diese Daten in einer Datenbank für zukünftige Zwecke gespeichert werden. Beispiele hierfür sind Alerting oder die Visualisierung der Monitoring-Daten, um Systemadministratoren weitestgehend bei dem Management der IT-Umgebung zu unterstützen.

Systemadministratoren müssen zunächst den Agenten modellieren, d. h., dessen Konfiguration definieren. Diese Konfiguration bestimmt das Verhalten des Agenten, z. B. welche Metriken er überwacht und wie oft er Messungen durchführt. Oftmals ist der zentralisierte Monitoring-Server, an welchen die Monitoring-Daten weitergeleitet werden, nicht im selben lokalen Netzwerk wie die Agenten. Dies ist beispielsweise der Fall, wenn mehrere Cloud-Umgebungen von einem einzigen Monitoring-System überwacht werden. Daher kann die Übertragung der Monitoring-Daten bei großen Datenmengen das Netzwerk des Unternehmens belasten. Eine höhere Bandbreite, um diesem Problem entgegenzutreten, ist hingegen wiederum mit höheren Kosten verbunden. Daher unterstützen Agenten oftmals Funktionalitäten wie das

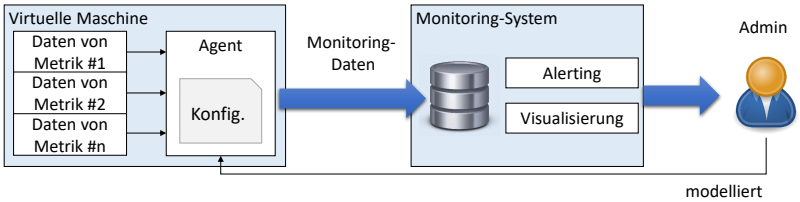

Abbildung 4.1: Monitoring einer virtuellen Maschine mittels agentenbasiertem Monitoring

Filtern und Aggregieren von Monitoring-Daten, um das Netzwerkverkehrvolumen stark zu reduzieren [ABdP13]. Systemadministratoren könnten beispielsweise kein Interesse an Messungen des verfügbaren Speichers haben, wenn deren Werte über 500 MB liegen, da dies nicht als Problem betrachtet werden könnte. Diese Messwerte könnten aus den Monitoring-Daten vom Agenten herausgefiltert werden, bevor dieser die Daten an den Monitoring-Server weiterleitet. Ebenso benötigen Systemadministratoren möglicherweise nicht unbedingt alle 10 s neue Messwerte für die CPU-Auslastung und ein Messwert alle 60 s würde ausreichen. In diesem Fall ist es ratsam, den Agenten dennoch alle 10 s messen zu lassen und diese Werte anschließend über den Zeitraum von 60 s zu aggregieren (z. B. mit der Berechnung des Durchschnitts), um Durchschnittswerte zu erhalten statt punktuelle Momentaufnahmen.

Die Konfigurationen der Agenten müssen auf die jeweiligen Spezifikationen der VMs zugeschnitten werden, um ein adäquates Monitoring zu gewährleisten. VMs können sich z. B. im Betriebssystem unterscheiden oder verschiedene Anwendungen wie Web-Server und Datenbanken beherbergen. In großen, komplexen Cloud-Umgebungen, wo der Grad an Heterogenität sehr viel größer sein kann, existieren daher unter Umständen eine Vielzahl verschiedener Konfigurationen von Agenten parallel.

Um all das zu ermöglichen, existiert eine Vielzahl an Monitoring-Systemen, aus denen Systemadministratoren wählen können. Die Wahl eines Monitoring-Systems kann auf mehreren Faktoren basieren, wie z. B. Funk-

tionalitäten bezüglich der Visualisierung der Monitoring-Daten oder des Alerting, oder Analysemethoden mittels neuronaler Netze oder Clustering. Des Weiteren liefern die Monitoring-Systeme in der Regel ihre eigenen Agenten, die sich in Bezug auf Funktionalitäten wie Filtern und Aggregieren unterscheiden. Da der Beitrag dieses Kapitels die Erstellung einer Abstraktionsstufe für die Modellierung von Agenten zum Ziel hat, wird zunächst eine Analyse bestehender Monitoring-Systeme und deren Agenten durchgeführt, um zu ermitteln, welche Funktionalitäten generische Agenten unterstützen müssen. Um eine Vielfalt an ausgewählten Monitoring-Systemen zu erhalten, wurden sowohl Open-Source als auch proprietäre Systeme ausgewählt sowie neue als auch alte Monitoring-Systeme. Da der Fokus auf den Agenten eines Monitoring-Systems liegt, sind agentenlose Monitoring-Systeme nicht Teil der Analyse. Aceto et. al [ABdP13] geben an, dass das Filtern und Aggregieren von Monitoring-Daten direkt auf dem Agenten unterstützt wird, um den Netzwerkverkehr zur verringern, der beim Weiterleiten an den Monitoring-Server entsteht. Daher wurden die Agenten in Bezug auf das Filtern (F) und Aggregieren (A) von Monitoring-Daten analysiert.

Darüber hinaus sind Bereitstellung oder Konfigurationsänderungen der Agenten umständlich und fehleranfällig, insbesondere in großen Cloud-Umgebungen. Daher ist die Automatisierung dieser Prozesse zwingend erforderlich [BMC20]. Es wurde untersucht, ob die automatische Bereitstellung (B) von Agenten auf VMs von den Monitoring-Systemen nativ unterstützt wird (d. h., nicht ausschließlich durch externe Anwendungen Dritter, wie z. B. Chef[1] oder Puppet[2]).

Die Ergebnisse der Analyse zeigen, dass das Filtern und Aggregieren von Monitoring-Daten oftmals unterstützt wird, wogegen die automatische Bereitstellung der Agenten von keinem einzigen Monitoring-System unterstützt wird.

Systemadministratoren müssen sich für eines der Monitoring-Systeme entscheiden und die Agenten entsprechend konfigurieren. Jedoch können sich zukünftig die Geschäftsbedürfnisse oder die Cloud-Umgebung ändern. Bei-

[1] https://www.chef.io/
[2] https://puppet.com/

Tabelle 4.1: Analyse von Monitoring-Systemen in Bezug auf das Filtern (F), Aggregieren (A), und die automatische Bereitstellung (B) von Agenten

Monitoring-System	**Agent**	**F**	**A**	**B**
Nagios [Nag21a]	NCPA [Nag21b]	✓	✓	✗
TICK Stack [Inf21c]	Telegraf [Inf21b]	✓	✓	✗
ELK Stack [Ela21a]	Beats [Ela21b]	✓	✗	✗
BMC TSIM [BMC18]	PATROL [BMC14]	✓	✓	✗
MoogSoft [Moo21b]	Observer [Moo21a]	✓	✓	✗
Sensu [Sen21a]	Sensu Client [Sen21b]	✓	✗	✗
Splunk [Spl21b]	Forwarder [Spl21a]	✗	✗	✗
Icinga [Ici21a]	Icinga Agent [Ici21b]	✓	✓	✗
Datadog [Dat21a]	Datadog Agent [Dat21b]	✗	✓	✗
Prometheus [Pro21b]	Collector [Pro21a]	✗	✗	✗
Zabbix [Zab21a]	Zabbix agent [Zab21b]	✓	✓	✗
PandoraFMS [Pan21a]	Pandora agent [Pan21b]	✓	✗	✗
AWS CloudWatch [Ama21a]	CloudWatch Agent [Ama21b]	✗	✓	✗

spielsweise könnten Linux-Server durch Windows-Server ersetzt werden oder das Unternehmen entscheidet sich dazu, fortgeschrittene Analysemethoden für das Monitoring einsetzen zu wollen. Das derzeitige Monitoring-System könnte daraufhin obsolet werden, wenn es die benötigten Voraussetzungen nicht erfüllt und ein Austausch des Monitoring-Systems muss durchgeführt werden. Das hat jedoch zur Folge, dass auch alle Agenten und deren Konfigurationen ersetzt werden müssen. Wie bereits oben angesprochen, unterscheiden sich die Agenten in der gewählten Programmiersprache und den Funktionalitäten, die zunächst von den Systemadministratoren erlernt werden müssen. Anschließend müssen die Agenten wieder auf den VMs bereitgestellt werden, was oftmals manuell geschieht und vor allem bei großen Cloud-Umgebungen zu Skalierbarkeitsproblemen führt. All das führt zu einem großen Zeit- und Kostenaufwand, auch wenn die zugrundeliegende Entscheidung, die zum Austausch des Monitoring-Systems geführt hat, nichts mit den Agenten zu tun hatte.

Folgende Anforderungen können aus der Analyse bestehender Systeme abgeleitet werden:

*A*1: **Generizität:** Statt lösungsspezifische Agenten zu modellieren, wird ein generischer Ansatz zum Modellieren von Agenten benötigt, der von den technischen Details der jeweiligen lösungsspezifischen Agenten abstrahiert. Auf diese Weise soll es ermöglicht werden, dass Systemadministratoren einen generischen Agenten nur ein einziges Mal modellieren müssen, ohne tieferes Verständnis der lösungsspezifischen Agenten zu besitzen. Systemexperten für die jeweiligen Monitoring-Systeme kennen die technischen Details und können die benötigte Transformationslogik liefern, um die generischen Agenten in lösungsspezifische Agenten transformieren zu können. Auch dies muss nur ein Mal pro unterstütztes Monitoring-System bereitgestellt werden. Somit können Systemadministratoren kostbare Zeit sparen und sich darauf fokussieren, generische Agenten bedarfsgerecht zu modellieren, anstatt sich mit den technischen Details lösungsspezifischer Agenten zu befassen.

*A*2: **Flexibilität:** Wie in Tabelle 4.1 zu sehen ist, unterstützt nicht jeder Agent die Filterung und Aggregation von Monitoring-Daten. Daher ist es möglich, generische Agenten zu modellieren, die nicht auf bestimmte, lösungsspezifische Agenten abbildbar sind. Um Systemadministratoren weder bei der Modellierung generischer Agenten noch bei der Wahl lösungsspezifischer Agenten einzuschränken, soll eine Transformation auch trotz fehlender Funktionalität ermöglicht werden.

*A*3: **Management von Agenten:** Mit wachsender Komplexität und Größe von Cloud-Umgebungen, müssen manuelle Aufgaben minimiert werden. Die Bereitstellung von Agenten muss automatisiert erfolgen. Darüber hinaus müssen auch Änderungen an bereits modellierten generischen Agenten und an bereits bereitgestellten, lösungsspezifischen Agenten auf den VMs propagiert werden. Des Weiteren wird ein Repository für das Management von generischen Agenten benötigt, um auf bereits Modellierte zuzugreifen und bei Bedarf Änderungen

vorzunehmen. Das Modellieren von generischen Agenten soll zudem mit einem grafischen Modellierungswerkzeug unterstützt werden, um diesen Prozess weiter zu vereinfachen.

4.3 Generische Agententemplates

In diesem Abschnitt werden die generischen Agententemplates vorgestellt, welche die im vorigen Abschnitt definierten Anforderungen $A1 - A3$ erfüllen sollen. Systemadministratoren modellieren Agenten entsprechend den Geschäftsbedürfnissen und den jeweiligen Spezifikationen der VMs und entscheiden, welche Ressourcen wie oft überwacht werden müssen und welche Funktionen auf den Monitoring-Daten durchgeführt werden sollen. In Abbildung 4.2 (links) ist zu sehen, dass die Systemadministratoren für jeden lösungsspezifischen Agenten von vorne beginnen und die Agenten neu modellieren müssen, auch wenn das zugrundeliegende Verhalten des Agenten (z. B. *CPU-Auslastung messen, Durchschnitt über 60 s berechnen, an Datenbank schicken*) unverändert bleibt.

Mithilfe von generischen Agententemplates (s. Abbildung 4.2 (rechts)) müssen Systemadministratoren hingegen nur dieses zugrundeliegende Verhalten einmalig in Form eines generischen Agenten modellieren. Dieser generische Agent ist jedoch nicht ausführbar und muss zunächst in einem lösungsspezifischen Agenten transformiert werden. Da generische Agenten einem vordefinierten Schema entsprechen, können Systemexperten, die detaillierte Kenntnisse über die verschiedenen lösungsspezifischen Agenten besitzen, die Transformationslogik bereitstellen, um generische Agenten in ausführbare Agenten für entsprechende Monitoring-Systeme zu überführen. Die Bereitstellung dieser Transformationslogik auf einer öffentlichen Plattform (z. B. GitHub) ermöglicht es, dass diese für verschiedene Szenarien unternehmensübergreifend wiederverwendbar sind und somit nur ein einziges Mal bereitgestellt werden muss pro unterstütztem Monitoring-System.

Des Weiteren wird ein erweiterter Lebenszyklus für Agenten vorgestellt (s. Abbildung 4.3). Wieder ist links in der Abbildung der derzeitige

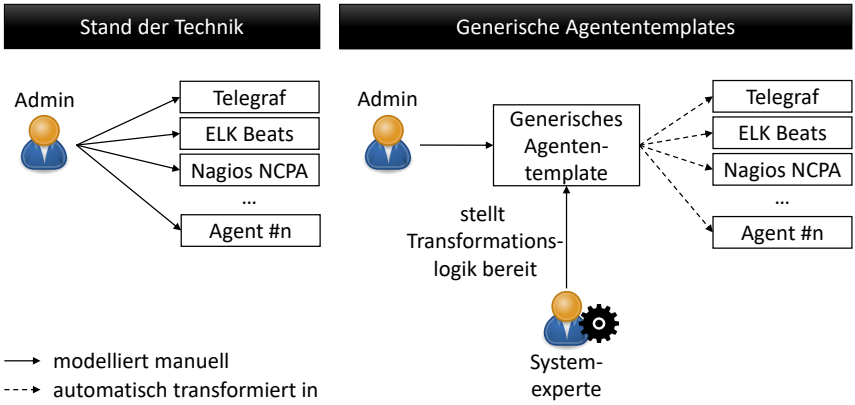

Abbildung 4.2: Links: Admin muss *n* Agenten modellieren; Rechts: Admin muss nur einen generischen Agenten modellieren

Stand der Technik gezeigt, der die Schritte *Modellierung* und *Bereitstellung* enthält. Beide Schritte werden manuell von den Systemadministratoren durchgeführt. Dieser Prozess wird jedes Mal wiederholt, sobald ein neuer Agent modelliert wird oder ein bestehender verändert wird. Daher wird ein erweiterter Lebenszyklus vorgestellt (s. Abbildung 4.3 (rechts)), welcher folgende vier Schritte beinhaltet:

- *Schritt 1 - Modellierung:* Der Anspruch an Generizität erfordert Änderungen im Modellierungsprozess. Hierfür werden generische Agententemplates genutzt, um eine Abstraktionsstufe einzuführen, welche die einfache Modellierung von generischen Agenten ermöglicht, die darauffolgend in mehrere lösungsspezifische Agenten überführt werden können.
- *Schritt 2 - Transformation:* Da die modellierten Agenten generisch und nicht ausführbar sind, müssen diese vor der Bereitstellung auf den VMs zunächst in ausführbare, lösungsspezifische Agenten transformiert

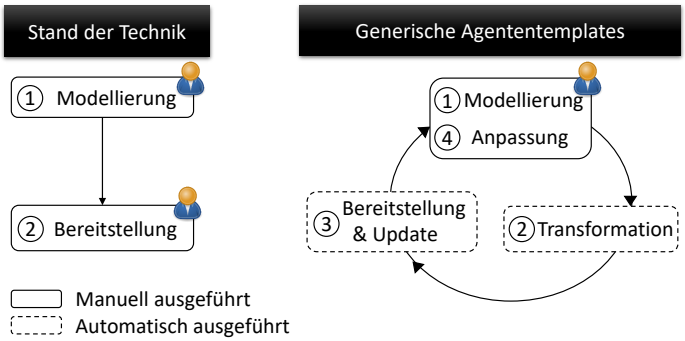

Abbildung 4.3: Links: Derzeitiger Agenten-Lebenszyklus; Rechts: Neuer, erweiterter Agenten-Lebenszyklus

werden. Dieser Schritt wird mittels der bereitgestellten Transformationslogik durch Systemexperten automatisiert und stellt somit keinen weiteren Aufwand für die Systemadministratoren dar.

- *Schritt 3 - Bereitstellung & Update:* Nach der Transformation werden die lösungsspezifischen Agenten auf automatisierte Art und Weise auf VMs mittels standardkonformen Technologien bereitgestellt, um Skalierbarkeitsproblemen in großen Cloud-Umgebungen entgegenzutreten. Bereitgestellte Agenten werden bei Änderungen des generischen Agenten entsprechend aktualisiert.
- *Schritt 4 - Anpassung:* Bereits modellierte generische Agenten werden in einem Repository verwaltet und können angepasst werden. Der Zyklus wird wiederholt und in *Schritt 3* werden bereits auf den VMs bereitgestellte Agenten mit der neuen Konfiguration aktualisiert.

Im Folgenden werden die einzelnen Schritte detailliert erläutert.

4.3.1 Schritt 1 - Modellierung

Üblicherweise sind Agenten mit einem bestimmten Monitoring-System gekoppelt und können nicht zusammen mit anderen Monitoring-Systemen

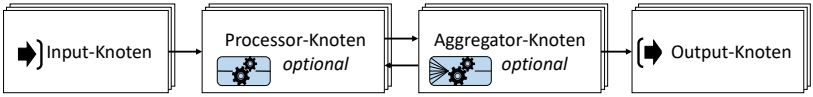

Abbildung 4.4: Komponenten der Agenten-Pipeline

genutzt werden. Hierbei gibt es Ausnahmen wie z. B. den Nagios NCPA Agenten [Nag21b], der aufgrund der großen Verbreitung von Nagios auch von weiteren Monitoring-Systemen unterstützt wird. Jedoch existiert kein Agent, der von allen Monitoring-Systemen unterstützt wird. Um dies zu ermöglichen, werden generische Agententemplates vorgestellt, um eine Abstraktionsstufe für die Modellierung von Agenten zu ermöglichen.

Die Grundlage bei der Modellierung von generischen Agenten ist die *Agenten-Pipeline* (s. Abbildung 4.4), die das Verhalten bezüglich Erfassung, Verarbeitung, und Aggregation von Monitoring-Daten, sowie die darauffolgende Weiterleitung an eine Datensenke beschreibt. Das daraus resultierende, generische Agententemplate ist ein gerichteter, azyklischer Graph, der aus den vier verschiedenen Typen von Knoten *Input, Processor, Aggregator* und *Output* besteht. Die Pipeline und Knotentypen sind angelehnt an die plugin-basierte Architektur von Agenten wie beispielsweise *Telegraf* [Inf21b] (TICK-Stack) oder *Beats* [Ela21b] (ELK-Stack) und ermöglicht es, Agenten aufgrund dieser Modularität flexibel und erweiterbar zu modellieren. Jede Instanz einer dieser vier Knotentypen ist definiert durch eine eindeutige ID, einen Namen, der dem Modellierer angezeigt wird, eine Konfiguration und seinem Typen. Zusätzlich besitzt jede Instanz jedes Knotentypens außer Output eine Liste von Nachfolgerknoten. Damit wird definiert, wie die Monitoring-Daten innerhalb des Agenten die Knoten durchlaufen. Mittels dieser Knotentypen können Systemadministratoren generische Agenten modellieren. Ein Beispiel hierfür wird in Abbildung 4.5 gezeigt, auf dessen Basis im Folgenden jeder der Knotentypen detailliert beschrieben wird.

4.3.1.1 Input-Knoten

Der Input-Knoten stellt den Einstieg in die Pipeline dar, daher besitzt jeder generische Agent mindestens einen Input-Knoten. Darin wird definiert, welche Metriken überwacht werden (z. B. *CPU-Auslastung* oder *Speicherauslastung*). Für jede dieser Metriken wird ein separater Input-Knoten erstellt. Dadurch ist es möglich, die Metriken auf unterschiedliche Weise zu überwachen. Beispielsweise lässt sich für jeden Input-Knoten eine individuelle Abtastfrequenz definieren, welche angibt, wie oft der Agent eine Messung vornimmt (z. B. alle 10 s für die CPU-Auslastung und alle 5 s für die Speicherauslastung). Wichtigere Metriken können somit häufiger überwacht werden. Analog können unwichtigere Metriken seltener überwacht werden, wodurch kostbare Netzwerkbandbreite gespart wird. Die gesammelten Monitoring-Daten werden daraufhin an die Nachfolgerknoten entsprechend der definierten Liste von Nachfolgerknoten weitergeleitet. Zulässige Nachfolgerknoten sind Processor- , Aggregator-, und Output-Knoten.

4.3.1.2 Processor-Knoten

Processor-Knoten sind optionale Knoten, die auf Basis der erhaltenen Monitoring-Daten eine *map*-Operation ausführen, d. h., die Operation wird

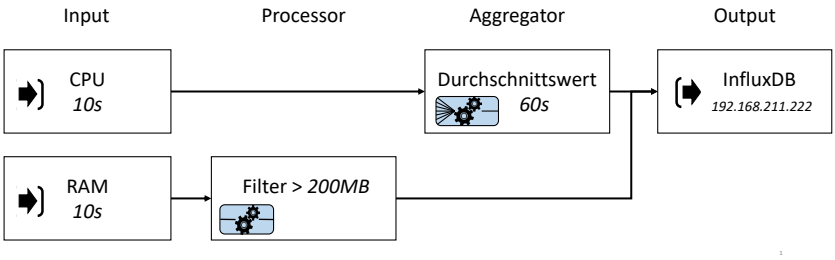

Abbildung 4.5: Beispielhafte Instanz einer Agenten-Pipeline

auf jedem einzelnen Messwert durchgeführt, der durch diesen Processor-Knoten geleitet wird. Die Operationen lassen sich in das *Filtern* und *Modifizieren* von Monitoring-Daten klassifizieren:

- Filtern: Mittels einer Filterfunktion ist es möglich, alle Messwerte herauszufiltern, die einer bestimmten Anforderung nicht genügen. Beispielsweise kann überprüft werden, ob ein Messwert (un)gleich oder über/unter einem vordefinierten Wert ist, oder ob der Messwert innerhalb oder außerhalb eines vordefinierten oberen und unteren Grenzwerts liegt (z. B. ob der Messwert für die Speicherauslastung mehr als 200 MB beträgt). Auf diese Weise lassen sich Messwerte filtern, die keine Relevanz haben und durch deren Filterung den Netzwerkverkehr verringern.
- Modifizieren: Die Modifikation wird genutzt, um eine Transformation der bereits gesammelten Monitoring-Daten vorzunehmen. Beispielsweise können überwachte Temperaturwerte von der Temperaturskala Fahrenheit in Celsius übertragen werden oder Zeitstempel in UNIX-Format in ein von Menschen lesbares Format transformiert werden. Es ist möglich, trotz Modifikation die originalen Monitoring-Daten weiterhin zu behalten und ebenso weiterzuleiten oder diese unwiderruflich zu löschen.

Ebenfalls können beide Operationen in beliebiger Reihenfolge kombiniert werden, z. B. kann zuerst eine Transformation von Messwerten in Fahrenheit auf Celsius stattfinden und anschließend alle Werte unter 40°C herausgefiltert werden. Es können keine oder mehrere Processor-Knoten definiert werden. Alle Messwerte werden daraufhin an die Nachfolgerknoten entsprechend der definierten Liste weitergeleitet. Zulässige Nachfolgerknoten sind Processor-, Aggregator-, und Output-Knoten.

4.3.1.3 Aggregator-Knoten

Aggregator-Knoten ähneln Processor-Knoten in der Hinsicht, dass Operationen auf den gesammelten Monitoring-Daten ausgeführt werden und sind

ebenfalls optional. Im Gegensatz zu Processor-Knoten werden die Operationen in Aggregator-Knoten allerdings auf einer Menge von Messwerten ausgeführt. Die Größe dieser Menge ist abhängig von dem Aggregationszeitraum, der im Aggregator-Knoten definiert wird, der Abtastfrequenz der Input-Knoten, d. h., wie oft der Agent neue Messwerte sammelt, und der Menge an Daten, die möglicherweise in vorherigen Processor-Knoten herausgefiltert worden sind. Verschiedene Arten von Aggregationen werden unterstützt, beispielsweise die Berechnung des Durchschnitts, Medians, Maximum oder Minimum für die Menge an Monitoring-Daten des jeweiligen Aggregationszeitraums (z. B. die Berechnung des Durchschnitts der CPU-Auslastung über einen Aggregationszeitraum von 60 s). Analog zu Processor-Knoten ist es möglich, die originalen Monitoring-Daten beizubehalten oder sie unwiderruflich zu löschen. In der Regel ist die Löschung der originalen Monitoring-Daten ratsam, da es hierbei zu einer enormen Senkung des Netzwerkverkehrs kommt. Allerdings erhalten Systemadministratoren hierdurch ein weniger akkurates Bild über den Zustand der überwachten Metrik. Es können keine oder mehrere Aggregator-Knoten definiert werden. Alle Messwerte werden daraufhin an die Nachfolgerknoten entsprechend der definierten Liste von Nachfolgerknoten weitergeleitet. Zulässige Nachfolgerknoten sind Processor-, Aggregator-, und Output-Knoten.

4.3.1.4 Output-Knoten

Output-Knoten stellen das Ende der Pipeline dar und sind dafür zuständig, die Monitoring-Daten an entsprechende Datensenken (z. B. die Datenbank *InfluxDB*), weiterzuleiten. Ebenso können die Daten an andere Datensenken wie Message-Queues weitergeleitet oder in eine lokale Datei geschrieben werden. Es muss mindestens ein Output-Knoten definiert werden. Mittels mehrerer Output-Knoten ist es möglich, Monitoring-Daten an verschiedene Datensenken weiterzuleiten. Beispielsweise könnten die aggregierten Messwerte der CPU-Auslastung an eine Datenbank und die gefilterten Messwerte der Speicherauslastung an eine andere Datenbank weitergeleitet werden. Ebenso ist es möglich, dass Monitoring-Daten an mehrere Datensenken

gleichzeitig weitergeleitet werden. Die Konfiguration der Output-Knoten enthält die URI der Datensenke, z. B. die IP des Monitoring-Servers und die Portnummer der Datenbank sowie weitere Parameter wie die Auswahl des Transportprotokolls TCP oder UDP.

Die vollständigen Informationen zu den verschiedenen Knotentypen sind in einem Schema definiert, das auf GitHub verfügbar ist [Mor21].

4.3.2 Schritt 2 - Transformation

Das Ergebnis der Modellierung im ersten Schritt ist ein generischer Agent. In dieser Form ist es nicht ausführbar, bspw. in Form eines JSON- oder XML-Dokuments. Daher wird zunächst eine Transformation in einen ausführbaren, lösungsspezifischen Agenten benötigt. Für diese Aufgabe wird die Komponente *Agent Mapper* eingeführt, welche die benötigte Transformationslogik beinhaltet. Hierfür müssen die Systemexperten, welche die technischen Details der lösungsspezifischen Agenten kennen, zuerst die Transformationslogik für jedes Monitoring-System bereitstellen, bevor eine Transformation in jenes unterstützt werden kann. Daher wird die Komplexität und Varianz der verschiedenen Monitoring-Systeme nicht verringert, sondern vielmehr von den Systemadministratoren auf einige wenige Systemexperten verlagert, welche die Transformationslogik für bestimmte Agenten bereitstellen. Durch die Bereitstellung dieser Transformationslogik auf einer öffentlichen Plattform können somit alle Unternehmen von der Arbeit eines einzigen Systemexperten profitieren.

Sobald die Transformationslogik im Agent Mapper vorliegt, kann eine Transformation eines generischen Agenten in einen lösungspezifischen Agenten des gewünschten Monitoring-Systems stattfinden. Der Transformationsschritt ist abhängig davon, ob dieser lösungsspezifische Agent alle Funktionalitäten unterstützt, die im generischen Agenten modelliert worden sind. Wie in Tabelle 4.1 zu sehen ist, werden nicht alle Funktionen gleichermaßen von allen Agenten unterstützt. Die grundlegende Funktionalität, nämlich das Erfassen von Messwerten und deren Weiterleiten an eine Datensenke, ist bei allen Agenten vorhanden. Lösungsspezifische Agenten wie beispielsweise

Telegraf unterstützten darüber hinaus auch das Filtern und Aggregieren von Daten. Daher erfordert eine Transformation von generischen Agenten zu Telegraf keine weiteren Maßnahmen. Anforderung *A*2 beinhaltet jedoch, dass eine Transformation auch hin zu lösungsspezifischen Agenten unterstützt werden soll, deren Funktionalität nicht jene der generischen Agententemplates abdeckt. In diesem Fall werden zwei Maßnahmen getroffen:

- Warnung: Wenn die Systemadministratoren eine Transformation eines generischen Agenten in einen lösungsspezifischen Agenten versuchen, der nicht die benötigten Funktionalitäten für eine direkte Transformation enthält, z. B. die Aggregation von Monitoring-Daten, wird zunächst eine Warnung ausgegeben. Diese Warnung gibt an, dass eine weitere Maßnahme *Outsourcing* getroffen werden muss, um eine erfolgreiche Transformation zu gewährleisten. Die Systemadministratoren haben daraufhin die Möglichkeit, einen anderen lösungsspezifischen Agenten zu wählen oder den generischen Agenten dahingehend anzupassen, damit eine direkte Transformation möglich ist, oder aber die Maßnahme *Outsourcing* zu wählen.
- Outsourcing: Wie in Kapitel 1 beschrieben, hängt die Auswahl eines geeigneten Monitoring-Systems nicht unbedingt von den jeweiligen Agenten ab, sondern kann andere Gründe wie z. B. die Unterstützung von Machine-Learning-Algorithmen haben. Entscheiden sich die Systemadministratoren für das Outsourcing, wird eine Transformation trotz ungenügender Funktionalität des lösungsspezifischen Agenten ermöglicht. Hierfür wird eine Complex-Event-Processing (CEP)-Engine verwendet. Auf dieser CEP-Engine werden die Funktionalitäten ausgeführt, die von den lösungsspezifischen Agenten nicht unterstützt werden. In Abbildung 4.6 ist das bereits oben verwendete Beispiel zu sehen, allerdings werden nun die Filteroperation und die Aggregation auf die CEP-Engine ausgelagert. Anstatt diese Funktionen selbst auszuführen, leitet der Agent alle relevanten Monitoring-Daten über den Output-Knoten *CEP-Engine* weiter an die eigentliche CEP-Engine, wo die entsprechenden Operationen ausgeführt werden. Die Ergebnis-

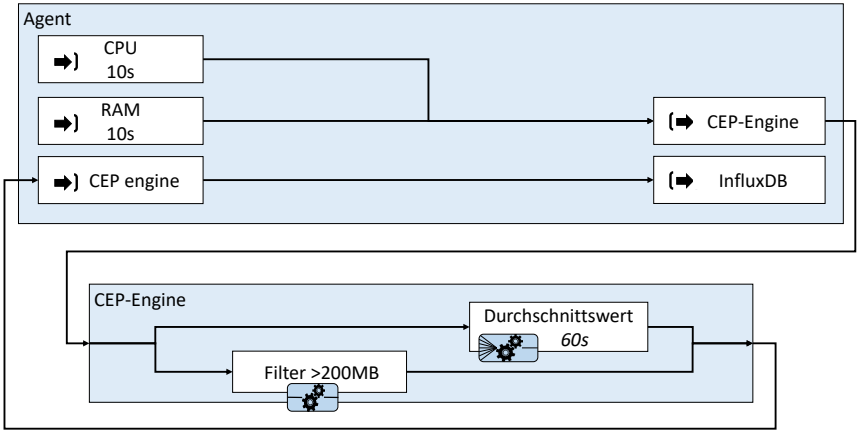

Abbildung 4.6: Auslagerung der Funktionen eines Agenten auf eine CEP-Engine

se werden wiederum zurück an den Agenten über den Input-Knoten *CEP-Engine* geleitet, der diese anschließend an die Datenbank *InfluxDB* leitet. Die benötigten Anpassungen, d. h., der neue Input- und Output-Knoten *CEP-Engine* und die Änderungen bezüglich der Weiterleitung von Monitoring-Daten innerhalb der Pipeline finden ausschließlich während der Transformation statt. Das bedeutet, dass der ursprünglich modellierte generische Agent beibehalten wird und jederzeit in seiner Grundform auch in andere lösungsspezifische Agenten transformiert werden kann. Mittels Outsourcing von Funktionalitäten wird somit die Wirksamkeit von generischen Agenten erheblich erhöht, da nun eine Transformation zu einer größeren Anzahl lösungsspezifischer Agenten möglich ist.

4.3.3 Schritt 3 - Bereitstellung

Vor allem in sehr großen Cloud-Umgebungen mit einer großen Anzahl an VMs ist jede manuelle Aufgabe sehr zeit- und kostenintensiv. Daher ist eine automatische Bereitstellung von essentieller Bedeutung [TJT+18; WB14]. Es existieren bereits mehrere Anwendungen und Plattformen, z. B. *Chef* oder *Puppet*, die eine automatische Bereitstellung ermöglichen. Ein standardisiertes Bereitstellungs-Framework ist jedoch wünschenswert, da proprietäre Technologien mit der Zeit tendenziell verschwinden.

Ein etablierter Standard für ein solches Bereitstellen ist die Topology and Orchestration Specification for Cloud Applications (*TOSCA* [OAS13]) von der Organization for the Advancement of Structured Information Standards (*OASIS*). TOSCA ermöglicht einen zweistufigen Software-Bereitstellungsansatz. Zunächst wird ein Template für die Topologie erstellt, das die Anwendungs-, Plattform- und Infrastrukturkomponenten modelliert. Anschließend wird diese Topologie für das Bereitstellen der Komponenten in der entsprechenden Infrastruktur verwendet.

Eine Implementierung des TOSCA-Standards ist OpenTOSCA [Uni19]. OpenTOSCA stellt ein Ökosystem zur Verfügung, das aus dem Modellierungswerkzeug für TOSCA-Topologien *Winery* [Ecl20] und dem OpenTOSCA-Container, der die eigentliche, auf der Topologie basierende Software-Bereitstellung abwickelt [BBKL14], besteht. TOSCA und die Implementierung OpenTOSCA wurden für die automatisierte Bereitstellung von Agenten eingesetzt. Da es sich bei TOSCA um einen Standard handelt, bietet es einen hohen Grad an Anwendbarkeit und ist zukunftssicher.

Es wurde eine TOSCA-Topologie modelliert, die für die Bereitstellung des Monitoring-Systems und der Agenten genutzt werden kann. Wird eine neue VM gestartet, wird automatisch ein Agent installiert, wodurch nicht nur die Skalierbarkeit verbessert wird, sondern auch eine sofortige Überwachung der VM beginnt und somit zu einer lückenlosen Überwachung führt.

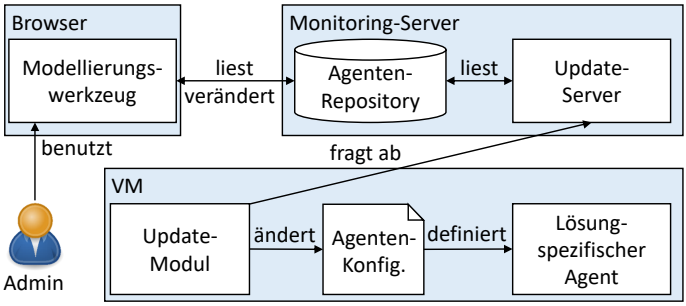

Abbildung 4.7: Architektur zur automatischen Anpassung bereitgestellter Agenten

4.3.4 Schritt 4 - Anpassung

Laut Gartner [PB18] ist eine hohe Flexibilität in Anbetracht einer sich stets verändernden IT-Architektur ein Schlüsselziel, wenn in ein neues Monitoring-System investiert werden soll. Die Modellierung von Agenten ist von den derzeitigen Geschäftsbedürfnissen und der gegenwärtigen Cloud-Umgebung beeinflusst. Beides davon kann sich jedoch mit der Zeit ändern, was wiederum eine Änderung der Agenten erfordert. In diesem Fall wäre es ein sehr hoher Aufwand, den gesamten Modellierungsprozess des generischen Agenten zu wiederholen oder jeden einzelnen lösungsspezifischen Agenten anzupassen. Stattdessen sollten Systemadministratoren die Möglichkeit haben, auf bereits modellierte Agenten zuzugreifen und diese nach Bedarf an die neuen Gegebenheiten anzupassen. Die durchgeführten Änderungen in den generischen Agenten werden anschließend wieder über den Transformationsschritt auf neue lösungsspezifische Agenten transformiert. Alle Änderungen werden automatisch auf bereits bereitgestellte Agenten auf den VMs propagiert, die jenen veränderten Agenten entstammen. Um diesen Prozess zu ermöglichen, wurde ein Prototyp implementiert, dessen Funktionsweise in Abbildung 4.7 zu sehen ist. Systemadministratoren passen einen generischen Agenten über das webbasierte Modellierungswerkzeug an, das mit dem *Update-Server* auf dem *Monitoring-Server* verbunden ist. Der Update-

Server ist für die Verwaltung der generischen Agenten verantwortlich, die im *Agenten-Repository* gespeichert sind. Auf den VMs fordert ein *Update-Modul* periodisch die Konfiguration (den transformierten generischen Agenten) für den Agenten an und löst Unterschiede zwischen der abgerufenen und der aktuell verwendeten Konfiguration auf. Bei geänderten Konfigurationen oder Fehlern startet das Update-Modul den Agent automatisch neu und wendet die neuen Konfigurationen an. Auf diese Weise erhalten alle Agenten immer die aktuellsten Konfigurationen, was vor allem bei einer großen Anzahl verschiedener Agenten und jeweiliger Konfigurationen einen skalierbaren Prozess darstellt, da kein weiterer manueller Aufwand für die Systemadministratoren entsteht.

4.4 Implementierung der Konzepte und Diskussion

Im Folgenden wird die prototypische Implementierung vorgestellt und anschließend diskutiert, ob die Anforderungen aus Abschnitt 4.2 erfüllt worden sind.

4.4.1 Prototypische Implementierung

Für die Modellierung generischer Agenten wurde auf Basis des oben beschriebenen Schemas ein grafisches, web-basiertes Modellierungswerkzeug prototypisch implementiert, welches auf GitHub[1] verfügbar ist. Dieses soll bei der Modellierung generischer Agenten unterstützen. In Abbildung 4.8 ist ein Screenshot des Modellierungswerkzeugs zu sehen. Vorgefertigte Knoten, z. B. Input-Knoten für das Überwachen der CPU- oder Speicherauslastung sind vorhanden und auf der linken Seite über das Dropdown-Menü *Input type* auswählbar (z. B. *MemoryInput*). Je nach Auswahl des *Input types* sind individuelle Einstellungen verfügbar, z. B. ob beim Monitoring der CPU-Auslastung jeder CPU-Kern separat überwacht werden soll. Rechts sind die grafischen Knoteninstanzen zu sehen. Mittels einfacher Mausklicks auf die

[1] https://github.com/mormulms/agent-centric-monitoring/blob/master/generic-agent/mona-template-editor

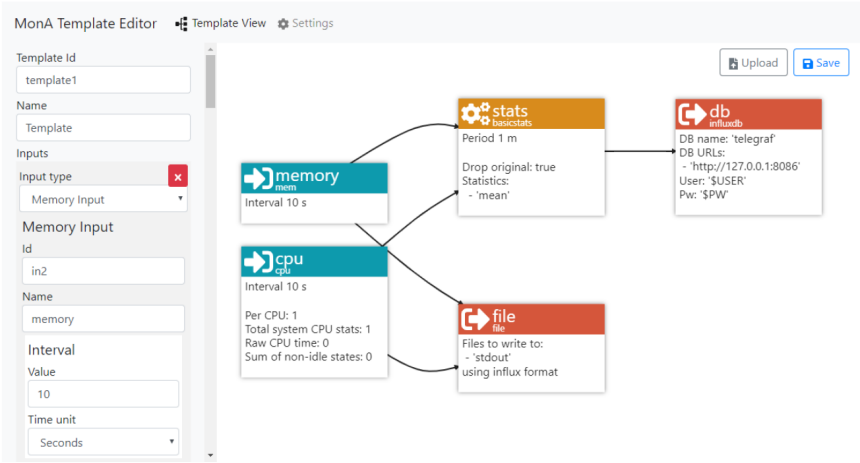

Abbildung 4.8: Web-basiertes Werkzeug zur Modellierung generischer Agenten

entsprechenden Knoten können diese miteinander verbunden werden, wodurch eine Instanz einer Agenten-Pipeline entsteht. Auf diese Weise wurde ein generischer Agent modelliert, das alle 10 s Messwerte der CPU- und Speicherauslastung erfasst, diese Messwerte anschließend im Aggregator-Knoten *stats* jeweils 1 m lang aggregieren, und letztlich über den Output-Knoten *db* an die Datenbank *InfluxDB* gesendet (s. Abbildung 4.8). Parallel dazu werden die Messwerte bereits vor der Aggregation zusätzlich in eine lokale Textdatei geschrieben.

Über den Button *Upload* können bereits modellierte Agenten aus dem Agenten-Repository in das Modellierungswerkzeug geladen werden. Über den Button *Save* können modellierte Agenten im Agenten-Repository gespeichert bzw. aktualisiert werden, falls ein bereits modellierter Agenten geladen und modifiziert wurde. Fehlerhafte Agenten, d. h., jene, die nicht konform mit dem oben beschriebenen Schema sind, können weder gespeichert noch aktualisiert werden.

Die Transformationslogik für den lösungsspezifischen Agenten Telegraf [Inf21b] des Monitoring-Systems [Inf21c] wurde implementiert, sodass jeder modellierte, generische Agent in einen Telegraf-Agenten transformiert werden kann. Ebenso wurde die automatische Bereitstellung von Telegraf-Agenten mittels OpenTOSCA implementiert.

```
1
  [agent]
3 interval = "10s"
  round_interval = true
5 [[inputs.cpu]]
  interval = "10s"
7   [inputs.cpu.tags]
    in1 = "1"
9
  [[aggregators.basicstats]]
11 period = "1m"
  drop_original = true
13 stats = [ "mean" ]
    [aggregatos.basicstats.tagpass]
15  in1 = [ "*" ]
    [aggregators.basicstats.tags]
17  agg1 = "1"

19 [[outputs.influxdb]]
  database = "telegraf"
21 urls = ["http://192.168.221:8086"]
    [outputs.influxdb.tagpass]
23  agg1 = ["*"]
```

Listing 4.1: Konfiguration des Agenten Telegraf

Zu Demonstrationszwecken ist in Listing 4.1 ein Ausschnitt der zugehörigen Konfiguration des in Abbildung 4.8 dargestellten Agenten im lösungsspezifischen Agenten Telegraf zu sehen. Im Codeausschnitt werden nur der Input-Knoten *cpu*, der Aggregator-Knoten *stats*, und der Output-Knoten *db* angezeigt. Wie zu sehen ist, stellt der Codeausschnitt Ansprüche an das

Verständnis in Bezug auf Syntax und Semantik der Programmiersprache sowie der Eigenheiten in der Funktionsweise von Telegraf. Im Gegensatz dazu ist die Modellierung von generischen Agenten mithilfe des grafischen Modellierungswerkzeugs ohne tiefere IT-Kenntnisse möglich.

Die erstmalige Bereitstellung lösungsspezifischer Agenten auf VMs wurde mit OpenTOSCA[1] realisiert. Für Updates bereits bereitgestellter Agenten bei der Modifikationen generischer Agenten wurde CloudConductor[2] verwendet.

4.4.2 Diskussion

Im Folgenden wird diskutiert, wie die jeweiligen Anforderungen $A1 - A3$ (s. Abschnitt 4.2) durch den Beitrag dieses Kapitels erfüllt worden sind.

Mittels der Agenten-Pipeline ist es Systemadministratoren möglich, ohne technische Kenntnisse generische Agenten und somit das Verhalten eines Agenten zu modellieren. Input-Knoten werden für das Erfassen der Monitoring-Daten verwendet, Processor- und Aggregator-Knoten für deren Bearbeitung, und Output-Knoten, um die Monitoring-Daten an eine Datensenke weiterzuleiten. Zur Definition der Agenten-Pipeline und zur Gewährleistung konformer Agenten wurde ein Schema definiert. Somit wurde Anforderung $A1$ erfüllt.

Die Auslagerung von Funktionalitäten auf externe Auswertungsmodule, z. B. CEP-Engines, ermöglicht eine Transformation generischer Agenten auf lösungsspezifische Agenten selbst dann, wenn die lösungsspezifischen Agenten nicht die erforderliche Funktionalität besitzen. Wie in Tabelle 4.1 betrifft dies sieben der dreizehn ausgewählten Monitoring-Systeme, die im Rahmen dieses Kapitels betrachtet wurden. Durch die Auslagerung der Funktionalitäten kann der Ansatz generischer Agententemplates daher auf einer noch größeren Anzahl von Monitoring-Systemen bzw. deren Agenten eingesetzt werden. Anforderung $A2$ ist daher erfüllt.

Ein erweiterter Lebenszyklus für Agenten unterstützt die automatische Bereitstellung lösungsspezifischer Agenten mit der standardkonformen Tech-

[1] https://www.iaas.uni-stuttgart.de/forschung/projekte/opentosca/
[2] http://www.cloudconductor.net/

nologie TOSCA und nachträgliche Updates jener Agenten mit handelsüblicher Software. Für die Modellierung der generischen Agenten wurde ein web-basiertes Modellierungswerkzeug implementiert. Modellierte Agenten können in einem Agenten-Repository gespeichert und verwaltet werden und sind über das Modellierungswerkzeug abrufbar und bei Bedarf veränderbar. Somit ist auch Anforderung *A*3 erfüllt.

4.5 Zusammenfassung

In diesem Kapitel wurden generische Agenten vorgestellt, um eine Abstraktionsstufe bei der Modellierung von Agenten einzuführen und somit die Gefahr eines Vendor Lock-ins zu verringern. Technische Details wurden abstrahiert, sodass Systemadministratoren keine technischen Kenntnisse der jeweiligen lösungsspezifischen Agenten benötigen. Systemexperten liefern daraufhin die benötigte Transformationslogik, um generische Agenten auf lösungsspezifische Agenten zu transformieren.

Zunächst wurde eine Analyse aktueller Monitoring-Systeme vorgenommen, aus welcher Anforderungen an die Funktionalitäten generischer Agenten und dem Lebenszyklus von Agenten aufgestellt wurden. Basierend auf diesen Anforderungen wurde die Agenten-Pipeline aufgestellt, welche zur Modellierung der generischen Agenten verwendet wird. Zusätzlich wurde ein erweiterter Lebenszyklus für Agenten vorgestellt, der auf Basis eines web-basierten Modellierungswerkzeugs das Verwalten und Bereitstellen von Agenten unterstützt.

Des Weiteren wurde es ermöglicht, selbst dann eine Transformation auf lösungsspezifische Agenten zu ermöglichen, wenn diese die im generischen Agenten modellierten Funktionalitäten, wie z. B. die Aggregation von Monitoringdaten, nicht unterstützen. Hierfür findet ein Outsourcing jener Funktionalitäten vom Agenten auf eine CEP-Engine statt, welche diese berechnet und die Ergebnisse zurück an den Agenten leitet.

Der Beitrag des nächsten Kapitels baut auf generischen Agententemplates auf und beschreibt einen Ansatz zur verteilten Auswertung von Alerting-Regeln mithilfe von Agenten.

Kapitel 5

Verteilte Auswertung von Alerting-Regeln

Wie bereits in Kapitel 2 beschrieben, werden erfasste Monitoring-Daten im Allgemeinen auf einem zentralisierten Monitoring-Server gespeichert [ARM+15; HW18]. Das Monitoring von Cloud-Umgebungen, die aus etlichen VMs bestehen, und auf denen jeweils mehrere Anwendungen ausgeführt werden, führt demnach schnell zum Monitoring mehrerer Tausend Metriken. Wird dies kontinuierlich und hochfrequent, z. B. jede Sekunde, durchgeführt, kann Monitoring als Big-Data-Problem betrachtet werden [Har17]. Vor allem das Datenvolumen der Monitoring-Daten steigt schnell auf eine Größe, welche die Bandbreite eines Netzwerks und Speichermedien stark belastet [ABdP13; HW18]. Eine gängige Lösung dieses Problems ist die Verwendung von Agenten, welche die Monitoring-Daten aggregieren, bevor sie diese an den Monitoring-Server weiterleiten [ABdP13]. Auf diese Weise kann die Datenmenge stark reduziert werden, allerdings führt eine Aggregation der Daten zwangsläufig zu einem Verlust der Genauigkeit [TPD15]. Aceto et al. [ABdP13] definieren ein Monitoring-System dann als *genau*,

wenn die Messwerte, die es bereitstellt, so nah wie möglich an dem realen Wert liegen. Demzufolge ist ein Monitoring-System auch genau, wenn es nur einen Messwert am Tag liefert, dieser aber dem realen Wert entspricht. Daher wird im Rahmen dieser Dissertation die Definition der Genauigkeit eines Monitoring-Systems dahingehend erweitert, dass sowohl der zeitliche Aspekt beachtet wird. Demzufolge ist ein Monitoring-System dann *genau*, wenn (1) die Messwerte so nah wie möglich an dem realen Wert liegen und (2) die Messwerte in einer für den Anwendungsfall geeigneten Abtastfrequenz erfasst werden. Infolgedessen ist ein genaues, feingranulares Monitoring nicht möglich, wenn eine Aggregation der Monitoring-Daten stattfindet. Vor allem die Qualität des Alerting wird durch eine schlechte Genauigkeit beeinträchtigt, wodurch die Zahl von falsch-negativen und falsch-positiven Alerts erhöht wird und es dadurch zu unnötigen Behebungsmaßnahmen oder sogar zum Übersehen schwerwiegender Probleme führt.

Eine Alternative zur Aggregation ist daher der Einsatz dezentralisierter Monitoring-Architekturen. Hierbei werden auf jeder VM voll funktionsfähige Monitoring-Systeme installiert, welche miteinander kommunizieren können. Die Monitoring-Daten können daraufhin lokal, d. h. direkt auf der VM, gesammelt und mit hoher Genauigkeit analysiert werden, ohne Netzwerkverkehr zu erzeugen. Allerdings führt eine Dezentralisierung zwangsläufig zu einer höheren Management-Komplexität, da kein SPoA mehr vorhanden ist und nun mehrere Systeme parallel verwaltet werden müssen [WB14]. Des Weiteren führt ein solcher Ansatz der Dezentralisierung zu einem nicht zu vernachlässigen Overhead in Bezug auf Speicher- und CPU-Auslastung auf den VMs, was wiederum zu höheren Kosten führen kann. Zusätzlich wird aufgrund der Verteilung der Monitoring-Daten eine holistische Sicht der Cloud-Umgebung erschwert.

Daher wird nun ein hybrider Ansatz verfolgt, der die Vorteile der zentralisierten mit denen der dezentralisierten Monitoring-Architekturen vereinen soll, um ein feingranulares Alerting zu ermöglichen, ohne jedoch die Nachteile der Aggregation von Monitoring-Daten oder die erhöhte Management-Komplexität dezentralisierter Monitoring-Architekturen zu übernehmen. Hierfür wird DEAR (**D**istributed **E**valuation of **A**lerting **R**ules; dt.: Verteilte

Auswertung von Alerting-Regeln) vorgestellt, welches ein Plugin für heutige Monitoring-Systeme darstellt. DEAR wird mit dem Alerting-Framework eines Monitoring-Systems verbunden und verteilt daraufhin die Alerting-Regeln an die entsprechenden VMs unter Berücksichtigung mehrerer Anforderungen wie z. B. Genauigkeit und Netzwerkverkehrvolumen. Um diese Anforderungen zu erfüllen, wird auf jeder VM ein Auswertungsmodul verwendet, welches für die Auswertung der Alerting-Regeln zuständig ist. Entsprechende Änderungen an den Agenten und dem Alerting-Framework werden von DEAR automatisiert vorgenommen und das Management der Alerting-Regeln verbleibt zentralisiert, um die Management-Komplexität nicht zu erhöhen.

Dieses Kapitel ist eine überarbeitete Version einer Publikation des Autors [MHSM20b].

Das restliche Kapitel ist wie folgt strukturiert: In Abschnitt 5.1 werden verwandte Arbeiten vorgestellt. Dazu zählen die verschiedenen Monitoring-Architekturen sowie hybride Ansätze, die sowohl Eigenschaften zentralisierter als auch dezentralisierter Monitoring-Architekturen vereinen. Anschließend werden in Abschnitt 5.2 die Anforderungen an DEAR anhand mehrerer Szenarien erläutert. In Abschnitt 5.3 wird DEAR mit seinen Komponenten und Funktionalitäten vorgestellt. Abschnitt 5.4 enthält die Evaluation von DEAR und Abschnitt 5.5 die Zusammenfassung dieses Kapitels.

5.1 Verwandte Arbeiten

Im Folgenden werden verwandte Arbeiten im Hinblick auf verschiedene Monitoring-Architekturen sowie hybride Ansätze betrachtet.

In **zentralisierten Monitoring-Architekturen** ist das Aggregieren von Monitoring-Daten auf dem Agenten eine gängige Methode, um den Netzwerkverkehr erheblich zu verringern [ABdP13]. Beispielsweise kann der Durchschnitt, Median, Maximum, oder Minimum einer Menge von Messwerten berechnet werden und an den Monitoring-Server weitergeleitet werden, anstatt alle Messwerte direkt weiterzuleiten. Weitere Methoden bei der Aggregation sind möglich [AMD+11; MDAM11]. Das Problem, das

bei der Aggregation von Monitoring-Daten entsteht, ist eine Verringerung der Genauigkeit des Monitoring [TJT+18; TPD15]. Ein weiterer Ansatz in zentralisierten Monitoring-Architekturen ist es, die Abtastfrequenz des Agenten zu verringern. Damit bezeichnet man die Häufigkeit, in welcher der Agent neue Messungen vornimmt und somit Monitoring-Daten erzeugt. Wo jedoch Abstände von 30 s zwischen zwei Messwerten oder länger in der Vergangenheit akzeptabel waren, ist heutzutage eine schnelle Reaktion auf unerwünschte Zustände zwingend notwendig, was unweigerlich zu einer hohen Abtastfrequenz führt [VT14; WB14].

Aus diesem Grund existieren parallel dazu N-tier- sowie vollständig **dezentralisierte Monitoring-Architekturen**. N-tier-Architekturen bestehen aus einer hierarchischen Baumstruktur der Tiefe n mit einem zentralen Monitoring-Server als Wurzel und den überwachten Ressourcen, z. B. VMs, als Blätter des Baums. Die dazwischen liegenden Knoten (für $n \geq 3$) stellen vollständige, autonome Monitoring-Systeme dar, die als *zentralisierte* Server für ein Subset der darunter im Baum liegenden überwachten Ressourcen gelten. Alle diese zwischengeschalteten Monitoring-Systeme senden die erhaltenen Daten zu ihrem Elternknoten weiter, bis diese am Wurzelknoten, d. h. dem eigentlichen zentralen Monitoring-Server, ankommen. Auf diesen Monitoring-Server greifen die Systemadministratoren zu, um die Monitoring-Daten vollständig zu erfassen. In vollständig dezentralisierten Monitoring-Architekturen existiert keine zentrale Komponente mehr, d. h., jeder Agent stellt selbst ein voll funktionsfähiges Monitoring-System dar, der mit anderen Agenten in der überwachten Umgebung kommuniziert. Eine Dezentralisierung ist allerdings immer mit einer erhöhten Management-Komplexität verbunden, die oftmals nicht akzeptabel ist, da kein SPoA mehr existiert und Systemadministratoren mehrere Monitoring-Systeme parallel verwalten müssen [WB14].

Hybride Monitoring-Architekturen versuchen, die Vorteile beider oben genannten Ansätze zu vereinen, indem Teile oder Funktionen nahe an die Quellen der Monitoring-Daten ausgelagert werden, während weiterhin eine zentrale Komponente beibehalten wird. Beispielsweise präsentieren Wange et al. das System *Monalytics* [KEW+10; WST+11], um Analysen nahe an den

Datenquellen durchzuführen. Hierfür wird eine Topologie der überwachten Umgebung erstellt. Physische Systeme werden in *Zonen* unterteilt und in jeder Zone erhalten Monitoring-Broker die gesammelten Monitoring-Daten der Agenten aus dieser Zone. Die Monitoring-Broker aggregieren und analysieren die Monitoring-Daten, um das Volumen der Monitoring-Daten und die Time-to-Insight (TTI: dt.: Zeit zur Erkenntnis) zu verringern. Es können jedoch nur simple Analysen durchgeführt werden, wohingegen kompliziertere Analysen explizit programmiert werden müssen. Weiterhin werden historische Daten nicht berücksichtigt, was dazu führt, dass die Möglichkeit für zukünftige Analyse verloren geht. DEAR ermöglicht die Analyse historischer Daten auf der Basis von aggregierten Monitoring-Daten. Des Weiteren ist es nicht nötig, eine Topologie der überwachten Umgebung zu erstellen.

Hauser et al. [HW18] stellen die Generierung von Ressourcenprofilen für überwachte Ressourcen vor. Diese Ressourcenprofile werden direkt auf der Datenquelle erstellt. Hierfür haben sie ein Tool entwickelt, welches auf dem Host installiert wird und daraufhin kontinuierlich Monitoring-Daten sammelt und daraus eine statistische Darstellung der Ressourcennutzung erstellt und regelmäßig aktualisiert. Hierfür werden beispielsweise Histogramme oder Markov-Chains verwendet und die rohen Monitoring-Daten werden verworfen. Das resultierende Modell für die Ressourcennutzung wird anschließend zur Pull-basierten Abfrage für Alerting-Systeme bereitgestellt, d. h., die Informationen werden vom zentralen Alerting-System bei Bedarf angefragt. Auf diese Weise werden der Netzwerkverkehr sowie der benötigte Speicherplatz auf dem Monitoring-Server reduziert. Dennoch werden Monitoring-Daten aggregiert, was mit der Verringerung der Genauigkeit der Daten einhergeht. Die Autoren behaupten, dass verringerte Genauigkeit keine Auswirkungen auf die Effektivität des Alerting hat, allerdings legen sie weder Beweise zur Untermauerung dieser Behauptung vor, noch zeigen sie, wie sehr die Genauigkeit im Vergleich zu einer direkten Datenübertragung der unveränderten Monitoring-Daten verringert wird. DEAR hingegen verringert die Genauigkeit der für das Alerting verwendeten Monitoring-Daten nicht und ermöglicht eine Analyse der unveränderten Monitoring-Daten.

Shao et al. [SWWM10] stellen einen Monitoring-Ansatz vor, bei dem die Monitoring-Daten gegen vordefinierte Regeln auf den Agenten geprüft werden. Beim Verstoß der Regeln werden Alerts an die Systemadministratoren gesendet. Ähnlich hierzu funktionieren Monitoring-Systeme wie z. B. Nagios [Nag21a] und Moogsoft [Moo21b], deren Agenten Daten direkt auf den Agenten analysieren und Warnungen oder Ähnliches erstellen können. Die Vorteile sind die Verringerung von Netzwerktraffic und sofortiger Einblick in die Monitoring-Daten aufgrund der geringeren Latenzzeit. Jedoch werden in all diesen Ansätzen die Alerting-Regeln direkt auf dem Agenten konfiguriert und verwaltet. In großen hochdynamischen Umgebungen führt diese Methode zu einem zu großen Management-Aufwand [WB14].

Unabhängig von der Domäne des Monitoring ist die verteilte Auswertung von Alerting-Regeln des Weiteren verwandt mit Operator-Placement (dt.: Operatorplatzierung) in Bezug auf CEP-Queries. Schultz-Møller et al. [SMP09] beschreiben einen Ansatz, um die Verteilung von CEP-Queries zu optimieren. CEP-Queries werden in Automaten transformiert und anschließend auf einen Cluster von Maschinen, z. B. VMs, verteilt. Cugola und Margara [CM12] stellen das *T-Rex CEP-System* vor, welches die Ausführung von CEP-Queries steuert. Die CEP-Queries sollen so nahe wie möglich an der Event-Quelle ausgeführt werden, um das benötigte Netzwerkverkehrvolumen zu verringern. Verwandte Arbeiten in diesem Bereich fokussieren sich rein auf die optimale Verteilung von CEP-Queries in Bezug auf individuelle Anforderungen, wie z. B. die Minimierung der Latenz oder des Netzwerkverkehrvolumens. DEAR hingegen konzentriert sich auf die konkrete Anwendung auf Monitoring-Systeme und Agenten. Die Konzepte der verwandten Arbeiten im Bereich Operator-Placement können daher in DEAR zur Verteilung der Alerting-Regeln angewandt werden und stehen nicht im Kontrast zu dem hier vorgeschlagenen Ansatz.

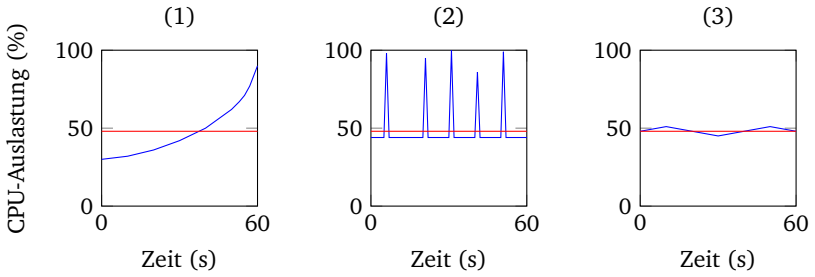

Abbildung 5.1: Beispielhafte Werte einer CPU-Auslastung einer VM (blau) und deren Durchschnittswert (rot)

5.2 Grundlagen & Anforderungen

In diesem Abschnitt wird betrachtet, inwiefern sich eine Aggregation von Monitoring-Daten auf die Genauigkeit des Monitoring bzw. Alerting auswirkt und welche negativen Folgen daraus resultieren können, basierend auf den Beispielen in Abbildung 5.1. In diesen drei Graphen werden beispielhafte Messwerte für eine beliebige überwachte Metrik, z. B. die CPU-Auslastung einer VM, über 60 s dargestellt. Der Agent erzeugt jede Sekunde einen neuen Messwert der CPU-Auslastung und berechnet nach dem Aggregationszeitraum von 60 s den Mittelwert (rotes Signal) dieser Messwerte. Im Folgenden werden zu jedem Beispiel das Resultat der Aggregation und deren Auswirkung auf das Monitoring der CPU-Auslastung analysiert:

Der erste Graph stellt eine schnell ansteigende Kurve dar und es ist denkbar, dass diese Kurve weiter steigen und zu einer CPU-Auslastung von 100 % führen könnte. Der berechnete Mittelwert allerdings stellt einen Wert unter 50 % dar und lässt vorerst auf keine zu hohe Auslastung schließen. Sollte die CPU-Auslastung weiter steigen und sich dauerhaft am Maximum befinden, wäre dies erst nach dem nächsten Aggregationszeitraum, d. h. nach weiteren 60 s, sichtbar, da dann der berechnete Mittelwert ebenso eine äußerst hohe Auslastung aufzeigen würde.

Der zweite Graph stellt eine in etwa gleichmäßige Auslastung der CPU mit einigen Auslastungsspitzen dar. Wie im ersten Graph würde der berechnete

Mittelwert bei einem Wert unter 50 % liegen. In diesem Fall würden die Auslastungsspitzen bei gleichbleibendem Verhalten weder in diesem noch in weiteren Aggregationszeiträumen identifiziert werden, obwohl diese jedoch möglicherweise auf ein zugrundeliegendes Problem der VM oder darauf installierten Anwendungen zurückzuführen sind.

Der letzte Graph zeigt eine nahezu gleichbleibende Auslastung der CPU an. Dies ist der einzige Graph, der durch den berechneten Mittelwert nahezu akkurat beschrieben werden kann. Auch hier zeigt die mittlere Auslastung einen Wert von knapp unter 50 % an.

Allgemein ist daher zu betrachten, dass trotz der unterschiedlichen Kurven in den jeweiligen Graphen der Mittelwert gleich ist und somit nicht auf die tatsächlichen Auslastungen geschlossen werden kann. Das Anscombe-Quartett [Ans73] veranschaulicht genau diese Problematik, dass verschiedene Mengen mit unterschiedlichen Datenpunkten die gleichen statistischen Merkmale wie z. B. Mittelwert und Varianz besitzen können. Analog zur Berechnung des Mittelwerts führen daher andere Formen der Aggregation wie z. B. die Berechnung des Medians, Maximums oder Minimums zu ähnlichen Problemen, d. h., feingranulare Informationen können verlorengehen.

Basierend auf den aus den Graphen resultierenden Beobachtungen werden folgende vier Anforderungen $A1 - A4$ und eine aus dem Cloud-Monitoring heraus allgemeine Anforderung $A5$ an DEAR gestellt:

A1: **Genauigkeit erhöhen**: Die Genauigkeit zählt zu den essentiellen Eigenschaften eines Monitoring-Systems. Ein Monitoring-System gilt als genau, wenn die Messwerte, die es bereitstellt, so nah wie möglich an dem realen Wert liegen [ABdP13], und eine für den Anwendungsfall geeignete Abtastfrequenz verwendet wird. Je feingranularer die Monitoring-Daten sind, d. h., je höher die Abtastfrequenz des Agenten ist, desto höher ist daher auch die Genauigkeit des Monitoring. Eine Aggregation der Monitoring-Daten wiederum senkt oftmals die Genauigkeit (s. Graph (1) und (2) in Abbildung 5.1). Beispielsweise können

die Auslastungsspitzen in Graph (2) nur dann erkannt werden, wenn die Abtastfrequenz des Agenten entsprechend hoch ist und gleichzeitig keine Aggregation der Monitoring-Daten vorgenommen wird.

A2: **Netzwerkverkehrvolumen von Monitoring-Daten verringern**: Aus *A*1 folgt aufgrund der hohen Abtastfrequenz der Agenten und der Vermeidung von Aggregation der Monitoring-Daten, dass nun eine große Menge an Daten entsteht. Vor allem in großen Cloud-Umgebungen, bei denen allein das Monitoring einer VM zu dutzenden Metriken führt, stellt dies eine große Belastung für die Netzwerkbandbreite und den benötigten Speicherplatz dar [Har17]. Daher muss das Volumen des Netzwerkverkehrs der Monitoring-Daten verringert werden, ohne die in Anforderung *A*1 geforderte Genauigkeit zu verringern, d. h. die gängige Methode des Aggregation von Monitoring-Daten ist nicht möglich.

A3: **Reaktionszeit verringern**: In der Domäne des Cloud Monitoring gilt als Maß für die Reaktionszeit die Time-to-insight (TTI). Die TTI beschreibt den Zeitraum zwischen dem Autreten eines Events und einer angemessenen Reaktion darauf und ist ausschlaggebend für eine schnelle Behebung von Problemen in IT-Umgebungen [ABdP13]. Im ersten Graph der Abbildung 5.1 kann beispielsweise das Ansteigen der CPU-Auslastung frühzeitig erkannt und entsprechend reagiert werden, bevor es zu einer Überlastung kommt.

A4: **Historisierung von Daten**: Monitoring-Daten müssen langfristig gespeichert und verwaltet werden können, um eine zukünftige Analyse der gesammelten Monitoring-Daten zu ermöglichen [BMC20]. In Graph (1) der Abbildung 5.1 kann beispielsweise das Muster der schnell ansteigenden CPU-Auslastung von neuronalen Netzen erlernt und entsprechend darauf automatisch reagiert werden. In Graph (3) hingegen kann z. B. das Muster des *normalen* Verlaufs der CPU-Auslastung erkannt werden, um daraufhin die Schwellwerte von Alerting-Regeln

entsprechend anzupassen. Für solche Methoden sind die Monitoring-Daten der letzten 60 Sekunden oder Minuten nicht ausreichend, daher ist eine langfristige Speicherung erforderlich.

A5: **Administration**: IT-Abteilungen sind oftmals unterbesetzt [Tec17]. Daher ist es notwendig, neue Ansätze möglichst automatisiert zu gestalten, um Systemadministratoren nicht zusätzlich zu belasten [BMC20]. Weiterhin muss, im Gegensatz zu dezentralisierten Monitoring-Ansätzen, ein SPoA vorhanden bleiben, d. h. Systemadministratoren greifen auf einen einzigen Server beziehungsweise ein einziges Monitoring-System zu, um das Monitoring und Management der überwachten IT-Umgebung zu kontrollieren.

5.3 DEAR

In diesem Abschnitt werden die Komponenten und Funktionalitäten von DEAR vorgestellt. DEAR stellt ein Plugin für die heutigen zentralisierten Monitoring-Systeme dar, um die Vorteile von dezentralisierten Lösungsansätzen zu nutzen, z. B. eine geringere TTI und weniger Netzwerkverkehr. Dennoch soll im Gegensatz zu dezentralisierten Lösungsansätzen ein SPoA vorhanden sein, um die Management-Komplexität nicht zu erhöhen. Hierfür fügt sich DEAR in die derzeitigen Monitoring-Architekturen ein. Abbildung 5.2 zeigt die erweiterte, zentralisierte Monitoring-Architektur mit eingebetteten Komponenten von DEAR, die im Beispiel dieser Abbildung zum Überwachen von vier VMs verwendet wird. Des Weiteren sind die üblichen Schritte (1) und (2) des agentenbasierten Monitoring zu sehen. Durch die Einführung von DEAR kommen die weiteren Schritte (3), (4) und (5) hinzu. Zusätzlich wird der ursprüngliche Schritt (1) durch Schritt (1') ersetzt.

Im Folgenden werden die einzelnen Schritte erläutert, um einen Überblick über die allgemeine Funktionsweise von DEAR zu verschaffen. Anschließend werden neue Komponenten sowie Veränderungen bereits bestehender Komponenten detailliert erläutert.

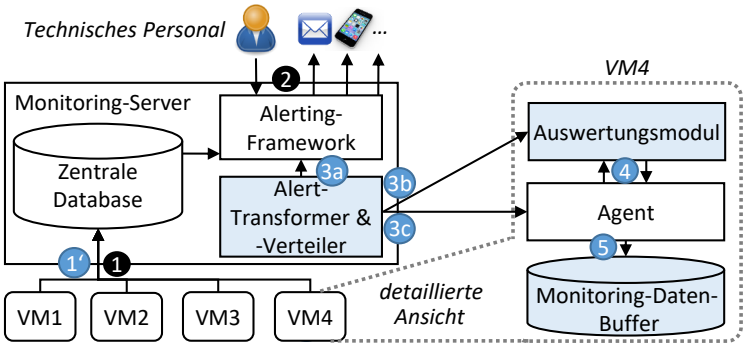

Abbildung 5.2: Integration der DEAR-Komponenten (blau) in eine bestehende Monitoring-Architektur; Prozessschritte von DEAR (neue Schritte in blau)

(1): Auf jeder VM ist ein Agent installiert wie in der detaillierten Sicht der *VM4* in der Abbildung 5.2 auf der rechten Seite zu sehen ist. Für jeden dieser Agenten wird wie im vorigen Kapitel definiert, was überwacht werden soll, wie z. B. CPU- oder Speicherauslastung. Zusätzlich werden (i) die Abtastfrequenz f_{Abtast} definiert, die bestimmt, wie oft Agenten neue Messungen durchführen, und (ii) der Aggregationszeitraum Z_{Aggr}, der den Zeitraum angibt, über den die Monitoring-Daten aggregiert werden sollen. Somit führt die Anwendung von Aggregation dazu, dass das Netzwerkverkehrvolumen von Monitoring-Daten um $(1 - \frac{f_{Abtast}}{Z_{Aggr}}) * 100\%$ reduziert wird. Jeder Agent sendet die Monitoring-Daten an die *zentrale Datenbank* auf dem *Monitoring-Server*.

(2): Basierend auf den gesammelten Monitoring-Daten können Systemadministratoren mithilfe des *Alerting-Framework* Alerting-Regeln definieren. Das Alerting-Framework evaluiert diese Alerting-Regeln kontinuierlich anhand aller Monitoring-Daten, die neu in die zentrale Datenbank geschrieben werden und für die entsprechenden Alerting-Regeln relevant sind. Wird eine der Alerting-Regeln verletzt, wird ein sogenannter Alert erzeugt und über Kommunikationskanäle wie z. B.

E-Mail oder Textnachricht, an die Systemadministratoren gesendet. Es wird ersichtlich, dass die Qualität des Alerting von dem Aggregationszeitraum Z_{Aggr} bezüglich der Genauigkeit und TTI stark beeinflusst wird, da die Aggregation zu grobgranularen Monitoring-Daten führt. Diese aggregierten Daten werden verspätet, d. h. erst nach Ablauf von Z_{Aggr}, an den Monitoring-Server weitergeleitet. Alerts könnten somit entweder erst deutlich später, gar nicht, oder fälschlicherweise erkannt werden.

(3): Die Komponente *Alert-Transformer & -Verteiler* greift auf das Alerting-Framework zu und transformiert die Alerting-Regeln in eine gängige Abfragesprache, sodass diese auf einem *Auswertungsmodul* platziert und ausgeführt werden können (s. Abbildung 5.2 (3a)). Anschließend werden die Alerting-Regeln auf dem Auswertungsmodul platziert (3b). Zuletzt werden die Konfigurationen der betroffenen Agenten entsprechend abgeändert, sodass diese die Monitoring-Daten direkt an das Auswertungsmodul weiterleiten (3c). Dies ermöglicht es, das Alerting lokal auf der entsprechenden VM und somit unabhängig von einer Aggregation der Monitoring-Daten stattfinden zu lassen.

(4): Von nun an sendet der Agent feingranulare Monitoring-Daten direkt an das Auswertungsmodul, auf welcher die Alerting-Regeln evaluiert werden, anstatt an den Monitoring-Server. Die Ergebnisse der Auswertung – die Alert-Daten – werden vom Auswertungsmodul zurück an den Agenten geschickt, der diese anschließend an den Monitoring-Server weiterleitet. Gleichzeitig führt der Agent eine Aggregation auf den ursprünglichen Monitoring-Daten aus und schickt die resultierenden, grobgranularen Monitoring-Daten an die zentrale Datenbank des Monitoring-Servers, um eine Historisierung dieser Daten zu ermöglichen. Auf diese Weise wurde die Qualität des Alertings unabhängig vom Aggregationszeitraum Z_{Aggr} gestaltet.

(5): Des Weiteren schickt der Agent feingranulare Daten an den *Monitoring-Daten-Buffer*, wo Monitoring-Daten kurzfristig gespeichert werden. Im Falle eines auftretenden Alerts sind somit die Monitoring-Daten, die

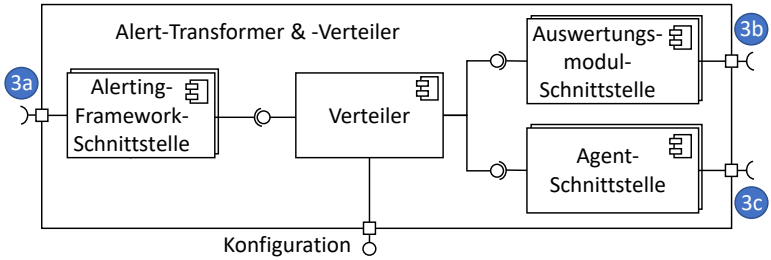

Abbildung 5.3: Aufbau des Alert-Transformers & -Verteilers; Schnittstellen 3a, 3b, 3c gemäß Abbildung 5.2

zu dem Alert geführt, in feinstmöglicher Granularität vorhanden und erlauben eine detaillierte Analyse, z. B., um eine Ursachenanalyse durchzuführen.

(1'): Das Resultat der vorigen Schritte 3-5 ist, dass die Kommunikation zwischen dem Agenten und dem Monitoring-Server sich verändert hat, daher wird der ursprüngliche Schritt 1 durch Schritt 1' ersetzt. Nur aggregierte Monitoring-Daten werden regelmäßig an den Monitoring-Server gesendet, um eine Historisierung zu ermöglichen und gleichzeitig die Netzwerkbandbreite nicht zu belasten. Relevante, feingranulare Monitoring-Daten werden nur im Falle eines Alerts aus dem Monitoring-Daten-Buffer in die zentrale Datenbank weitergeleitet, um auch diese Daten dauerhaft zu speichern.

Im Folgenden werden die einzelnen Komponenten und generelle Funktionsweise von DEAR detailliert erläutert.

5.3.1 Alert-Transformer & Verteiler

Der Alert-Transformer & -Verteiler ist die zentrale Komponente von DEAR und für (i) die Transformation von Alerting-Regeln, (ii) deren Verteilung an lokale Auswertungsmodule, und (iii) die Anpassung aller beteiligten Agenten verantwortlich. Eine modulare Architektur (s. Abbildung 5.3) ermöglicht es, weitere Alerting-Frameworks, Agenten, und Auswertungsmodule mittels

der Erzeugung neuer Schnittstellen anzubinden. Daher ist DEAR in Bezug auf verschiedene Monitoring-Systeme, deren Agenten, Alerting-Framework, und Auswertungsmodulen einsetzbar. Im Folgenden werden die jeweiligen Schnittstellen im Detail vorgestellt.

5.3.1.1 Alerting-Framework-Schnittstelle

Die Alerting-Framework-Schnittstelle wird verwendet, um Zugriff auf die im Alerting-Framework definierten Alerting-Regeln zu erhalten. Je nach dem, welches Alerting-Framework verwendet wird, sind die Definition und weitere Attribute unterschiedlich. Grundlegende Bestandteile jeder Alerting-Regel sind jedoch eine Regelbedingung und eine Aktion bei Verstoß dieser Regel. Weitere Attribute, z. B. der Name und Schweregrad einer Alerting-Regel, sind zudem hilfreich, um die Systemadministratoren dabei zu unterstützen, einen Alert schnell einordnen zu können. Da die Syntax und Aussagekraft von Alerting-Regeln sich je nach Alerting-Framework unterscheiden können, muss zunächst für jedes zu unterstützende Alerting-Framework eine neue Schnittstelle erstellt werden.

Bei der Transformation wird als erstes die Regelbedingung der Alerting-Regel in eine einheitliche Darstellung transformiert. Aufgrund dieser einheitlichen Darstellung wird die benötigte Anzahl von Transformationen bei n unterstützten Alerting-Frameworks und m unterstützten Auswertungsmodule von $n * m$ auf $n + m$ Transformationen verringert. Somit wird der Aufwand für das Hinzufügen eines weiteren Alerting-Framework oder eines anderen Auswertungsmoduls erheblich reduziert. Als einheitliche Darstellung werden Binary-Expression-Trees (BETs) [Pre98] mit einer Inorder-Traversierung verwendet. Weitere Darstellungsmöglichkeiten sind ebenso möglich. Die Knoten des BETs stellen boolesche Operatoren dar und die Blätter die Bedingungen der Alerting-Regel. Eine Regelbedingung mit mehreren Bedingungen und der äquivalente BET sind in Abbildung 5.4 dargestellt. Der BET wird als Input für die Auswertungsmodul-Schnittstelle verwendet, wo er in die entsprechende Abfragesprache des verwendeten Auswertungsmoduls transformiert wird, z. B. CEP-Queries bei Verwendung einer CEP-Engine. Die ursprüngli-

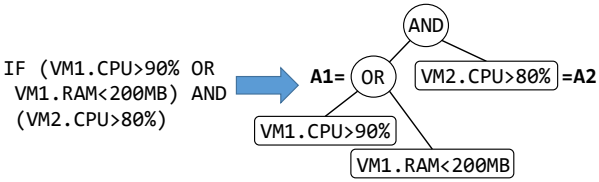

Abbildung 5.4: Transformation einer Alerting-Regel in einen Binary-Expression-Tree (BET)

che Regelbedingung im Alerting-Framework wird durch einen Platzhalter ersetzt, dessen Bezeichnung mit dem Bezeichner des berechneten Ergebnisses in dem Auswertungsmodul übereinstimmt. Die Regelbedingung in Abbildung 5.4 wird daher in $IF\ (VM1.A1)\ AND\ (VM2.A2)$ abgeändert und das Auswertungsmodul auf $VM1$ berechnet ($IF\ CPU > 90\%\ OR\ RAM < 200MB$) $THEN\ (A1 = true)$ und sendet $A1$ zurück an den Agenten, der das Ergebnis an den Monitoring-Server und somit an das Alerting-Framework weiterleitet. Analog dazu berechnet das Auswertungsmodul auf $VM2$ das Ergebnis $A2$.

Der Rest der Alerting-Regel bleibt unverändert, sodass das Management der Alerting-Regeln, z. B. die Definition der verwendeten Kommunikationskanäle (z. B. E-Mail) und deren Konfiguration (z. B. E-Mail-Adresse) gleichbleibend sind und ein SPoA vorhanden bleibt.

5.3.1.2 Verteiler

Die Komponente *Verteiler* ist verantwortlich für die Verteilung der Alerting-Regeln auf Basis der BETs. Mittels einer Inorder-Traversierung werden die längsten Teilbäume der BETs identifiziert, deren Blätter Bedingungen an Monitoring-Daten derselben VM darstellen. Diese Teilbäume werden von der Auswertungsmodul-Schnittstelle in die Abfragesprache des gewählten Auswertungsmoduls transformiert und an die VMs verteilt. Ab diesem Punkt muss der Agent die feingranularen Monitoring-Daten an das lokale Auswer-

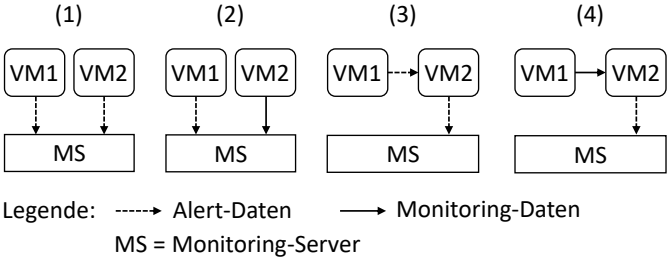

Abbildung 5.5: Strategien zur Verteilung von Alerting-Regeln

tungsmodul leiten und nicht mehr an den Monitoring-Server. Hierfür wird die Konfiguration des Agenten durch die *Agent-Schnittstelle* angepasst, um die Monitoring-Daten entsprechend umzuleiten.

Im vorigen Abschnitt wurde bereits eine Möglichkeit zur Verteilung der Alerting-Regeln vorgestellt. Abhängig von den Datenquellen, die eine Alerting-Regel betreffen, gibt es allerdings auch andere mögliche Strategien zur Verteilung wie in Abbildung 5.5 zu sehen ist. Um diese Strategien zu veranschaulichen, wird die Regelbedingung aus Abbildung 5.4 verwendet. Analog dazu können die Strategien auch auf mehr als zwei VMs angewendet werden.

(1): Alle VMs erhalten ihren entsprechenden BET-Teilbaum zugeordnet und die Agenten der VMs schicken die Resultate der Auswertung, d. h. die Alert-Daten in Form eines *TRUE* oder *FALSE*, zurück an den Monitoring-Server. Dies entspricht der Verteilung, wie sie im vorigen Abschnitt beschrieben wurde. Die Alerting-Regeln werden in der gleichen Frequenz ausgewertet, wie die Monitoring-Daten eintreffen. Das bedeutet, dass wenn jede Auswertung der Alerting-Regeln, egal ob diese zu *TRUE* oder *FALSE* ausgewertet werden, an den Monitoring-Server weitergeleitet wird, dass weiterhin ein Großteil der Datenmenge weiterbesteht und die Netzwerkbandbreite belastet. Daher werden die Alert-Daten nur dann weitergeleitet, wenn ein Zustandswechsel von *TRUE* zu *FALSE*

bzw. umgekehrt stattfindet. Dennoch können auch diese Zustandswechsel häufig auftreten, wenn sich die Werte der Monitoring-Daten um den Grenzwert der Regelbedingung herum bewegen.

(2): Wenn eine VM aus bestimmten Gründen, wie z. B. eine zu geringe Rechenleistung, keine lokale Auswertung der Teilbäume des BET zulässt, wird nur ein Teil der Teilbäume auf einzelnen VMs berechnet. Folglich senden die Agenten auf den restlichen VMs (im Beispiel ist dies VM2), die unveränderten Monitoring-Daten an den Monitoring-Server. Dies führt allerdings erneut zu einem erhöhten Aufkommen an Netzwerkverkehrvolumen.

(3): Im Beispiel aus Abbildung 5.4 kann die vollständige Alerting-Regel nur verletzt werden, wenn beide Teilbäume zu *TRUE* evaluiert werden. Durch häufige Zustandswechsel einzelner Teilbaum-Auswertungen können dennoch weiterhin große Datenmengen entstehen. Dies kann beispielsweise mit Strategie (3) verhindert werden. Hierbei sendet jede VMs, die Teil der Alerting-Regel ist, ihre Daten an eine VM (im Beispiel ist dies VM2), welche die finale Auswertung der Alerting-Regel durchführt und nur dann Alert-Daten an den Monitoring-Server leitet, wenn der gesamte BET einen Zustandswechsel hat. Strategie (3) ist (1) allerdings nur dann vorzuziehen, wenn die Kommunikation unter den VMs günstig ist, z. B., wenn sie Teil desselben Clusters bzw. derselben Cloud-Umgebung sind.

(4): Analog zu Strategie (2), wenn eine VM keine lokale Auswertung der Teilbäume des BET zulässt und zusätzlich die Kommunikation unter den VMs günstig ist, können die gesamten Monitoring-Daten an eine weitere VM weitergeleitet werden, die die Auswertung dieses Teilbaums übernimmt.

Die Wahl der Verteilungsstrategie ist abhängig von dem Anwendungsfall und der vorliegenden Topologie der Cloud-Umgebung, z. B., ob die VMs sich

im gleichen Cluster befinden, oder ob eine Multi-Cloud-Umgebung vorliegt. Durch geeignete Kombinationen der Strategien kann demnach eine optimale Verteilung bzgl. der Minimierung von Netzwerkverkehrvolumen stattfinden.

5.3.1.3 Auswertungsmodul-Schnittstelle

Für die Auswertung von Alerting-Regeln können verschiedene Auswertungsmodule verwendet werden, beispielsweise die CEP-Engine Esper [Esp21], die jeweils verschiedene oder leicht voneinander abgewandelte Abfragesprachen besitzen. Für jedes zu unterstützende Auswertungsmodul müssen demnach neue Auswertungsmodul-Schnittstellen geschrieben werden, um die BETs in die entsprechenden Abfragesprachen zu übersetzen.

5.3.1.4 Agent-Schnittstelle

Die Agent-Schnittstelle führt die benötigten Änderungen an der Konfiguration der Agenten durch. Hierbei wird in erster Linie das Routing der Monitoring-Daten an das Auswertungsmodul verändert, so dass aus dem ursprünglichen Agenten wie in Abbildung 5.6 der transformierte, DEAR-kompatible Agent aus Abbildung 5.7 entsteht. Die in diesem Kapitel betrachteten Agenten sind konform zu dem im vorigen Kapitel vorgestellten Schema für die Modellierung generischer Agententemplates, welches die Agenten-Pipeline, bestehend aus den Knoten *Input*, *Processor* (optional), *Aggregator* (optional), und *Output*, zugrunde liegt. Demnach ist dieser Ansatz entsprechend auf eine Vielzahl von Monitoring-Systemen und deren Agenten anwendbar.

In Abbildung 5.6, ist der ursprüngliche Agent zu sehen, der für das Überwachen der CPU einer VM verwendet wird. Der Input-Knoten wird zum Sammeln der CPU-Auslastungsmesswerte verwendet und führt diese Aufgabe mit der Abtastfrequenz f_{Abtast} durch, z. B. einmal pro Sekunde. Die gesammelten Messwerte werden an den Aggregator-Knoten weitergeleitet, welcher die Messwerte über den Aggregationszeitraum Z_{Aggr} hinweg aggregiert, z. B. über 60 Sekunden. Das Resultat der Aggregation wird an

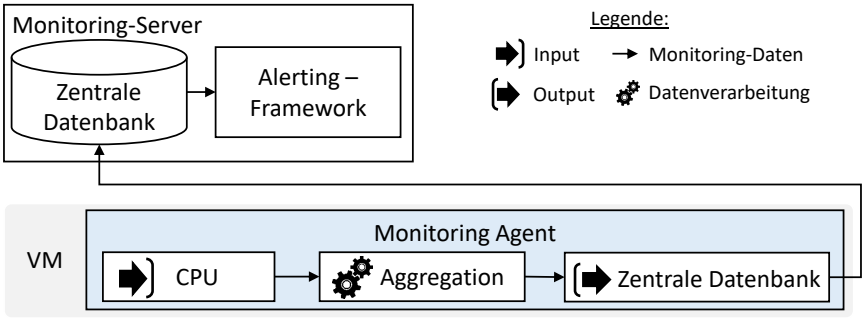

Abbildung 5.6: Anfängliche Konfiguration des Agenten

den Output-Knoten geleitet, welcher die Datensenke definiert, z. B. eine Datenbank. In diesem Beispiel wird durch Aggregieren der Messwerte der Netzwerkverkehr um $(1 - \frac{f_{Abtast}}{Z_{Aggr}}) * 100 = 98.\overline{3}\%$ verringert im Vergleich zur Übermittlung der unveränderten Monitoring-Daten. Die aggregierten, und daher weniger genauen, Monitoring-Daten werden anschließend in der zentralen Datenbank gespeichert und unter anderem durch das Alerting-Framework zur Auswertung von Alerting-Regeln verwendet.

Im Vergleich zum anfänglichen Agenten ist in Abbildung 5.7 der angepasste Agent nach der Transformation der Alerting-Regeln in BETs und deren Verteilung an die Auswertungsmodule zu sehen. Die Routing-Pfade *RP* innerhalb des Agenten sind mit $RP1 - RP4$ annotiert.

$RP3$ wird für die Auswertung der Regelbedingungen verwendet. Da das Auswertungsmodul auf der gleichen VM wie der Agent liegt, ist die Aggregation zur Reduzierung des Netzwerkverkehrvolumens nicht notwendig. Stattdessen werden die Messwerte der CPU-Auslastung unverändert an das Auswertungsmodul geleitet, auf welcher die Regelbedingungen ausgewertet werden. Die Auswertung gibt den Booleschen Wert $A1$ zurück, der angibt, ob der Grenzwert der Regelbedingung überschritten wurde oder nicht. Wie bei den Verteilungsstrategien in Abschnitt 5.3.1.2 beschrieben, werden nur dann Alert-Daten erzeugt, wenn ein Zustandswechsel von *TRUE* auf *FALSE* bzw. umgekehrt stattfindet.

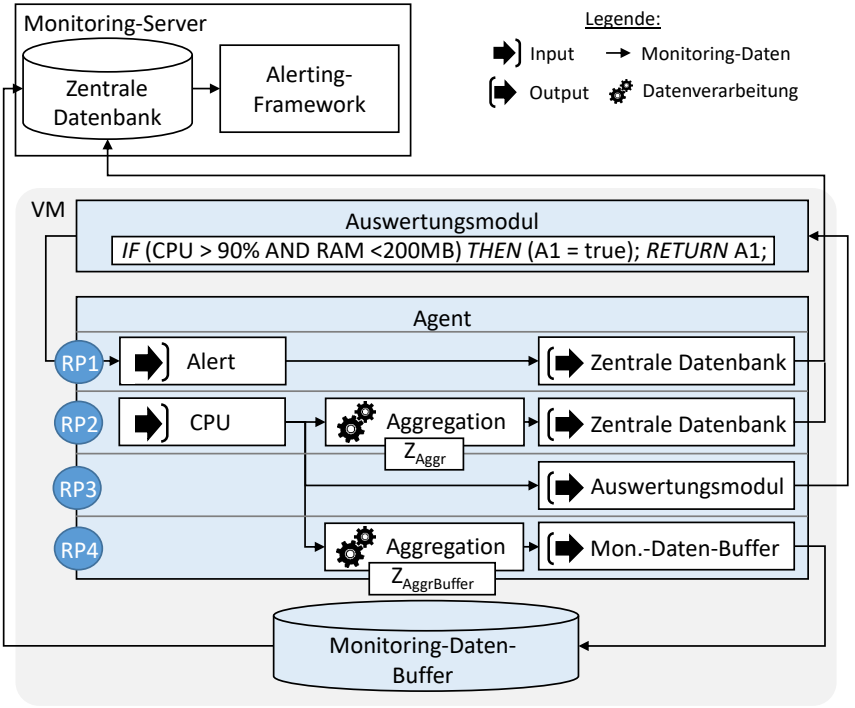

Abbildung 5.7: Umkonfigurierter Agent und DEAR-Komponenten

Über $RP1$ wird das Ergebnis $A1$ des Auswertungsmoduls an die zentrale Datenbank weitergeleitet. Dieses wird im Alerting-Framework verwendet, um Benachrichtigungen an Systemadministratoren anzustoßen.

Es wird ersichtlich, dass durch die lokale Auswertung der Regelbedingung eine Abkopplung zwischen der Abtastfrequenz f_{Abtast} und dem resultierenden Netzwerkverkehr stattgefunden hat. Das bedeutet, dass f_{Abtast} erhöht werden kann, um Anwendungsfälle zu unterstützen, die sehr feingranulare Monitoring-Daten für das Alerting benötigen, ohne die Netzwerkbandbreite mit deren Übertragung an den zentralen Monitoring-Server zu belasten.

*RP*2 stellt den ursprünglichen Routing-Pfad dar, wie er auch in Abbildung 5.6 zu sehen ist. Dieser wird weiterhin verwendet, um zu gewährleisten, dass Monitoring-Daten in der zentralen Datenbank historisiert werden, um diese für spätere Analysen verfügbar zu machen. Da nun jedoch die Auswertung der Regelbedingung lokal stattfindet, wurde der Aggregationszeitraum Z_{Aggr} von der Qualität des Alerting abgekoppelt. Daher kann Z_{Aggr} weiter erhöht werden, um mehr Netzwerkverkehr und Speicherplatz auf dem Monitoring-Server zu sparen.

*RP*4 wird verwendet, um im Falle eines Alerts dennoch eine feingranulare Analyse der Monitoring-Daten, die zu diesem Alert geführt haben, durchzuführen. Hierfür werden alle Monitoring-Daten für einen begrenzten Zeitraum lokal abgespeichert. Wird eine Regelbedingung verletzt, werden die relevanten, lokal gespeicherten Monitoring-Daten bei Bedarf an die zentrale Datenbank zur dauerhaften Historisierung für spätere Analysen geschickt, z. B. für eine Ursachenanalyse eines Alerts. Hierdurch erhöht sich allerdings wieder das Netzwerkverkehrvolumen. Daher können alternativ die Daten lokal auf der VM analysiert werden. Abhängig davon, wie lange der Zeitraum für die lokale Speicherung der Monitoring-Daten sein soll und wieviel Speicherplatz die VM besitzt, kann auch in *RP*4 Aggregation mit dem Aggregationszeitraum $Z_{AggrBuffer}$ eingesetzt werden, um das Datenvolumen der Monitoring-Daten zu verringern. Zu beachten ist, dass $Z_{AggrBuffer} < Z_{Aggr}$ zu wählen ist, da über *RP*2 ohnehin aggregierte Monitoring-Daten an die zentrale Datenbank geschickt werden und es daher keinen Vorteil ergeben würde, die Monitoring-Daten in gleicher oder sogar gröberer Granularität lokal zu speichern.

Für die Mindestfunktionalität sind die Routing-Pfade *RP*1 und *RP*3 erforderlich, um die lokale Auswertung von Regelbedingungen zu ermöglichen. *RP*2 und *RP*4 sind je nach Anforderungen bzw. Anwendungsfällen vorteilhaft, um zusätzliche Analysen auf grobgranularen bzw. feingranularen Monitoring-Daten durchzuführen.

5.3.2 Auswertungsmodul

Das Auswertungsmodul übernimmt die Aufgabe eines lokalen Alerting-Framework auf den VMs. Bei der Wahl einer geeigneten Engine ist auf eine geringe Latenz zur Unterstützung einer geringen TTI sowie auf hohen Durchsatz aufgrund der großen Datenmengen zu achten. Weiterhin sollte das Auswertungsmodul leichtgewichtig sein, d. h. sie sollte die Ressourcen der VM wie CPU und Speicher nur geringfügig beeinflussen. Gleichzeitig sollte die Abfragesprache ausdrucksstark genug sein, um die Alerting-Regeln adäquat darstellen zu können.

5.3.3 Monitoring-Daten-Buffer

Der Monitoring-Daten-Buffer agiert als kurzfristige Speichermöglichkeit für Monitoring-Daten. Der Zweck besteht darin, zusätzlich zur Auswertung der Regelbedingungen, eine feingranulare Analyse der Ursachen von Alerts zu ermöglichen, indem alle Monitoring-Daten für einen begrenzten Zeitraum gespeichert werden, z. B. eine Stunde lang. Alle Daten, die älter als dieser Zeitraum sind, werden unwiderruflich gelöscht. Die CPU-Auslastungsspitzen in Graph 2 der Abbildung 5.1 sind beispielsweise nach Aggregation der Monitoring-Daten nicht mehr sichtbar, obwohl diese Messwerte symptomatisch für einen bevorstehenden Fehler sein könnten oder den Verstoß einer Regelbedingung mit darauf folgendem Alert darstellen könnten. Durch die kurzfristige Speicherung im Monitoring-Daten-Buffer ist es möglich, detailliertere Analysen bezüglich der Ursachen von Alerts durchzuführen und Präventivmaßnahmen oder eine schnellere Reaktion auf einen drohenden Fehler zu ermöglichen.

5.4 Implementierung der Konzepte und Evaluation

In diesem Abschnitt werden die prototypische Implementierung und die Ergebnisse der Evaluation vorgestellt. Basierend auf verschiedenen Einstellungen werden die Vorteile von DEAR gegenüber dem agentenbasierten, zen-

tralisierten Ansatz quantitativ evaluiert. Anschließend werden die Resultate diskutiert und überprüft, ob die in Abschnitt 5.2 definierten Anforderungen $A1 - A5$ durch DEAR erfüllt worden sind.

5.4.1 Prototypische Implementierung

Zu Evaluationszwecken wurde DEAR prototypisch implementiert und ist auf GitHub verfügbar[1]. Die Komponenten wurden in Java entwickelt, um eine plattformunabhängige Lösung zu gestalten. Als Monitoring-System wurde TIG-Stack gewählt, welches aus den Komponenten Telegraf [Inf21b] (v1.9.3), InfluxDB [Inf21a] (v1.7.3), und Grafana [Gra21] (v6.0.0) besteht. Telegraf stellt den Agenten dar, auf den, wie in Kapitel 4 vorgestellt, Transformationen von generischen Agententemplates durchgeführt werden können. InfluxDB ist eine Zeitreihendatenbank und agiert gleichzeitig als Monitoring-Daten-Buffer und Grafana als Alerting-Framework. Entsprechend wurden eine Alerting-Framework-Schnittstelle für Grafana sowie eine Agent-Schnittstelle für Telegraf entwickelt. Als Auswertungsmodul zur lokalen Auswertung der Alerting-Regeln wurde die CEP-Engine Esper [Esp21] eingesetzt und eine entsprechende Auswertungsmodul-Schnittstelle implementiert. Im Prototyp entspricht die Verteilung der Alerting-Regeln der Strategie (1) aus Abbildung 5.5.

5.4.2 Evaluation

Im Folgenden werden der Versuchsaufbau und die Ergebnisse der Evaluation präsentiert. Basierend auf mehreren Einstellungen und Alerting-Regeln werden folgende Eigenschaften gemessen: (i) das Netzwerkverkehrvolumen, die verursachte Last (ii) von Telegraf und (iii) Esper bzgl. CPU- und Speicherauslastung (RAM), (iv) der Speicherplatz für Monitoring-Daten im Monitoring-Daten-Buffer, und (v) die TTI, jeweils für DEAR in Kombination mit dem TIG-Stack sowie für den TIG-Stack ohne DEAR.

[1] https://github.com/mormulms/agent-centric-monitoring

5.4.2.1 Versuchsaufbau

Für die Evaluation wurden zwei VMs der Private-Cloud-Plattform OpenStack [Ope21] verwendet. Die VMs besitzen die gleichen, folgenden Charakteristika: 2 VCPU, 4 GB RAM, 40 GB Disk, Ubuntu 16.04 Server. Die erste VM wird als Monitoring-Server verwendet, auf welcher InfluxDB, Grafana, und DEAR installiert ist, die zweite als zu überwachende VM, auf welcher Telegraf und Esper installiert sind.

Telegraf misst mit der Abtastfrequenz f_{Abtast} die CPU-Auslastung und in Grafana wurden die folgenden Regelbedingungen $R1 - R3$ erstellt:

```
R1: IF CPU.load > 90 %
R2: IF CPU.load > 90 % FOR 10s
R3: IF CPU.load > 90 % FOR 60s
```

Listing 5.1: Regelbedingungen

Es wird überprüft, ob die CPU-Auslastung der zu überwachenden VM über 90 % steigt bzw. in $R2$ und $R3$, ob dies über einen Zeitraum $Z_{Regelbedingung}$ von zehn bzw. sechzig Sekunden der Fall ist.

Des Weiteren werden für diese Evaluation drei verschiedene Einstellungen $E1 - E3$ verwendet, die sich in Bezug auf die Abtastfrequenz f_{Abtast}, den Aggregationszeitraum Z_{Aggr}, und den Aggregationszeitraum $Z_{AggrBuffer}$ unterscheiden. Die verschiedenen Einstellungen sind in Tabelle 5.1 aufgelistet. Für jede dieser Einstellungen wurden die Messungen über einen Zeitraum von einer Stunde erfasst. Die erste Einstellung gilt als Referenz, um die positiven als auch negativen Auswirkungen der Aggregation im Vergleich zu den anderen Einstellungen darzustellen. Die zweite Einstellung stellt ein übliches

Tabelle 5.1: Einstellungen für die Evaluation

Parameter	E1	E2	E3
Abtastfrequenz f_{Abtast}	1 / s	1 / s	100 / s
Aggregationszeitraum Z_{Aggr}	-	10s	60s
Aggregationszeitraum $Z_{AggrBuffer}$	-	10s	-

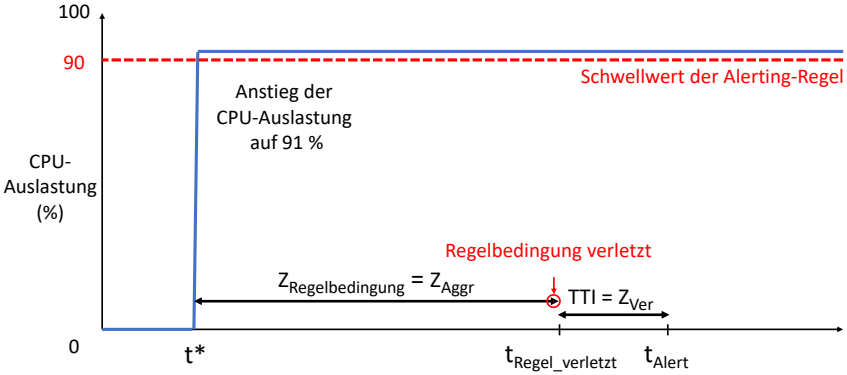

Abbildung 5.8: Berechnung der TTI

Vorgehen dar, indem mittels einer zehnsekündigen Aggregation ein Großteil des Netzwerkverkehrvolumens eingespart wird. Die letzte Einstellung entspricht einem Extrem-Szenario, bei dem eine sehr hohe Abtastfrequenz von 100 Messwerten pro Sekunde (d. h., ein Messwert alle 10 ms) benötigt wird, welches beispielsweise beim Monitoring einer Edge Cloud im industriellen Rahmen aufgrund hoher Latenzanforderungen eine wichtige Rolle spielen könnte.

Das Verfahren zur Messung der TTI wird in Abbildung 5.8 dargestellt. Die TTI wird demnach als der Zeitraum zwischen einer verletzten Regelbedingung und einem entsprechenden Alert betrachtet. Zu sehen ist der Verlauf der CPU-Auslastung (blau), welche zu Beginn bei 0 % liegt. Zu einem beliebigen Zeitpunkt t* erhöht sich die CPU-Auslastung auf 91 % und verbleibt auf diesem Wert. Der Messwert liegt über dem Schwellwert der Alerting-Regeln von 90 % (rot). Daher ist nach Ablauf von $Z_{Regelbedingung}$ zum Zeitpunkt $t_{Regel\_verletzt}$ die Regelbedingung verletzt. DEAR kann diese verletzte Regelbedingung unmittelbar auf der VM erkennen. Ohne Einsatz von DEAR, d. h., wenn das Alerting ausschließlich auf dem Monitoring-Server stattfindet, ist dieser Prozess von dem Aggregationszeitraum Z_{Aggr} abhängig. Angenommen, $Z_{Regelbedingung}$ beträgt 10 s, während Z_{Aggr} 60 s beträgt, so

Tabelle 5.2: Ergebnisse der Evaluation

		E1	E2	E3
Netzwerkverkehrvolumen (kB)		2.043	203	34
Last (%) (Telegraf)	CPU	0.0	0.0	2.5
	RAM	1.1	1.1	1.3
Last (%) (Esper)	CPU	0.11	0.10	0.52
	RAM	0.61	0.67	1.07
Speicherplatz (kB)		300	72	27.965
TTI (ms)	ohne DEAR	29	6.711	27.211
	mit DEAR	379	361	378

kann auf dem Monitoring-Server frühstens nach 60 s die verletzte Regelbedingung erkannt werden, während dies bei DEAR nach 10 s der Fall ist. Um einen fairen Vergleich zwischen dem TIG-Stack mit DEAR und dem TIG-Stack ohne DEAR zu ermöglichen, wird daher $Z_{Aggr} = Z_{Regelbedingung}$ gesetzt. Demnach werden jeweils Einstellung $E1$ mit der Alerting-Regel $R1$ evaluiert, $E2$ mit $R2$, und $E3$ mit $R3$. Nach Ablauf von $Z_{Regelbedingung}$ bzw. Z_{Aggr} wird gemessen, wie lange der Zeitraum der Verarbeitung Z_{Ver} ist, bis ein tatsächlicher Alert zum Zeitpunkt t_{Alert} im Alerting-Framework erzeugt wird. In Abbildung 5.8 entspricht Z_{Ver} daher der TTI.

Die TTI beim TIG-Stack ohne DEAR setzt sich daher in der Regel aus der Netzwerklatenz zwischen den VMs und der benötigten Zeit zur Auswertung der Alerting-Regel im Alerting-Framework zusammen. Zur TTI beim TIG-Stack mit DEAR kommt zusätzlich die lokale Auswertung auf dem Auswertungsmodul Esper hinzu. Um Schwankungen der Netzwerklatenz und resultierende Ungenauigkeiten zu vermeiden, wurden daher für die Messungen zur TTI alle Komponenten des TIG-Stack und DEAR auf einer einzigen VM installiert.

5.4.2.2 Ergebnisse

Die Ergebnisse der jeweiligen Messungen zu den verschiedenen Einstellungen $E1 - E3$ sind in Tabelle 5.2 dargestellt. Das Netzwerkverkehrvolumen

wurde zwischen der zu überwachenden VM und dem Monitoring-Server gemessen. Hierfür wurde kein Vergleich durchgeführt, da DEAR in erster Linie kein Netzwerkverkehrvolumen verringern soll, sondern trotz geringeren Netzwerkverkehrvolumen die Qualität des Alerting verbessern soll, d. h., TTI verringern und eine feingranulare Analyse ermöglichen. Daher sind bei den Messungen zum Netzwerkverkehrvolumen ausschließlich die Effekte von Z_{Aggr} zu sehen. Wie zu erwarten ist, sinkt das Netzwerkverkehrvolumen, indem Z_{Aggr} erhöht wird.

Die Last, die durch Telegraf und Esper entsteht, hat sich zwischen $E1$ und $E2$ nicht messbar verändert. In $E3$ hingegen erzeugt die hohe Abtastfrequenz von 100 Messwerten pro Sekunde einen vergleichsweise hohen Overhead bei der CPU-Auslastung von 2.5 %[1].

Speicherplatz bezieht sich auf den benötigten Speicherplatz des Monitoring-Daten-Buffers zur kurzfristigen Speicherung der Monitoring-Daten. Wie zu erwarten führt eine Erhöhung des Aggregationszeitraum $Z_{AggrBuffer}$ zu einer Verringerung des benötigten Speicherplatzes, da seltener aggregierte Monitoring-Daten gespeichert werden. Eine sehr hohe Abtastfrequenz ohne die Verwendung von Aggregation wie in $E3$ führen zu einem vergleichsweise hohen Datenvolumen der Monitoring-Daten, etwa 28 MB[2].

Wie in Tabelle 5.2 zu sehen ist, ist die TTI mit DEAR unabhängig von Z_{Aggr} und beträgt konstant ca. 370 ms. Ohne DEAR steigt die TTI von 29 ms bei Einstellung $E1$ auf 6.7 s bei $E2$ und auf 27.2 s bei $E3$.

5.4.3 Diskussion

Im Folgenden wird diskutiert, ob die in Abschnitt 5.2 definierten Anforderungen $A1 - A5$ auf Basis der Evaluationsergebnisse und der Funktionsweise von DEAR erfüllt worden sind.

[1] Die Auslastung kann sich je nach eingesetztem Agent oder Auswertungsmodul unterscheiden.

[2] Der benötigte Speicherplatz hängt von der verwendeten Technologie ab. Daten in InfluxDB werden in Write-Ahead-Logdateien gespeichert und mit der Kompressionsmethode Snappy komprimiert.

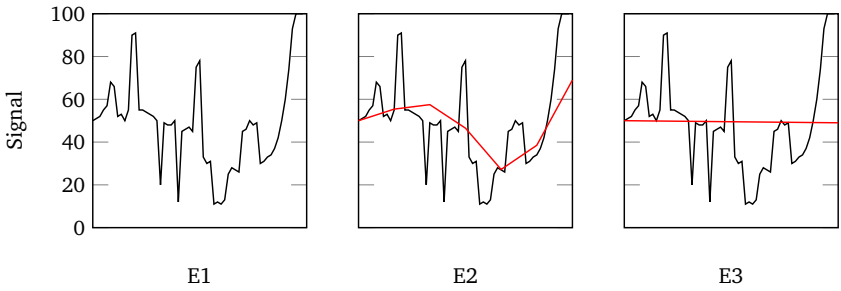

Abbildung 5.9: Originales Signal (schwarz) und aggregiertes Signal (rot) einer generierten Zeitreihe

Durch das Verteilen der Auswertung der Alerting-Regeln nahe an die Datenquellen, wurde das Netzwerkverkehrvolumen von der Granularität der Monitoring-Daten für das Alerting entkoppelt. Das bedeutet, dass äußerst feingranulare Monitoring-Daten für die Auswertung der Alerting-Regeln verwendet werden können ($A1$), ohne, dass es dadurch zu einem erhöhten Netzwerkverkehrvolumen kommt ($A2$). Um den Effekt der Aggregation und dem einhergehenden Verlust der Genauigkeit der Monitoring-Daten zu zeigen, ist in Abbildung 5.9 einerseits das Original eines generierten Signals zu sehen (schwarz) sowie die aggregierten Signale (rot) entsprechend der Aggregationszeiträume Z_{Aggr} der Einstellungen $E1 - E3$. Indem DEAR genutzt wird, ist das originale Signal in jeder Einstellung für das Alerting verfügbar, während im Ansatz ohne DEAR das aggregierte, weniger genaue Signal verwendet werden muss. Die Last von Telegraf und Esper sind akzeptabel in Einstellungen wie $E2$, wo bereits Vorteile gegenüber Ansatz ohne DEAR vorliegen. In $E3$ sind diese bei einer sehr hohen Abtastfrequenz jedoch deutlich erhöht. Daher sollten hohe Abtastfrequenzen bedacht und nur für Metriken verwendet werden, die dies in bestimmten Anwendungsfällen erfordern.

Wenn kein Aggregationszeitraum Z_{Aggr} verwendet wird (s. E1 in Tabelle 5.2), ist die TTI mit DEAR höher als ohne DEAR. Dies folgt aus der daraus, dass die Daten beim Einsatz von DEAR zusätzlich an das Auswertungsmodul geschickt und dort ausgewertet werden. DEAR wurde jedoch entwickelt,

um eine Abkopplung von Netzwerkverkehrvolumen und Granularität der Monitoring-Daten für Alerting zu ermöglichen, damit Aggregation nicht die Qualität des Alerting beeinträchtigt. Daher ist es widersprüchlich, DEAR ohne Aggregation zu verwenden. Wird eine Aggregation verwendet wie in $E2$ und $E3$, bleibt die TTI auch im Vergleich zu $E1$ weiterhin niedrig, während diese im Ansatz ohne DEAR vom Beginn des Aggregationszeitraums Z_{Aggr} abhängig ist. Ohne DEAR hängt die TTI davon ab, zu welchem Zeitpunkt in Telegraf ein neuer Aggregationszeitraum beginnt. Dieser Zeitpunkt t' liegt zwischen t* und $t_{Regel\_verletzt}$. Beginnt der neue Aggregationszeitraum zum Zeitpunkt t*, ist die TTI demnach entsprechend gering wie in $E1$. Beginnt der Aggregationszeitraum allerdings erst bei t', so muss zunächst der gesamte Aggregationszeitraum Z_{Aggr} abgewartet werden. Ab diesem Zeitpunkt $t'^{(2)}$ folgt wie in Abbildung 5.8 der Zeitraum Z_{Ver} zur Verarbeitung der Daten, bis schließlich zum Zeitpunkt t'_{Alert} ein entsprechender Alert erstellt wird. Daher ergibt sich ist im Vergleich zu DEAR eine höhere TTI, da die Regelbedingung weiterhin ab dem Zeitpunkt $t_{Regel\_verletzt}$ berechnet wird. Da t' beliebig zwischen t* und $t_{Regel\_verletzt}$ liegen kann, geht die TTI ohne DEAR bei Gleichverteilung bzgl. des Beginns des Aggregationszeitraums und einer ausreichend großen Anzahl von Tests demnach durchschnittlich gegen $\frac{Z_{Aggr}}{2} + Z_{Ver}$, während die TTI mit DEAR konstant bei etwa 370 ms bleibt. Im Allgemeinen ist daher mittels DEAR die TTI gesenkt worden und Anforderung $A3$ erfüllt.

Trotz einer lokalen, feingranularen Auswertung der Alerting-Regeln ist es weiterhin möglich, Monitoring-Daten an den Monitoring-Server zu schicken (s. $RP2$ in Abbildung 5.7). Auf diese Weise können Monitoring-Daten historisiert werden, um sie für zukünftige Analysen zu verwenden. Beispielsweise können hierdurch die groben Verhaltensweisen einer VM beschrieben werden, für die jedoch im Vergleich zu der Auswertung von Alerting-Regeln verhältnismäßig hohe Aggregationszeiträume Z_{Aggr} akzeptabel sind, wodurch wenig Netzwerkverkehrvolumen entsteht. Für Langzeitanalysen wie z. B. *wie verhält sich die CPU-Auslastung über die nächsten Tage?* sind feingranulare Monitoring-Daten weniger wichtig. Stattdessen können diese sogar zu schlechteren Ergebnissen führen, wenn viele Ausreißer und Rauschen

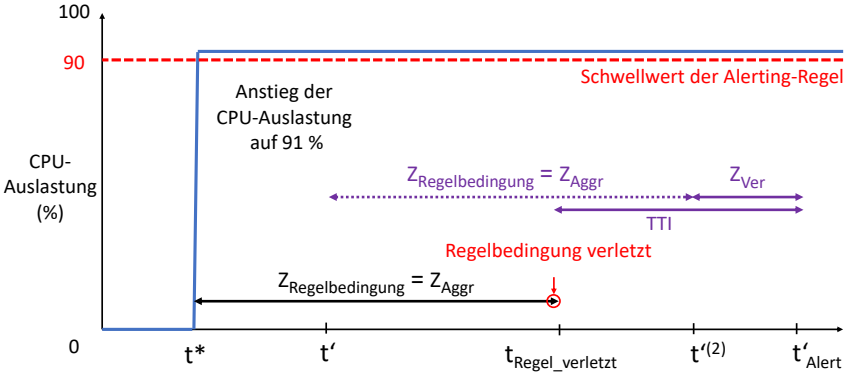

Abbildung 5.10: TTI ohne DEAR in Abhängigkeit von Beginn von Z_{Aggr}

in den Daten enthalten sind und somit z. B. die Genauigkeit der Vorhersagen von Machine-Learning-Modellen negativ beeinflussen können. Damit es beim Training der Machine-Learning-Modelle nicht zu einem *Overfit* kommt, werden die Trainingsdaten daher in der Regel zunächst bereinigt, um jene Ausreißer zu entfernen. Eine Möglichkeit der Bereinigung von Ausreißern in Zeitreihendaten ist die Aggregation. Daher werden mittels Aggregation aus feingranularen Monitoring-Daten Trenddaten, mit denen zur Trendvorhersage bessere Ergebnisse erzielt werden können. Des Weiteren können sich Systemadministratoren mithilfe von Trenddaten schneller einen Überblick über den Zustand eines Systems verschaffen. Zu beachten ist, dass falls die Ausreißer von Interesse sind und erkannt werden sollen, hierzu die lokale Auswertung von Alerting-Regeln verwendet werden kann. Die Folge längerer Aggregationszeiträume Z_{Aggr} ist allerdings, dass die VMs über diesen Zeitraum hinweg nicht mit dem Monitoring-Server kommunizieren und daher ein kompletter Ausfall einer VM gegebenenfalls nicht rechtzeitig erkannt wird. Hierfür ist es sinnvoll, parallel zum Monitoring der einzelnen Metriken in kurzen und regelmäßigen Abständen den *Heartbeat* (dt.: Herzschlag) der VM zu messen. Hierzu werden leichtgewichtige Nachrich-

ten zwischen einer VM und dem Monitoring-Server versendet, welche die Verfügbarkeit der VM bestätigen und einen nur geringen Overhead bzgl. des Netzwerkverkehrvolumen erzeugen.

Gleichzeitig kann es dennoch notwendig sein, auf sehr feingranulare Monitoring-Daten zuzugreifen, z. B., um die Ursache eines Alerts zu ergründen. Für diese sogenannte *Root Cause Analysis* (dt.: Fehlerursachenanalyse) könnte es in bestimmten Fällen hilfreich sein, die Monitoring-Daten in ihrer unveränderten Form zu analysieren, um bspw. die Auslastungsspitzen im anfänglich gezeigten Beispiel zu erkennen (s. Abbildung 5.1 Graph (2)). Hierfür wurde der Monitoring-Daten-Buffer eingeführt, der über einen definierten Zeitraum alle vom Agenten gesammelten Monitoring-Daten auf der VM speichert. Wird eine Alerting-Regel verletzt, können so die Monitoring-Daten weiterhin im Detail analysiert werden, die in Bezug zu einer verletzten Alerting-Regel stehen. Da bei vielen überwachten Metriken schnell eine große Menge an Monitoring-Daten entstehen kann, muss eine angemessene Dauer für die Speicherung gewählt werden, nach welcher die Monitoring-Daten unwiderruflich gelöscht werden, um den Speicherplatz einer VM nicht zu belasten. Daher kann es bei einer verletzten Alerting-Regel je nach Bedarf sinnvoll sein, die entsprechenden Monitoring-Daten aus dem Monitoring-Buffer in die zentrale Datenbank auf dem Monitoring-Server zu übertragen. Dies erzeugt zwar erneut Netzwerkverkehrvolumen, ermöglicht allerdings eine Historisierung der benötigten Daten.

Zusammenfassend ist es demnach möglich, aggregierte Monitoring-Daten zu historisieren sowie bei Bedarf feingranulare Monitoring-Daten für Analysen zur Verfügung zu stellen, wodurch Anforderung *A*4 erfüllt wird.

Da nur die Regelbedingungen der Alerting-Regeln verteilt werden, bleibt ein Single Point of Administration bestehen, d. h., dass Systemadministratoren Alerting-Regeln weiterhin zentral über das Alerting-Framework des Monitoring-Systems erstellen und verwalten können. Die Verteilung der Regelbedingungen und entsprechende Anpassungen der Alerting-Regeln erfolgen automatisch durch DEAR und erhöhen die Management-Komplexität daher nicht, wodurch Anforderung *A*5 erfüllt wird. Allerdings entstehen durch den Einsatz von DEAR zusätzliche Einstellungen wie z. B. die Entschei-

dung der Verteilungsstrategie oder geeigneter Aggregationszeiträume Z_{Aggr} und $Z_{AggrBuffer}$. Um Systemadministratoren bei diesen Entscheidungen zu helfen, können Metadaten über die Cloud-Umgebung genutzt werden. Wie in Abschnitt 5.3 beschrieben, hilft bspw. die Information, ob die überwachten VMs im selben Cluster liegen, bei der Entscheidung über eine geeignete Verteilungsstrategie. Hierfür wird in Kapitel 7 ein Kontextmodell vorgestellt, mit dem eine Topologie einer industriellen Cloud-Umgebung erstellt werden kann, welche diese Informationen beinhaltet.

5.5 Zusammenfassung

In diesem Kapitel wurde DEAR vorgestellt, ein Plug-in für zentralisierte Monitoring-Systeme, um die Vorteile von dezentralisierten Lösungsansätzen, z. B. eine geringere TTI oder weniger Netzwerkverkehrvolumen, zu nutzen. Dennoch bleibt im Gegensatz zu dezentralisierten Lösungsansätzen ein SPoA, um die Management-Komplexität nicht zu erhöhen. Hierfür wurde die Auswertung von Alerting-Regeln auf die VMs verschoben, um eine feingranulare Auswertung und weniger Netzwerkverkehrvolumen zu ermöglichen.

Auf Basis von mehreren Beispielen wurden die auftretenden Probleme beim Aggregieren von Monitoring-Daten aufgezeigt. Aus dieser Problematik heraus und unter Beachtung weiterer Aspekte, wie z. B. AIOps, wurden mehrere Anforderungen an DEAR aufgestellt. Zur Erfüllung dieser Anforderungen wurden die Konzepte und Komponenten von DEAR vorgestellt. Es wurde dargestellt, welche Änderungen beim Agenten durchgeführt werden, um die Verteilung zu ermöglichen.

Abschließend wurde mittels einer prototypischen Implementierung eine Evaluation durchgeführt, welche die Vorteile von DEAR gegenüber einem agentenbasierten, zentralisierten Monitoring-System darstellt. Es wurde gezeigt, dass bei der Aggregation von Monitoring-Daten der Einsatz von DEAR zu erheblichen Vorteilen, wie z. B. einer geringeren TTI, führt.

Kapitel 6

Verteilte Situationserkennung

Das in letzten Jahren immer weiter verbreitete Paradigma von I4.0 beschreibt eine noch tiefgreifendere Digitalisierung in der verarbeitenden Industrie und führt dadurch zu sogenannten Smart Factories [LCW08]. In I4.0 werden industrielle Assets jeglicher Art mit Sensoren ausgestattet. Dies soll bspw. eine Reduzierung von Stillstandzeiten zu Folge haben, indem Fehler frühzeitig erkannt und behoben werden sollen. Daher ist auch in I4.0 ein umfassendes Monitoring der industriellen Assets wichtig. Dies erfordert ein zeitgerechtes Erfassen und und eine Analyse der Monitoring-Daten, um basierend auf den Analyseergebnissen entsprechende Aktionen auszuführen. Im Vergleich zur Domäne des Cloud-Monitoring haben sich jedoch in I4.0 bisher keine allgemein verwendbaren Systeme für das Monitoring industrieller Assets etabliert. Beim Cloud-Monitoring können VMs verschiedener Cloud-Anbieter im Allgemeinen mit denselben Monitoring-Systemen überwacht werden, da die grundlegenden Metriken der virtuellen Maschinen, z. B. CPU- und Speicherauslastung, dieselben sind. Ebenso können in der Regel dieselben,

groben Schwellwerte zur Problemerkennung, bspw. eine CPU-Auslastung von über 90 %, verwendet werden. Bei der Überwachung industrieller Assets hingegen müssen unterschiedliche Metriken, z. B. die Drehzahl einer Bohrmaschine, überwacht werden – Metriken, die bei anderen industriellen Assets nicht vorhanden sind. Des Weiteren können bereits bei Bohrmaschinen unterschiedlicher Hersteller oder verschiedener Einsatzszenarien verschiedene Schwellwerte zur Problemerkennung bestehen, die Fehler in der Produktion vermeiden sollen. Daher müssen im Bereich von I4.0 für das Monitoring industrieller Assets maßgeschneiderte Systeme zum Einsatz kommen.

Zu diesem Zweck wurden in der Vergangenheit Kontextmodelle entwickelt, um die Gegebenheiten einer industriellen Umgebung abzubilden [GBH+05]. Mittels der Erfassung von Sensordaten wurden auf Basis dieser Kontextmodelle kontextsensitive Anwendungen realisiert. Ein Beispiel hierfür sind kontextsensitive Workflows, die auf Sensordaten reagieren können und ihre Ausführung entsprechend anpassen. Beispielsweise kann somit der Produktionsablauf in einer Produktionslinie bei Ausfällen einzelner Maschinen angepasst werden, um weiterhin eine fortlaufende Fertigung zu ermöglichen [WSBL15]. Die Herausforderung bei kontextsensitiven Anwendungen im Allgemeinen liegt jedoch auf der Verwaltung der Vielzahl von low-level Sensordaten [HWS+16]. Für kontextsensitive Workflows bedeutet dies beispielsweise, dass mit steigender Anzahl verwendeter Kontextdaten auch eine steigende Anzahl möglicher Abzweigungen im Workflow modelliert werden müssen [WSBL15], um alle möglichen Zustände abbilden und darauf reagieren zu können.

Das Forschungsprojekt SitOPT [Uni21] hatte daher zum Ziel, kontextsensitive Workflows mit situationssensitiven Workflows zu ersetzen. Aus low-level Sensordaten wurden hierzu zunächst höherwertige Informationen – genannt Situationen – abgeleitet, um daraufhin basierend auf den erkannten Situationen Workflows zu modellieren. Der Vorteil liegt darin, dass die Erkennung von Situationen in eine Situationserkennung ausgelagert werden kann und Workflows unabhängig vom bereitliegenden Kontext modelliert werden können. Ein Workflow, der einen Ausfall einer Produktionsmaschine beachten soll, muss daher nicht jede Kombination von Kontextdaten

modellieren, die jenen Ausfall beschreiben würden. Mit einer separaten Situationserkennung muss der Workflow stattdessen nur auf eine Situation, z. B. *Maschinen_kritisch* reagieren. Wie diese Situation zustande kommt, wird auf die Situationserkennung ausgelagert und vereinfacht somit die Modellierungskomplexität der Workflows [WSBL15]. Des Weiteren hat die Auslagerung zum Vorteil, dass eine erkannte Situation für mehrere situationssensitive Workflows sowie weitere situationssensitive Anwendungen verwendet werden kann, ohne die Details, d. h. die Abhängigkeiten der Kontextdaten, in jeder Anwendung aufs Neue modellieren zu müssen.

Die Situationserkennung in SitOPT wird auf Basis von Situationstemplates (s. Abschnitt 2.4) durchgeführt und die Situationen wurden in einer monolithischen IT-Infrastruktur in der Public-Cloud erkannt. In Domänen wie I4.0 ist dieser Ansatz jedoch ungeeignet, um wichtige Anforderungen wie eine niedrige Latenz zu erfüllen [YWJ+15]. Gleichzeitig müssen zunächst alle Kontextdaten in die Public-Cloud übertragen werden, was zu einem hohen Netzwerkverkehrvolumen führt. Um diesen Problemen entgegenzutreten, muss die Situationserkennung so nahe wie möglich an der Quelle der Kontextdaten stattfinden. Durch das Aufkommen von Edge-Computing bzw. Edge-Clouds (s. Abschnitt 2.1) sind inzwischen jedoch genügend Rechenressourcen nahe an den Datenquellen verfügbar, um eine Datenverarbeitung, bspw. eine Situationserkennung, ohne hohe Netzwerklatenz zu ermöglichen [SCZ+16].

In diesem Kapitel werden daher die Vorarbeiten zu SitOPT aufgegriffen, um die Vorteile des Edge-Computing für die Situationserkennung zu nutzen. Jedoch ist es nicht in allen Fällen möglich, die gesamte Situationserkennung auf die Edge zu verlagern, z. B., weil nicht alle benötigten Datenquellen an einem Standort verfügbar sind. Je nach Szenario ist es daher notwendig, die Situationserkennung in der Public-Cloud, auf der Edge, oder in einer Kombination aus beidem auszuführen. Zunächst werden die Anforderungen an die Situationserkennung im Rahmen von Industrie 4.0 aufgestellt und anschließend die Auswirkungen dieser drei verschiedenen Möglichkeiten auf die Situationserkennung basierend auf den Anforderungen analysiert.

Des Weiteren hat die Kombination von Public-Cloud und Edge zur Folge, dass die Situationserkennung verteilt werden muss. Dies wiederum führt dazu, dass sowohl das Schema der Situationstemplates als auch die Ausführung der Situationserkennung angepasst werden müssen. Ergänzend wird eine schichtenbasierte Modellierung eingeführt, die es ermöglicht, neben Kontextdaten auch erkannte Situationen als Input für die Situationserkennung zu verwenden. Es wird gezeigt, wie die Situationserkennung automatisch auf Public-Cloud und Edge verteilt werden kann. Um die Modellierung von Situationstemplates zu vereinfachen, wird zudem ein web-basiertes Modellierungswerkzeug vorgestellt.

Dieses Kapitel ist eine überarbeitete und zusammenfassende Version mehrerer Publikationen des Autors [MHWM18; MHWM19].

Der Rest dieses Kapitels ist wie folgt strukturiert: Im nächsten Kapitel werden zunächst verwandte Arbeiten im Bereich der Situationserkennung vorgestellt. In Abschnitt 6.2 werden die Anforderungen an die Situationserkennung anhand eines Beispielszenarios aufgestellt. Abschnitt 6.3 enthält die Anpassungen und Erweiterungen der Konzepte aus SitOPT sowie die Analyse der unterschiedlichen Ausführungsumgebungen. In Abschnitt 6.4 wird gezeigt, wie eine Verteilung der Situationserkennung erfolgt, beginnend bei der Modellierung eines Situationstemplates. Abschnitt 6.5 enthält die prototypische Implementierung und Abschnitt 6.6 fasst dieses Kapitel zusammen.

6.1 Verwandte Arbeiten

Verwandte Arbeiten beschreiben Ansätze für eine verteilte Situationserkennung mittels Ontologien, z. B. Fang et al. [FZYZ08]. Diese Ansätze genügen aufgrund des zeitintensiven Reasonings jedoch nicht den Latenzanforderungen zeitkritischer Szenarien, wie sie in Domänen wie I4.0 herrschen [LCW08]. Das Ziel dieses Ansatzes ist es, eine niedrige Latenz im Bereich von Millisekunden für die verteilte Situationserkennung zu erhalten.

Im Gegensatz dazu befinden sich viele Ansätze auf Basis von Ontologien im Bereich von Minuten, auch, wenn keine Verteilung der Situationserkennung vorliegt.

Im Bereich des verteilten Complex Event Processings (CEP), zielen Schilling et al. [SKPR10] auf die Nutzung verschiedener CEP-Systeme ab, indem eine gemeinsame Metasprache verwendet wird. Dies erlaubt, verschiedene CEP-Systeme gleichzeitig zu nutzen und die einzelnen Ergebnisse anschließend zu integrieren. Dieser Ansatz kann für die in diesem Kapitel vorgestellte verteilte Situationserkenunng hilfreich sein, da hierdurch die Beschränkung auf eine einzige Ausführungsumgebung aufgehoben wäre. Jedoch müssen in diesem Ansatz die CEP-Queries händisch geschrieben und verteilt werden, was für unerfahrenes Personal, dem die benötigten Fähigkeiten fehlen, eine schwierige Aufgabe darstellt. Im Beitrag dieses Kapitels hingegen wird ein Abstraktionslevel mittels Situationstemplates bereitgestellt, das die Modellierung durch ein grafisches Modellierungswerzkzeug unterstützt. Des Weiteren werden die Nutzer bei der Aufteilung des Situationstemplates und der Verteilung auf verschiedene Ausführungsumgebungen unterstützt.

Weitere Ansätze im Bereich von verteiltem CEP, z. B. von Schultz-Møller et al. [SMP09], folgen dem Ansatz des automatischen Umschreibens von CEP-Queries. Diese CEP-Queries werden durch automatisches Umschreiben aufgeteilt und anschließend basierend auf einem Kostenmodell verteilt, welches größtenteils auf der CPU-Auslastung der zur Verfügung stehenden Knoten basiert. In diesem Ansatz hingegen werden darüber hinaus z. B. Datenschutz- oder sicherheitsrelevante Aspekte beachtet.

Zuletzt soll der Unterschied zwischen einer verteilten Situationserkennung und DEAR, dem Ansatz zur verteilten Auswertung von Alerting-Regeln in Kapitel 5, deutlich gemacht werden. Beide Ansätze haben allgemein zum Ziel, eine Verarbeitung von Daten nahe an der Datenquelle zu ermöglichen, um Anforderungen bzgl. Latenz und Netzwerkverkehrvolumen zu genügen. Ein Situationstemplate kann im Allgemeinen als Kombination eines Monitoring-Agenten und zugehöriger Alerting-Regel betrachtet werden, da neben der Definition benötigter Kontextdaten auch die Bedingungen für das Auftreten einer Situation (vgl. Alert) definiert werden. Der Vorteil beim Einsatz von

Agenten im Cloud-Monitoring ist die Vielzahl von vordefinierten Konfigurationen, die das Monitoring der üblichen Metriken wie CPU-Auslastung ermöglichen. Aufgrund der starken Heterogenität unter industriellen Assets im Vergleich zu virtuellen Maschinen müssen Situationstemplates stattdessen individuell maßgeschneidert werden. Zusätzlich gelten striktere Anforderungen bzgl. der Latenz. Somit ist DEAR mit einer zentralisierten Verwaltung von Alerts nicht geeignet.

Demnach ist eine verteilte Situationserkennung auf Basis von Situationstemplates mit dezentralisierten Monitoring-Ansätzen verwandt. Dennoch werden Alerting-Regeln separat von den Agenten zur Datenerfassung modelliert. Situationstemplates hingegen werden maßgeschneidert für spezielle industrielle Assets modelliert. Daher ist es aufgrund einer übersichtlicheren Modellierung vorzuziehen, die Kontextdaten sowie deren Relationen und Bedingungen für das Auftreten einer Situation, gemeinsam in einem Situationstemplate zu modellieren, anstatt verteilt innerhalb eines Agenten und einer zugehörigen Alerting-Regel.

6.2 Anforderungen

In diesem Abschnitt wird ein Beispielszenario aus der Domäne I4.0 vorgestellt, das im Rahmen dieses Kapitels für die Anforderungsanalyse verwendet wird. Das Szenario wird in Abbildung 6.1 dargestellt und beschreibt die Erkennung der Situation *Supply-Chain_kritisch*, die domänenübergreifend mehrere Situationen vereint. Hierfür werden Kontextdaten der industriellen Assets *Maschine 1, Maschine 2* aus der Produktionsdomäne und *LKW* aus der Logistikdomäne gesammelt. Die Maschinen sind Teil der gleichen Produktionslinie und produzieren fortlaufend neue Werkstücke auf Basis von Werkmaterial, während der LKW diese Werkmaterialien an die Smart-Factory liefert. Das Ziel ist die Erkennung der folgenden kritischen Situationen, die z. B. durch Fehler der Maschinen oder einer Verzögerung der Lieferung entstehen können: (i) *Produktionslinie_kritisch*, die durch Fehler einer oder

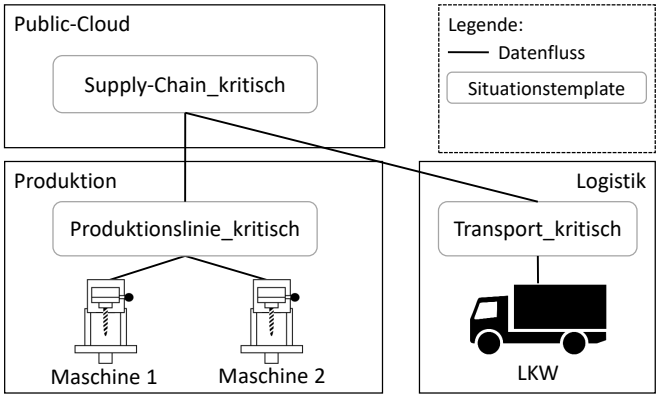

Abbildung 6.1: Beispielszenario für eine verteilte Situationserkennung

beider Maschinen entstehen kann, (ii) *Transport_kritisch*, die ein Problem des LKWs anzeigt, und (iii) *Supply-Chain_kritisch*, die durch Probleme in der Produktionslinie und/oder des LKWs entsteht.

Bei der Anwendung der anfänglichen Methode auf dieses Szenario treten neue Herausforderungen auf, die in den folgenden Anforderungen erfasst werden. Es werden sieben neue Anforderungen definiert, die bezüglich der Modellierung von Situationstemplates und der Ausführung der Situationserkennung unterteilt werden. Diese werden im Folgenden erläutert:

- *A*1 **- Mächtigere Situationstemplates:** Im ursprünglichen Ansatz von SitOPT konnten einzelne industrielle Assets effizient überwacht werden [HWS+16]. Im jetzigen Szenario ist es allerdings erforderlich, die Abhängigkeiten mehrerer Assets zueinander innerhalb eines einzigen Situationstemplates zu modellieren, z. B., um die Situation *Produktionslinie_kritisch* zu erkennen.
- *A*2 **- Geringe Modellierungskomplexität und Domänenunabhängigkeit:** Durch die aus Anforderung *A*1 entstehenden Änderungen an Situationstemplates, können durch die Modellierung mehrerer Assets größere und komplexere Situationstemplates modelliert werden, was zu einer aufwändigen und fehleranfälligen Modellierung führen

kann. Zusätzlich werden in diesem Beispiel Experten verschiedener Domänen, z. B. Produktion und Logistik, benötigt, um ein Situationstemplate zu modellieren. Dies kann zu hohen Kosten aufgrund des Overheads bzgl. der Kommunikation der Experten führen. Daher wird ein erweiterter Ansatz für die Modellierung benötigt, um die Komplexität des Modellierens von Situationstemplates zu verringern.

- *A*3 **- Niedrige Latenz:** Eine niedrige Latenz ist eine Grundvoraussetzung in vielen Szenarien in Industrie 4.0. So unterliegt beispielsweise die Automationsebene strikten Anforderungen bezüglich der Ende-zu-Ende-Latenz von bis zu einer Millisekunde oder weniger [YWJ+15]. Daher muss es möglich sein, kritische Situationen, wie z. B. Maschinenfehler, die zum Ausfall der Produktion führen könnten, rechtzeitig zu erkennen.
- *A*4 **- Niedriges Netzwerkverkehrvolumen:** Der stetig zunehmende Einsatz von Sensoren führt in modernen Szenarien zu einer großen Menge von erzeugten Kontextdaten, die verarbeitet und gespeichert werden müssen, um Situationen zu erkennen. Beispielsweise produziert ein autonomes Fahrzeug etwa 35 GB an Daten pro Stunde [MZP+17]. Im Vergleich dazu wurde ein Fahrtest durchgeführt und eine maximale und minimale Upload-Geschwindigkeit von 30 Mbps (13,5 GB/Stunde) bzw. 3,5Mbps (1,58 GB/Stunde) aufgezeichnet auf Basis des aktuellen Mobilfunkstandards LTE-A. Daher ist es mit aktuellen Technologien unmöglich, alle Daten eines autonomen Fahrzeugs in die Cloud zu übertragen. Die Reduktion des Netzwerkverkehrvolumens stellt somit einen wichtigen Faktor bei der Situationserkennung dar.
- *A*5 **- Niedrige Kosten:** Die Kosten stellen in der Industrie offensichtlich immer einen essentiellen Faktor dar. Das Aufbauen und Verwalten einer eigenen Infrastruktur kann zu hohen Kosten führen. Im Vergleich dazu stellt der Pay-as-you-go-Ansatz des Cloud-Computing unter Umständen eine kostengünstigere Alternative dar. Die verschiedenen Varianten müssen in Anbetracht der zu erfüllenden Anforderungen analysiert werden, um dem Nutzer eine klare Übersicht geben zu können.

- *A6* **- Sicherheit:** Die Daten eines Unternehmens müssen auf einer sicheren Plattform verarbeitet werden, um gegen Diebstahl und Missbrauch abgesichert zu sein. Hierbei führt der Einsatz von Public-Clouds zu potentiellen Sicherheitsrisiken, die den Umgang mit sensiblen Informationen eines Unternehmens erschweren.
- *A7* **- Unternehmensübergreifende Situationserkennung:** Produkte werden kaum noch von einem einzigen Unternehmen hergestellt, sondern bedienen sich einem Netz vieler Zulieferer. Demnach muss für eine Situationserkennung zur Überwachung der Produktion auf die Daten mehrerer Unternehmen zugegriffen werden, um beispielsweise die *Supply-Chain_kritisch* zu erkennen.

6.3 Verteilte Situationserkennung

Bisher waren Situationstemplates für die Überwachung einzelner Maschinen gedacht. Grund hierfür war, die Modellierungs- und Ausführungskomplexität gering zu halten. Jede Maschine konnte mittels der an ihr angebrachten Sensoren überwacht werden. Die Sensoren einer Maschine wurden an einer Plattform registriert und verwaltet [HWBM16a], auf deren Basis eine effiziente Situationserkennung durchgeführt wurde [FHWM16], und die resultierenden Situationen wurden verwaltet [MHWM17].

In diesem Abschnitt werden die Verbesserungen bezüglich der Modellierung von Situationstemplates und der Ausführung der Situationserkennung als Grundlage für eine verteilte Situationserkennung vorgestellt, um die im vorigen Abschnitt definierten Anforderungen $A1 - A7$ zu erfüllen. Um die Verteilung zu ermöglichen, wird das Konzept des Edge-Computing verwendet, bei dem nahe an der Quelle der Kontextdaten gelegene Hardware für die Ausführung der Situationserkennung verwendet wird. Weiterhin bleibt eine rein Public-Cloud-basierte Situationserkennung oder eine Kombination beider Technologien möglich. Diese drei Möglichkeiten zur Verteilung, (i)

Edge-Cloud, (ii) Public-Cloud, und (iii) eine Kombination von Edge- und Public-Cloud, der Situationserkennung werden vorgestellt und jede dieser Strategien auf Basis der oben genannten Anforderungen analysiert.

6.3.1 Verbesserungen bei der Modellierung von Situationstemplates

Die Einschränkung, dass nur jeweils eine Maschine pro Situationstemplate modelliert werden kann, stellt Grenzen an die Komplexität der zu erkennenden Situation auf. Beispielsweise lassen sich komplexere Situationen wie z. B. *Produktionslinie_kritisch* oder *Supply Chain_kritisch* nicht durch den Zustand einer einzigen Maschine oder Ähnlichem darstellen, sondern nur durch mehrere, in Abhängigkeit zueinander stehenden Assets. Diese Einschränkung basiert allerdings auf der Instanziierung der Situationstemplates und nicht aufgrund von Unzulänglichkeiten bei der Modellierung. Grund hierfür ist, dass bei der Instanziierung eines Situationstemplates nur Angaben für eine einzelne Maschine gemacht werden können. Daraus folgt, dass alle im Situationstemplate modellierten Kontext-Knoten ausschließlich Kontextdaten einer einzigen Maschine erhalten können. Durch eine entsprechende Änderung bei der Instanziierung wird für jeden Kontext-Knoten abgefragt, zu welcher Maschine dieser gehört bzw. aus welcher Quelle die Kontextdaten stammen. Somit wurde die Modellierung mehrerer Assets ermöglicht, ohne eine Änderung am Schema der Situationstemplates zu benötigen.

Durch die Möglichkeit, mehrere Maschinen in einem Situationstemplate zu modellieren, steigt die Modellierungskomplexität eines Situationstemplates potentiell stark an. Je nach Anzahl von benötigten Kontextknoten der einzelnen Assets und der Relationen der Assets zueinander kann die Modellierung sehr aufwändig und fehleranfällig werden. Gleichzeitig können sich Teile des Situationstemplates wiederholen, wenn baugleiche Assets überwacht werden, da sie die gleichen Sensoren und folgend die gleichen Kontextdaten zur Verfügung stellen. Beispielsweise ist dies bei der Situation *Produktionslinie_kritisch* in Abbildung 6.2 der Fall. Zu bemerken ist, dass die unterschiedlichen Zahlenwerte in den Kontext-Knoten *Werkzeug* und *Materi-*

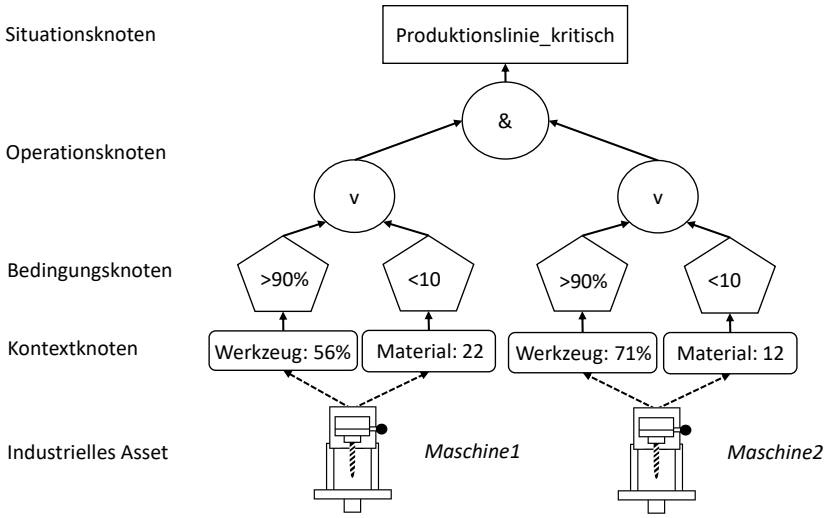

Abbildung 6.2: Mehrere Assets in einem Situationstemplate

al beispielhafte Kontextdaten der jeweiligen Maschinensensoren zur Laufzeit der Situationserkennung darstellen sollen und nicht Teil des modellierten Situationstemplates sind.

Um dieser steigenden Modellierungskomplexität entgegenzutreten, wird eine schichtenbasierte Modellierung für Situationstemplates eingeführt. Das Ziel hierbei ist es, bereits modellierte Situationstemplates und die dadurch erkannten Situationen als Input für weitere Situationstemplates zu verwenden. Durch die Verteilung der Situationserkennung auf Public-Cloud und Edge und die Verwendung von Situationen als Input werden folgende Begrifflichkeiten eingeführt, um die unterschiedlichen Arten von Situationstypen zu beschreiben:

- **Subsituation:** Eine Situation, die als Eingabe für eine weitere Situationserkennung verwendet wird, wird als Subsituation bezeichnet. Subsituationen können direkt aus Kontextdaten oder selbst aus anderen Subsituationen heraus erkannt werden.

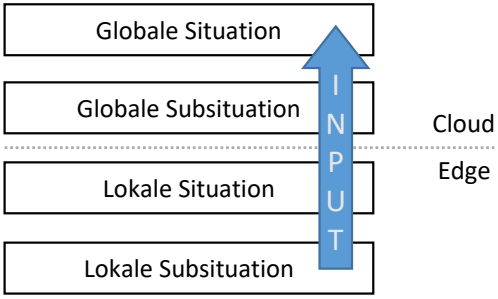

Abbildung 6.3: Verschiedene Situationen der schichtenbasierten Modellierung

- **Lokale Situation:** Eine Situation, die auf der Edge erkannt wird, wird als lokale Situation bezeichnet, da nur im lokalen Netz auf sie zugegriffen werden kann. Werden lokale Situationen als Input für weitere Situationserkennungen verwendet, gilt der Begriff *lokale Subsituation*.
- **Globale Situation:** Eine Situation, die in der Public-Cloud erkannt wird, wird als globale Situation bezeichnet, da sie über die Cloud global verfügbar ist. Werden globale Situationen als Input für weitere Situationserkennungen verwendet, gilt der Begriff *globale Subsituation*.

Die verschiedenen Situationstypen sind in Abbildung 6.3 dargestellt. Analog dazu werden die zugehörigen Situationtemplates als *Subsituationstemplates, lokale Situationstemplates, lokale Subsituationstemplates, globale Situationstemplates* und *globale Subsituationstemplates* bezeichnet.

Durch die Einführung des Schichtenmodells ist es möglich, die Komplexität bei der Modellierung großer Situationstemplates durch die Verwendung von Subsituationen zu verringern. Das in Abbildung 6.2 dargestellte Situationstemplate ist äquivalent zu dem in Abbildung 6.4 dargestellten Situationstemplate. Wie bereits oben beschrieben, können durch die Modellierung baugleicher Assets Teilbäume des Situationstemplates gleich sein. In Abbildung 6.2 ist dies für die Teilbäume ausgehend von *Maschine 1* und *Maschine-2* bis zu den jeweiligen *v*-Knoten (logisches ODER) der Fall. Durch

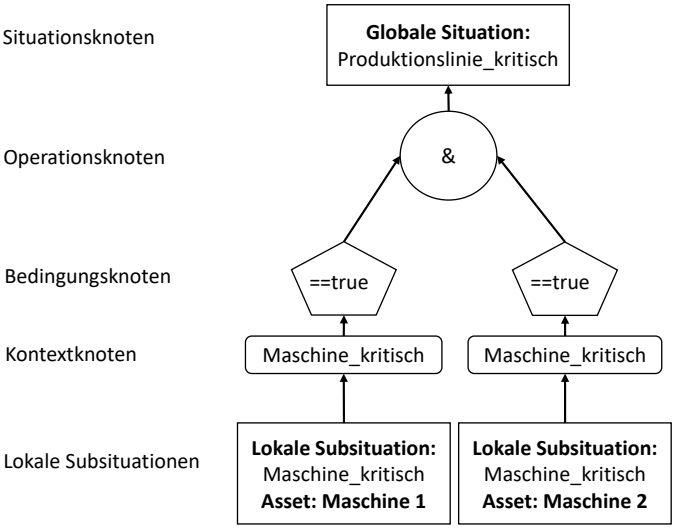

Abbildung 6.4: Situationen als Input in Situationstemplates

die Verwendung von Situationen als Input, werden diese Teilbäume im ersten Schritt aus dem ursprünglichen Situationstemplates extrahiert bzw. bei der Neumodellierung separat modelliert. Dadurch entsteht ein eigenes Situationstemplate *Maschine_kritisch*, welches jeweils auf *Maschine 1* und *Maschine 2* angewendet wird. Die dadurch erkannten Situationen werden anschließend als lokale Subsituationen als Input für das ursprüngliche Situationstemplate *Produktionslinie_kritisch* verwendet, in welchem Kontext- und Bedingungsknoten entsprechend angepasst werden, um statt den Maschinensensoren einen Boolean zu erhalten und diesen auf den Wahrheitswert *wahr* überprüfen wie in Abbildung 6.4 zu sehen ist.

Durch diesen Ansatz können mehrere Vorteile erzielt werden: (i) Baugleiche Assets müssen nicht mehrfach modelliert werden. Durch die Modellierung des Situationstemplates *Maschine_kritisch* kann die daraus resultierende Situation durch einmalige Modellierung in allen weiteren Situationstemplates verwendet werden, was zu einer starken Reduzierung des

Modellierungsaufwands führen kann. Des Weiteren können die einzelnen Situationstemplates unabhängig voneinander verändert oder ausgetauscht werden. (ii) Auch bei nicht baugleichen Assets ist eine separate Modellierung von Subsituationstemplates sinnvoll. Hierdurch wird die Komplexität des ursprünglichen Situationstemplates reduziert, da anstatt der detaillierten Modellierung der einzelnen Assets und deren Relationen zueinander nur noch ausschließlich die Relationen modelliert werden müssen. (iii) In vielen Fällen können die Subsituationen selbst von Interesse sein. Beispielsweise ist das Auftreten der Situation *Maschine_kritisch* unabhängig von der Situation *Produktionslinie_kritisch* wichtig und könnte anders behandelt werden als die Situation bzgl. der Produktionslinie.

6.3.2 Verbesserungen bei der Ausführung der Situationserkennung

Die Verbesserungen bzgl. der Modellierung von Situationstemplates im letzten Abschnitt sind die Grundlage für die Verteilung der Situationserkennung. Wie bereits beschrieben, wurde die Situationserkennung im bisherigen Ansatz ausschließlich in einer Public-Cloud durchgeführt. Demnach mussten alle Kontextdaten zunächst in die Cloud transportiert werden, um als Input für die Situationserkennung dienen zu können, was zu einem hohen Netzwerkverkehrvolumen und höherer Latenz führt (s. Anforderungen *A*3 und *A*4). Um diese Herausforderungen zu bewältigen, wird das in den letzten Jahren entstandene Paradigma des Edge-Computing eingesetzt. In unseren Ansatz bedeutet dies eine Verarbeitung von Kontextdaten nahe an den Assets zur Erkennung von Situationen.

Indem im neuen Ansatz Edge-Computing zum Einsatz kommt, kann eine Verteilung der Situationserkennung auf Public-Cloud und Edge stattfinden. Die Verteilung der Situationserkennung für das Situationstemplate aus Abbildung 6.1 ist offensichtlich. Auf Basis der schichtenbasierten Modellierung, können die lokalen Subsituationen *Produktionslinie_kritisch* und *Transport_kritisch* sowie die globale Situation *Supply Chain_kritisch* modelliert werden. Die Situationserkennung für die lokalen Subsituationen findet hierbei auf der Edge statt, d.h., direkt auf dem Shopfloor der Fabrik bzw.

im LKW, und jene für die globale Situation wird in der Public-Cloud ausgeführt. Die lokal erkannten Situationen werden an die Public-Cloud geschickt und als Input für die Erkennung der globalen Situation verwendet. Allerdings können je nach Use-Case auf Basis der Anforderungen $A3 - A7$ andere Verteilungen von Vorteil sein.

Im Folgenden werden die Verbesserungen bzgl. der Situationserkennung gezeigt, die aus der Verteilung der Situationserkennung heraus resultieren. Zuerst wird das Konzept der *Kontextentfernung* und dessen Vorteile vorgestellt. Anschließend werden drei grundlegende Verteilungsstrategien für die Situationserkennung vorgestellt und auf Basis der Anforderungen $A3 - A7$ analysiert, um eine Entscheidungshilfe für die Wahl der am besten geeigneten Verteilungsstrategie für ein bestimmtes Szenario zu ermöglichen.

6.3.2.1 Kontextentfernung

Das Resultat einer Situationserkennung ist ein sogenanntes Situationsobjekt, welches über ein Situationsmodell definiert ist, das in Kapitel 7 vorgestellt wird. Das Situationsmodell definiert alle relevanten Informationen zu einer erkannten Situation, z.B., das dafür verwendete Situationstemplate und das überwachte Asset mit dazugehörigen Sensordaten. Ausschlaggebend für die Kontextentfernung sind die Kontextdaten, die ebenfalls im Situationsobjekt hinterlegt sind. Werden eine große Anzahl Kontextdaten für die Erkennung einer Situation benötigt, entsteht ein entsprechend großes Situationsobjekt, bei dem ein Großteil der Datenmenge auf die Kontextdaten zurückzuführen ist. Dies hat zur Folge, dass bei der Übertragung einer auf der Edge erkannten Subsituation das Situationsobjekt mitsamt aller Kontextdaten an die Public-Cloud geschickt wird, wenn sie als Input für die Erkennung einer globalen Situation verwendet wird. Demnach entsteht allerdings keine Einsparung des Netzwerkverkehrvolumens durch den Einsatz von Edge-Computing. Das Ziel der Kontextentfernung ist es daher, die Kontextdaten aus dem Situationsobjekt zu entfernen, um bei der Übertragung einen Großteil der benötigten Netzwerkbandbreite einzusparen.

Bei der Kontextentfernung findet daher ein Kompromiss statt, über den der Nutzer individuell entscheiden muss. Ursprünglich war vorgesehen, alle Kontextdaten im Situationsobjekt zu hinterlegen, um eine detaillierte, nachträgliche Analyse zu aufgetretenen Situationen und deren Ursache zu ermöglichen. Die Kontextdaten waren bereits ohnehin an die Public-Cloud gesendet worden und die höhere Datenmenge von Situationsobjekten hatte sich nur auf den benötigten Festplattenspeicherplatz ausgewirkt. Wird eine Kontextentfernung durchgeführt, wird zwar ein Großteil des Netzwerkverkehrvolumens eingespart, allerdings auf Kosten einer fehlenden, nachträglichen Analyse der Situationen, da die Kontextdaten nach der Situationserkennung dauerhaft gelöscht werden.

6.3.2.2 Ausführungsumgebungen

Die Verteilung der Situationserkennung hängt vom Szenario und den damit verbundenen Anforderungen $A3-A7$ statt. Daher existiert keine allgemein *beste* Ausführungsumgebung. Stattdessen werden im Folgenden drei Möglichkeiten zur Ausführung der Situationserkennung vorgestellt: (i) Public-Cloud, (ii) Edge, und (iii) Hybrid, eine Kombination von Public-Cloud und Edge-Computing. Diese sind in Abbildung 6.5 und Abbildung 6.6 dargestellt und werden ausgehend von dem in Abbildung 6.1 dargestellten Szenario und den Anforderungen $A3-A7$ analysiert.

Public-Cloud (Abbildung 6.5, links): Die Verwendung der Public-Cloud entspricht der ursprünglichen Ausführungsumgebung der Situationserkennung. Hierbei werden alle Kontextdaten, d. h., im Beispiel die Sensordaten der Maschinen und die des LKWs, an die Public-Cloud übertragen, wo die Situationserkennung stattfindet. Auch trotz der Vorteile des Edge-Computing kann eine Situationserkennung in der Public-Cloud eine sinnvolle Option sein. Die Einführung von Edge-Computing ist keine triviale Aufgabe und stellt Unternehmen vor viele Herausforderungen [SCZ+16]. Vor allem Unternehmen mit wenig IT-Expertise oder keiner eigenen IT-Abteilung profitieren finanziell davon, ihre IT-Infrastruktur und die dazu benötigte Expertise ganzheitlich an

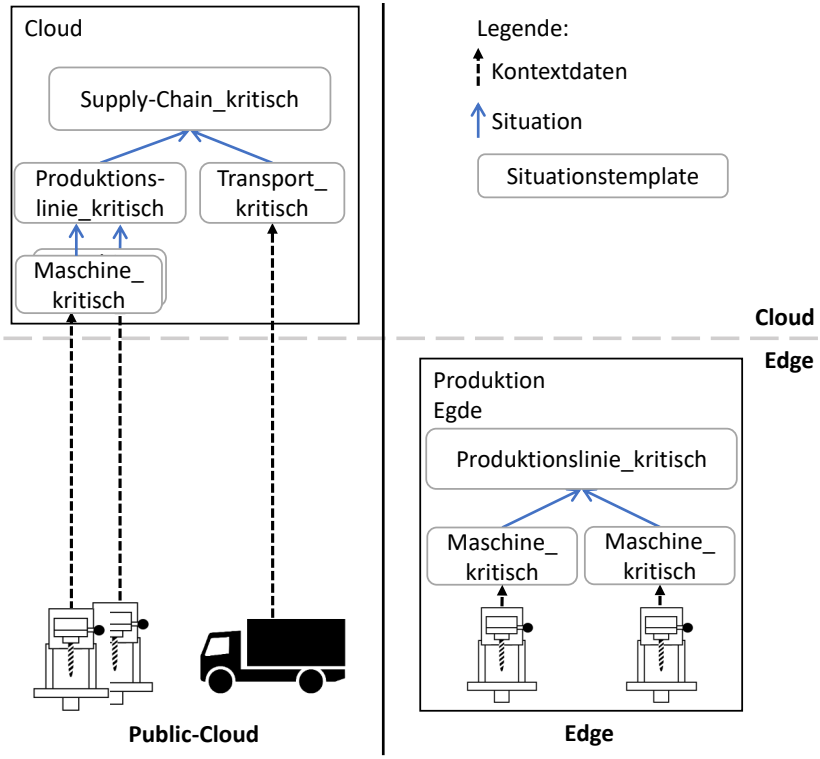

Abbildung 6.5: Verteilung der Situationserkennung auf Public-Cloud oder Edge

externe Cloud-Anbieter auszulagern. Vor allem klein- und mittelständische Unternehmen können sich auf diese Weise auf ihr Kernprodukt fokussieren und mittels des bedarfsabhängigen Pay-as-you-go-Modells eine kosteneffektive und skalierbare IT-Infrastruktur erhalten. Die Anforderungen *A3* – *A7* haben folgende Auswirkungen:

- *A3* - **Niedrige Latenz:** Derzeit sind sehr niedrige Latenzzeiten im Bereich weniger Millisekunden allein aufgrund der Netzwerklatenz zwischen der Datenquelle und der Public-Cloud nicht zu erreichen.

Daher kann die Anforderung *A*3 nicht erfüllt werden und Niedriglatenz-Szenarien, wie z. B. eine automatische Notabschaltung einer Produktionsmaschine im Falle eines Fehlers können nicht gelöst werden.

- *A*4 **- Niedriges Netzwerkverkehrvolumen:** Da zunächst die gesamten Kontextdaten an die Public-Cloud übermittelt werden müssen, ist es nicht möglich, das Netzwerkverkehrvolumen zu reduzieren. Die Kontextentfernung ist nicht anwendbar, da diese erst nach der Situationserkennung erfolgen kann. Daher sind Szenarien, bei denen viele, hochfrequente Kontextdaten entstehen, mit der Public-Cloud nur schwer zu realisieren und mit höheren Kosten für die Datenübertragung verbunden.
- *A*5 **- Niedrige Kosten:** Die Berechnung der Kosten ist stark abhängig von der bereits vorhandenen IT-Infrastruktur. Wenn ein Unternehmen seine gesamte IT-Infrastruktur bereits in eine Public-Cloud verlagert hat, entstehen durch die Einführung von Edge-Computing neue Kosten für Hardware und Personal [MLB+11]. Szenarien, in denen keine hohen Datenmengen entstehen und daher keine hohen Kosten für die Datenübertragung entstehen, können mit einer Public-Cloud realisiert werden.
- *A*6 **- Sicherheit:**

  Da alle Kontextdaten in die Cloud übertragen werden, ergeben sich neue Sicherheitsrisiken und Unternehmensrichtlinien können das Senden sensibler Kontextdaten an die Cloud verbieten. Die Sicherheit der Daten kann daher nur mit zusätzlichen Sicherheitsmaßnahmen wie z. B. der Verschlüsselung der Kontextdaten gewährleistet werden, welche wiederum Auswirkungen auf Kosten und Latenzzeiten haben. Daher kann Anforderung *A*6 nur bedingt erfüllt werden.
- *A*7 **- Unternehmensübergreifende Situationserkennung:** Für eine unternehmensübergreifende Situationserkennung ist es notwendig, dass Kontextdaten verschiedener Unternehmen für die Situationser-

kennung verwendet werden können. Durch die Nutzung einer Public-Cloud, in diesem Kontext im Rahmen einer Community-Cloud, ist dies möglich und Anforderung *A7* kann erfüllt werden.

Wie zu sehen ist, werden die meisten Anforderungen durch den reinen Einsatz der Public-Cloud nicht erfüllt. Dennoch kann sie für Szenarien, in denen höhere Latenzzeiten akzeptabel sind, das Netzwerkverkehrvolumen niedrig ist, und keine sicherheitskritischen Kontextdaten übertragen werden, eingesetzt werden. Hierfür spricht vor allem das Kostenmodell einer Public-Cloud im Vergleich zu einer selbst zu verwaltenden IT-Infrastruktur.

Edge (Abbildung 6.5, rechts):
Der zweite Ansatz beschreibt die Ausführung der Situationserkennung ausschließlich auf der Edge. Dies ist grundsätzlich nur möglich, wenn alle benötigten Kontextdaten auf der Edge verfügbar sind. Somit ist eindeutig, dass das vorgestellte Szenario nicht vollständig realisierbar ist.

- *A3* **- Niedrige Latenz:** Yi et al. [YHQL15] zeigen, dass durch das Verschieben einer Anwendung auf die Edge die Latenz um bis zu 82 % verringert werden kann. Mit einer Latenz von 3 ms für die Ausführung der Situationserkennung [FHWM16] ist daher ein Einsatz in Niedriglatenz-Szenarien auf der Edge möglich.
- *A4* **- Niedriges Netzwerkverkehrvolumen:** Es werden keinerlei Kontextdaten über ein unternehmensexternes Netzwerk übertragen. Daher entsteht aufgrund der Situationserkennung kein Netzwerkverkehr und Anforderung *A4* ist erfüllt.
- *A5* **- Niedrige Kosten:** Floyer [Flo15] stellt ein Szenario einer Windanlage vor, bei der zusätzlich zur Verwendung einer Public-Cloud Edge-Computing eingesetzt wird und prognostiziert potentielle Kosteneinsparungen. Eine Einsparung von 95% des Netzwerkverkehrvolumenss führt zu einer Kostenreduzierung um etwa 64% und beinhaltet bereits die Kosten für die Hardware für das Edge-Computing. Für einen kosteneffektiven Einsatz von Edge-Computing musste die Datenmenge der Windanlage um mindestens 30% auf der Edge verringert

werden, um die Kosten für das Netzwerkverkehrvolumen einzusparen und damit die Gesamtkosten zu senken. Allerdings wurden IT-Personal und Management der Hardware nicht beachtet. Daher ist wieder zu bemerken, dass die Kosten stark abhängig vom gewählten Szenario sind und eine Einführung von Edge-Computing nicht unausweichlich zu erhöhten Kosten führt.

- *A6* **- Sicherheit:** Sicherheit bleibt für viele Unternehmen weiterhin ein kritischer Aspekt bei der Verwendung von Cloud-Computing. Da alle Kontextdaten und auch die Situationsobjekte auf der Edge und damit im eigenen Unternehmensnetzwerk verbleiben, haben die Unternehmen daher völlige Kontrolle über die eingesetzten Sicherheitsmaßnahmen und Anforderung *A6* kann erfüllt werden.
- *A7* **- Unternehmensübergreifende Situationserkennung:** Für eine unternehmensübergreifende Situationserkennung auf der Edge ist es notwendig, dass alle relevanten Kontextdaten auf derselben Edge verfügbar sind. Im Allgemeinen sind die Datenquellen verschiedener Unternehmen jedoch geographisch an verschiedenen Standorten. Im verwendeten Beispiel ist es daher nicht möglich, die Kontextdaten des LKWs zusammen mit den Kontextdaten der Maschinen bzw. der Produktionslinie gemeinsam zu verarbeiten. Anforderung *A7* wird somit nicht erfüllt.

Komplexe Szenarien, die globale Situationen erkennen müssen, können allein mit Edge-Computing aufgrund der geographisch verteilten Datenquellen nicht realisiert werden. Daher ist diese für die Erkennung lokaler Situationen mit hohen Anforderungen an die Latenz und Netzwerkverkehrvolumen geeignet. Vor allem in mobilen Umgebungen, z. B. einem LKW, mit hochvoluminösen Kontextdaten, ist die Verwendung von Edge-Computing vorteilhaft.

Hybrid (Abbildung 6.6): Weder allein mit der Public-Cloud noch mit Edge-Computing ist das vorgestellte Szenario in Anbetracht der Anforderungen *A3* – *A7* realisierbar. Der LKW produziert zu viele Daten für eine rein

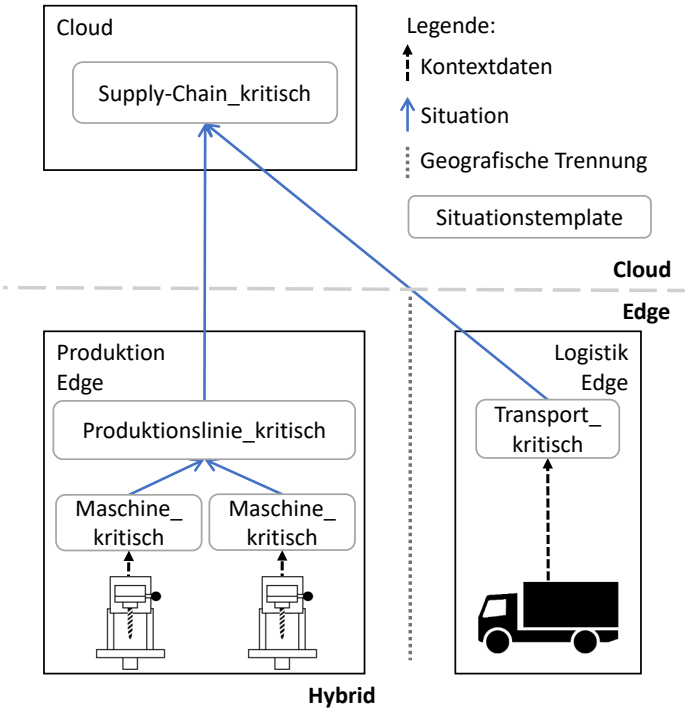

Abbildung 6.6: Verteilung der Situationserkennung auf Public-Cloud und Edge

Public-Cloud-basierte Situationserkennung und die geographische Verteilung der Datenquellen verhindert eine ausschließlich auf der Edge ausgeführte Situationserkennung. Daher wird zuletzt ein hybrider Ansatz gewählt, der sich beide zuvor vorgestellten Ansätze zunutze macht und die Situationserkennung auf die Public-Cloud und die Edge verteilt, um lokale Situationen auf der Edge und globale Situationen in der Cloud zu erkennen.

- *A*3 **- Niedrige Latenz:** Die Latenz für die Erkennung von lokalen Situationen auf der Edge kann wie beim vorherigen Ansatz stark verringert

werden, jene für globale Situationen ist jedoch weiterhin von der Netzwerklatenz beeinträchtigt. Daher kann Anforderung *A*3 nur bedingt erfüllt werden.

- *A*4 **- Niedriges Netzwerkverkehrvolumen:** Netzwerkverkehrvolumen kann eingespart werden, indem die Situationserkennung auf die Edge verschoben wird. Die resultierenden, lokalen Subsituationen müssen zur Erkennung der globalen Situationen dennoch an die Public-Cloud gesendet werden. Durch den Einsatz von Kontextentfernung beschränkt sich das Datenvolumen der Situationsobjekte jedoch auf das Minimum, um eine weiterführende Erkennung globaler Situationen zu ermöglichen. Somit ist Anforderung *A*4 erfüllt.
- *A*5 **- Niedrige Kosten:** Die potentiellen Kosteneinsparungen entsprechen denen, die auch bei der vorigen, Edge-basierten Situationserkennung vorgestellt wurden. Durch den Einsatz von Kontextentfernung wird das Datenvolumen lokaler Situationsobjekte und damit das Netzwerkverkehrvolumen verringert, wodurch Kosten eingespart werden können. Demnach ist auch hier Anforderung *A*5 erfüllt.
- *A*6 **- Sicherheit:** Durch die Erkennung lokaler Subsituationen können sensible Kontextdaten bereits auf der Edge verarbeitet werden. Durch den Einsatz von Kontextentfernung ist es dennoch möglich, diese Subsituationen zur Erkennung globaler Situationen einzusetzen, da keine sensiblen Daten innerhalb der Situationsobjekte in die Cloud gelangen. Anforderung *A*6 kann somit erfüllt werden.
- *A*7 **- Unternehmensübergreifende Situationserkennung:** Wie beim reinen Einsatz der Public-Cloud ist es möglich, eine kollaborative Situationserkennung zu ermöglichen. Ein großer Vorteil liegt auch hier im Einsatz der Kontextentfernung, da Unternehmen untereinander keine sensiblen Kontextdaten bereitstellen müssen. Jedes Unternehmen kann ausschließlich Subsituationen ohne jegliche Kontextdaten bereitstellen, die das Minimum an Informationen bereitstellen, um eine globale Situation zu erkennen. Daher ist Anforderung *A*7 erfüllt.

Tabelle 6.1: Übersicht der Erfüllung der ausführungsbezogenen Anforderungen $A3 - A7$ der verschiedenen Ausführungsumgebungen

	$A3$	$A4$	$A5$	$A6$	$A7$
Public-Cloud	X	X	(✓)	X	✓
Edge	✓	✓	(✓)	✓	X
Hybrid	(✓)	✓	✓	✓	✓

Bis auf die Niedriglatenz-Anforderung für globale Situationen werden alle Anforderungen durch den hybriden Ansatz erfüllt. Vor allem der Einsatz der Kontextentfernung ermöglicht viele Vorteile bei der Übertragung lokaler Subsituationen in die Cloud. Der hybride Ansatz ist daher für komplexe Szenarien mit mehreren Datenquellen geeignet, bei denen eine schnelle Reaktion auf lokale Subsituationen erforderlich ist und eine zentrale Situationserkennung für globale Situationen mit geringem Netzwerkverkehrvolumen benötigt wird. Mehrere Unternehmen können kollaborativ Situationen erkennen, ohne sensible Kontextdaten miteinander zu teilen, wenn die Kontextentfernung eingesetzt wird.

In Tabelle 6.1 werden die Ergebnisse der Analyse für die jeweiligen Verteilungsstrategien zusammengefasst. Da die Kosten stark abhängig von dem betrachteten Szenario sind, werden diese in Klammern dargestellt, da es prinzipiell möglich ist, geringe Kosten mit allen Verteilungsstrategien zu ermöglichen.

6.4 Ausführung einer verteilten Situationserkennung

In diesem Abschnitt wird beschrieben, wie eine verteilte Situationserkennung realisiert werden kann, beginnend mit der Modellierung eines Situationstemplates über die Ausführung und Verteilung der Situationserkennung bis zur Erkennung der modellierten Situation.

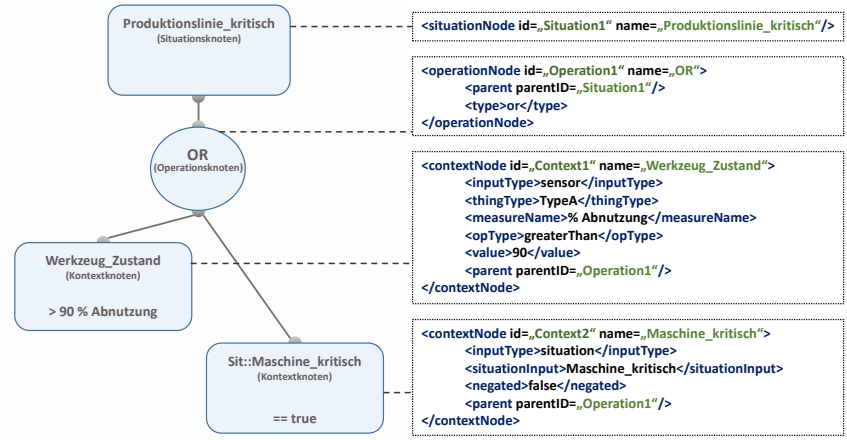

Abbildung 6.7: Modellierung eines Situationstemplates mit zugehörigen XML-Snippets

6.4.1 Tool-basierte Unterstützung der Modellierung

In Abschnitt 6.3.1 wurde die schichtenbasierte Modellierung von Situationstemplates vorgestellt, mit derer die Wiederverwendung bereits modellierter Situationstemplates und der Verteilung der Situationserkennung ermöglicht wurde. Wie beschrieben, sind Situationstemplates XML-Dateien, die konform zu einem vordefiniertem XML-Schema sind. Die Modellierung ohne Tool-Unterstüztung ist daher trotz der schichtenbasierten Modellierung mühsam und fehleranfällig. Hierfür wird das Situation-Template-Modeling-Tool (STMT) vorgestellt, das ein grafisches, webbasiertes Werkzeug zur Unterstützung der Modellierung von Situationstemplates darstellt. Auf Basis des XML-Schemas validiert das STMT modellierte Situationstemplates und gibt bei Falscheingaben bzw. fehlerhafter Modellierung Fehlermeldungen an den Nutzer zurück. In Abbildung 6.7 wird die Modellierung des Situationstemplates für die Situation *Produktionslinie_kritisch* dargestellt. Zu Illustrationszwecken wurde im Vergleich zur vorherigen Situation ein Situationsinput mit einem Sensorinput ersetzt, um deren syntaktische Unterschiede aufzuzeigen. Des Weiteren wurden in der grafischen Oberfläche

die Kontext- und Bedingungsknoten vereint, da per Schema zwischen der Menge der Kontext- und der Bedingungsknoten Bijektivität herrscht und somit eine einfachere Übersicht ermöglicht werden kann.

In Abbildung 6.7 (rechts) ist ein Ausschnitt des entsprechenden XML-Dokuments zur Definition der Knoten zu sehen. Jeder Knoten wird mit einer eindeutigen *ID* zur Identifikation innerhalb des Situationstemplates und einem *Namen* zur Anzeige für den Nutzer versehen. Darüber hinaus dient der Name des Situationsknotens gleichzeitig als Name der erkannten Situation bzw. des Situationsobjekts sowie des Situationstemplates. In allen restlichen Knoten gilt zusätzlich das Attribut *parentID*, der über die ID der Knoten auf den entsprechenden Elternknoten verweist. Operationsknoten enthalten einen *type*, der den logischen Operator angibt (OR, AND, XOR, NOR, NAND). Durch die schichtenbasierte Modellierung ist es möglich, Sensordaten sowie Situationen als Input zu verwenden. Dieser Unterschied wird über das Attribut *inputType* angegeben. Wird *sensor* ausgewählt, wird ein einzelner Sensor eines Assets verwendet. Situationstemplates werden nicht für spezielle Assets definiert, stattdessen werden sie für eine Baugruppe von Assets verwendet, um zu gewährleisten, dass die modellierten Sensoren bei den Assets vorhanden sind. Hierfür wurde das Attribut *thingType* hinzugefügt. Daher wird zunächst bei der Modellierung eines Kontextknotens vom Typ *sensor* der *thingType* ausgewählt. Hierfür besteht eine Verbindung zwischen dem STMT und einer Datenbank, in der die Baugruppen der Assets und die dazugehörigen Sensoren hinterlegt sind. Anschließend werden alle zur Verfügung stehenden Sensoren der Baugruppe angezeigt. Wird ein Sensor ausgewählt, wird der *measureName*, der den Messwert des Sensors beschreibt, automatisch ausgewählt. Über die Attribute *opType* und *value* werden der Vergleichsoperator ($>, \geq, <, \leq, ==$) und der Schwellenwert bzw. Vergleichswert definiert. Bei der Modellierung eines Situationsinputs werden stattdessen über den Datenbankzugriff ein Situationstemplate gewählt, dessen definierte Situation als Input für das neu modellierte Situationstemplate verwendet werden soll. Ein Vergleichsoperator wird nicht benötigt, da bei Situationsinputs nur der Wahrheitswert der Situation überprüft wird.

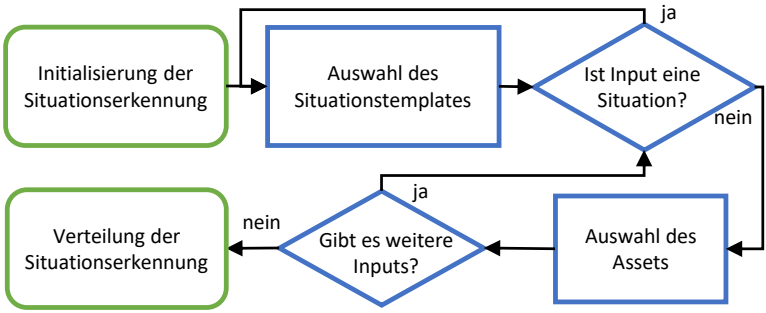

Abbildung 6.8: Flowchart zur Initialisierung der Situationserkennung

Lediglich wird ein *negated*-Attribut bereitgestellt, um den Wahrheitswert zu negieren. Das modellierte Situationstemplate wird auf Validität überprüft und kann anschließend in der Datenbank abgespeichert werden.

6.4.2 Initialisierung der Situationserkennung

Der nächste Schritt nach der Modellierung von Situationstemplates ist deren Initialisierung, um die Situationserkennung zu starten. Im ursprünglichen Ansatz von SitOPT wurden auf der Ebene der Kontextknoten ausschließlich Sensoren desselben Assets modelliert. Bei der Initialisierung wurde demnach das Asset ausgewählt und die Datenquellen der Sensoren angegeben. Durch die Veränderungen, die sich durch die Einführung der schichtenbasierten Modellierung ergeben haben, muss die Prozedur der Initialisierung entsprechend angepasst werden, da sowohl mehrere Assets als auch Situationen als Input verwendet werden können. Die einzelnen Schritte werden anhand des in Abbildung 6.8 dargestellten Flowcharts erläutert. Zuerst wird das zu initialisierende Situationstemplate ausgewählt. Für jeden Kontextknoten wird überprüft, ob dieser einen Sensor- oder Situationsinput darstellt. Im Fall eines Sensorinputs wird das gewünschte Asset und der entsprechende Sensor ausgewählt. Im Fall eines Situationsinputs wird das gewünschte Situationstemplate ausgewählt. Für dieses neu ausgewählte Situationstem-

plate wird diese Prozedur wiederholt. Das neue Situationstemplate kann wiederum einen Situationsinput enthalten, wodurch ein erneuter Aufruf dieser Prozedur stattfindet, welcher abgeschlossen wird, sobald alle auszuwählenden Kontextknoten Sensorinputs darstellen. Anschließend können die initialisierten Situationstemplates verteilt werden.

6.4.3 Verteilung der Situationserkennung auf Public-Cloud und Edge

Durch die schichtenbasierte Modellierung gibt es die Möglichkeit, auf zwei verschiedene Weisen zwei äquivalente Situationstemplates zu modellieren. In Abbildung 6.9 in der oberen Hälfte ist ein einziges Situationstemplates modelliert worden, das in sich abgeschlossen alle Details der Assets modelliert, wohingegen auf der unteren Hälfte die schichtenbasierte Modellierung angewandt wurde und das Situationstemplate *Maschine_kritisch* zwei Mal als Input für das Situationstemplates *Produktionslinie_kritisch* agiert. Auf der unteren Hälfte ist dadurch die Verteilung der einzelnen Situationstemplates direkt möglich[1]. Dennoch ist es in manchen Fällen einfacher und schneller, Situationstemplates in ihrer Gesamtheit wie in der oberen Hälfte in Abbildung 6.9 zu modellieren, solange die Komplexität nicht zu hoch ist, anstatt zwei separate Situationstemplates wie in der unteren Hälfte zu modellieren[2]. Um dennoch die Vorteile einer verteilten Situationserkennung zu erhalten, wird im Folgenden das automatische Aufteilen von Situationstemplates eingeführt. Daraufhin wird erläutert, wie die jeweiligen Situationen in der Public-Cloud und auf der Edge erkannt werden.

6.4.3.1 Automatisches Aufteilen von Situationstemplates

Bei dem Vorgang des automatischen Aufteilens wird das Situationstemplate auf der oberen Hälfte in Abbildung 6.9 nach der Instanziierung automatisch in die drei einzelnen Situationstemplates auf der unteren Hälfte aufgeteilt.

[1] Im Rahmen dieses Beispiels wird davon ausgegangen, dass die Assets in einer jeweils eigenen Edge liegen, da ansonsten die gesamte Situationserkennung auf der Edge ablaufen könnte und eine Verteilung auf Public-Cloud und Edge keinen Sinn ergeben würde.

[2] Obwohl drei Situationstemplates zu sehen sind, sind die unteren zwei durch dasselbe Situationstemplate beschrieben, welches in diesem Beispiel zweimal instanziiert würde.

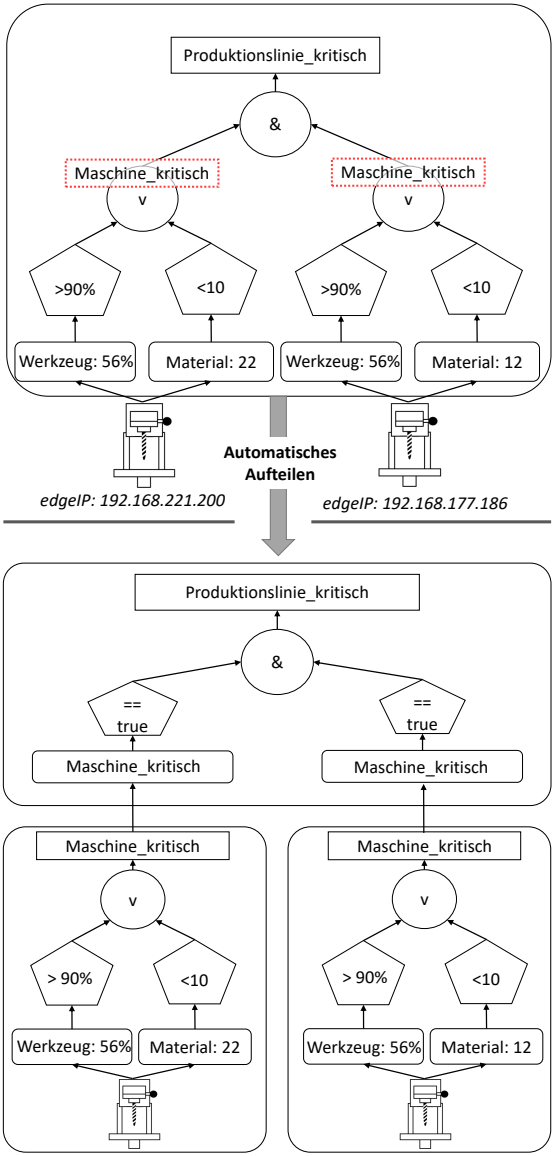

Abbildung 6.9: Verteilung der einzelnen Situationstemplates auf Public-Cloud und Edge

Hierbei wird erkannt, ob lokale Subsituationen extrahiert werden und in jeweils eigenen Situationstemplates modelliert werden können, welche auf der Edge instanziiert werden können. Diese *Erkennung* wird in Abbildung 6.9 durch das Label *Maschine_kritisch* (rot gepunktet umrandet) an den *OR*-Operationsknoten des oberen Situationstemplates angedeutet. Hierfür ist es erforderlich, dass zu den einzelnen Assets bekannt ist, ob und zur welchen Edge sie gehören. Diese Metadaten müssen in einer Datenbank für jedes Asset hinterlegt sein. Die *edgeIP* gibt die IP-Adresse der zugeordneten Rechenressource an, auf welcher eine Situationserkennung ausgeführt werden kann und wohin die Kontextdaten des Assets geschickt werden. Ist kein Wert für die *edgeIP* vorhanden, ist die Ausführung der Situationserkennung auf der Edge für dieses Asset nicht möglich. Ob eine *edgeIP* für ein Asset existiert und wie diese lautet, ist in jedem Kontextknoten zum Zeitpunkt der Instanziierung enthalten, da die Auswahl der Assets bereits stattgefunden hat. Mittels einer Post-Order-Traversierung werden ausgehend von Kontextknoten die Werte der jeweiligen *edgeIPs* an den Operationsknoten miteinander verglichen. Stimmen diese überein, wird der Wert bei dem Operationsknoten hinterlegt, das Situationstemplate weiter traversiert und der Operationsknoten mit anderen Operationsknoten verglichen. Ist bei dem Operationsknoten unmittelbar vor dem Situationsknoten eine *edgeIP* hinterlegt, kann das gesamte Situationstemplate auf der Edge instanziiert werden. Falls nur im Situationstemplate tiefer gelegene Operationsknoten eine *edgeIP* hinterlegt haben, kann an dieser Stelle ein neues Situationstemplate erstellt werden, das diesen Operationsknoten als obersten Operationsknoten unmittelbar vor dem Situationsknoten enthält. Ist an keinem Operationsknoten eine *edgeIP* hinterlegt, ist eine Aufteilung der Situationserkennung nicht möglich.

6.4.3.2 Verteilung auf Public-Cloud und Edge

Entweder wurde das Situationstemplate von Beginn an mit der schichtenbasierten Modellierung modelliert oder mithilfe des *automatischen Aufteilens* nach der Instanziierung in separate Situationstemplates aufgeteilt. Diese einzelnen Situationstemplates werden auf die Public-Cloud und Edge verteilt

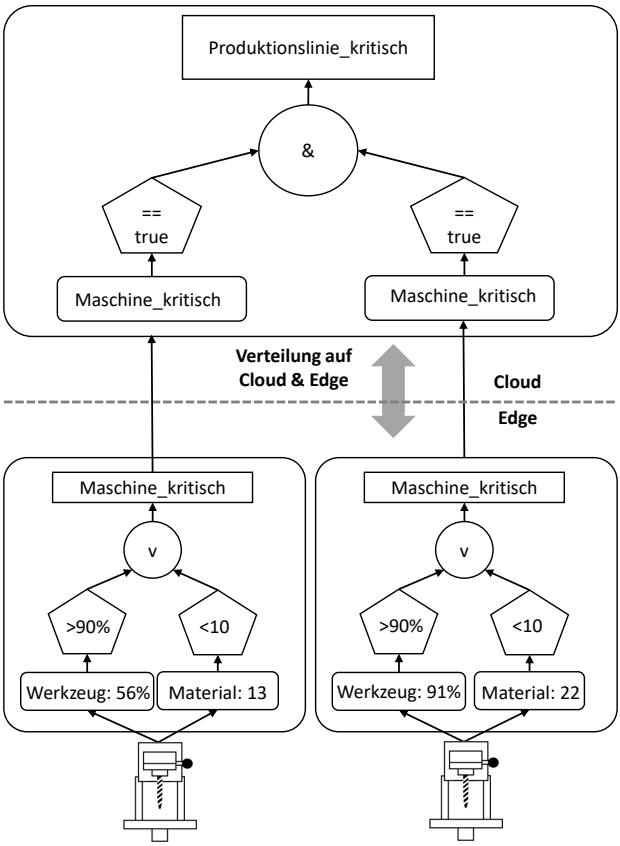

Abbildung 6.10: Verteilung der einzelnen Situationstemplates auf Public-Cloud und Edge

(s. Abbildung 6.10). Wenn der auf der höchsten Ebene gelegene Operationsknoten eine *edgeIP* enthält, kann die Situationserkennung für dieses Situationstemplate auf der Edge durchgeführt werden, anderenfalls in der Public-Cloud. Jede Umgebung, auf der eine Situationserkennung ausgeführt

werden soll, enthält ein System für die Situationserkennung (z. B. eine CEP-Engine) und eine Publish- und Subscribe-basierte nachrichtenorientierte Middleware (z. B. einen MQTT-basierten Broker).

Zur Laufzeit werden die Kontextdaten der Assets zunächst an den Message-Broker auf der Edge gesendet. Auf diese Weise können die Daten neben der Situationserkennung für andere Zwecken genutzt werden. Das System, das für die Situationserkennung genutzt wird, abonniert die Kontextdaten beim Message-Broker und erhält diese, sobald sie bei ihm eintreffen. Die bei der Situationserkennung entstandenen Situationsobjekte werden wiederum an den Message-Broker geschickt, sodass diese als Subsituationen bei weiteren Situationserkennungen bzw. in situationssensitiven Anwendungen verwendet werden können. Zusätzlich werden die Situationsobjekte aller erkannten Situationen an den in der Public-Cloud liegende Message-Broker gesendet, damit lokale Subsituationen zur Erkennung globaler Situationen verwendet werden können.

6.5 Implementierung der Konzepte und Evaluation

6.5.1 Prototypische Implementierung

Die Konzepte dieses Kapitels wurden prototypisch implementiert und sind auf GitHub verfügbar[1]. Zur Modellierung der Situationstemplates wurde ein in JavaScript geschriebenes Modellierungswerkzeug entwickelt. Modellierte Situationstemplates werden auf Konformität zum XML-Schema überprüft und können anschließend in der NoSQL-Datenbank CouchDB[2] gespeichert und bei Bedarf wieder geladen und angepasst werden. In CouchDB werden neben den Situationstemplates alle relevanten Informationen für die Situationserkennung verwaltet. Dies umfasst alle benötigten Informationen zu den industriellen Assets, z. B. die *edgeIP* und deren Sensoren, als auch die erkannten Situationen in Form von Situationsobjekten. Eine in Java entwickelte Komponente ist für die im Konzept beschriebenen Prozesse wie

[1] https://github.com/Sitopt
[2] https://couchdb.apache.org/

Tabelle 6.2: Ergebnisse der Evaluation

#Kontext	Netzwerkverkehrvolumen (kB/s)	Kontextanteil(%)
0	0,26	-
10	0,4	35
50	1,22	78
200	4,39	94
1000	21,1	98,8

das automatische Aufteilen von Situationstemplates sowie die Verteilung der Situationserkennung zuständig. Erkannte Situationen werden an den MQTT-basierten Message-Broker Mosquitto [1] gesendet.

6.5.2 Evaluation

Im Folgenden werden die Ergebnisse einer quantitativen Evaluation in Bezug auf Anforderungen *A*3 bzgl. Latenz und *A*4 bzgl. Netzwerkverkehrvolumen erläutert.

Die Laufzeit einer einzelnen Situationserkennung entspricht weiterhin der des ursprünglichen Ansatzes von SitOPT. Daher gelten weiterhin Laufzeiten von durchschnittlich 3 ms [FHWM16]. Somit ist durch die Ausführung der Situationserkennung auf der Edge und einer daraus resultierenden Minimierung der Netzwerklatenz für die Übertragung der Kontextdaten eine sehr geringe Latenz für die Erkennung lokaler Situationen ermöglicht worden. Die Erkennung globaler Situationen unterliegt weiterhin einer entsprechend hohen Netzwerklatenz.

Für die Messungen des Netzwerkverkehrvolumens wurden einmal pro Sekunde Situationsobjekte erstellt und über ein Netzwerk übertragen. Die Ergebnisse werden in Tabelle 6.2 angezeigt. Situationsobjekte enthalten den Kontext, der zur Erkennung beigetragen hat. Daher wurden Situationsobjekte mit verschiedener Anzahl von Kontext erstellt (#Kontext). Kontext stellt hierbei einzelne numerische Werte dar. Wie zu erwarten, wächst mit stei-

[1] https://mosquitto.org/

gender Anzahl an Kontext das Netzwerkverkehrvolumen. Zuletzt wird in der letzten Spalte der prozentuale Anteil der Kontextdaten an der Gesamtgröße eines Situationsobjekts betrachtet.

Wie zu sehen ist, stellen die Kontextdaten ab einer Anzahl von 200 bereits mehr als 94 % der Gesamtgröße eines Situationsobjekts und damit den Großteil des Netzwerkverkehrvolumens dar. Diese Größenordnung wird im Rahmen von I4.0 schnell erreicht, da industrielle Assets mit einer Vielzahl von Sensoren ausgestattet bzw. erweitert werden, um bspw. Szenarien wie den Digitalen Zwilling zu ermöglichen, der eine digitale Kopie eines industriellen Assets darstellt. Vor allem bei Situationen, die aus den Kontextdaten bzw. Situationen mehrerer industrieller Assets resultieren, kann es daher sinnvoll sein, eine Kontextentfernung durchzuführen. Wie in Abschnitt 6.3.2.2 beschrieben, können Kontextdaten lokaler Subsituationen vor der Übertragung in die Public-Cloud entfernt werden. Diese Subsituationen können dennoch zur Erkennung globaler Situationen verwendet werden.

6.6 Zusammenfassung

In diesem Kapitel wurde die verteilte Situationserkennung mittels Situationstemplates vorgestellt. Auf Basis des Forschungsprojekts SitOPT wurde die Situationserkennung dahingehend erweitert, um weitere Anforderungen aus Industrie 4.0 zu berücksichtigen. Zunächst wurden daher die Anforderungen erfasst, die durch den anfänglichen Ansatz nicht erfüllt werden konnten.

Auf Basis der Anforderungen wurde die Modellierung der Situationstemplates angepasst. Es wurde ermöglicht, mehrere Assets sowie Situationen als Input zu modellieren. Die daraus resultierende, höhere Modellierungskomplexität wurde durch eine schichtenbasierte Modellierung von Situationstemplates und ein grafisches Modellierungswerkzeug wiederum reduziert.

Es wurden verschiedene Ausführungsumgebungen analysiert, auf denen eine Situationserkennung ausgeführt werden kann und wie die erfassten Anforderungen dadurch erfüllt werden können.

Zuletzt wurden Details zur Ausführung erläutert. Automatisches Aufteilen erlaubt die Vorteile einer verteilten Situationserkennung, auch wenn keine schichtenbasierte Modellierung verwendet wird. Durch Kontextentfernung werden Situationsobjekte auf das Nötigste reduziert, um bei der Übermittlung von der Edge in die Public-Cloud das Netzwerkverkehrvolumen zu verringern und dennoch als Subsituationen zur Erkennung globaler Situationen dienen zu können.

Kapitel 7

Kontextmodell für das Monitoring industrieller Cloud-Umgebungen

In den bisherigen Kapiteln wurden verschiedene Ansätze für das Monitoring in den Domänen Cloud-Computing und I4.0 erläutert. In der Regel existiert in Unternehmen kein einzelnes System, das für das Monitoring einer Cloud-Umgebung verwendet wird [NKFW19] und Studien zeigen, dass viele Unternehmen mehr als zehn Monitoring-Systeme parallel betreiben [Big19].

Hinzu kommen im Rahmen dieser Arbeit Systeme für das Monitoring der industriellen Assets, z. B. eine Situationserkennung, sowie weitere Management-Systeme, die wertvolle und relevante Daten für das Verwalten einer industriellen Cloud-Umgebung besitzen. In Geräte-Management-Systemen sind bspw. Details zu den industriellen Assets hinterlegt. Cloud-Management-Systeme enthalten statische Informationen zu den VMs wie das Erstellungsdatum oder Event-Informationen, wie z. B. die Anweisung

zum Abschalten einer VM. Service-Level-Agreement (SLA)-Management-Systeme enthalten die vertraglich vereinbarten Regelungen für gebuchte Cloud-Ressourcen bei Drittanbietern.

Die Summe all dieser Systeme zum Monitoring und Management einer industriellen Cloud-Umgebung führt zu einer Vielzahl von Datensilos, die von verschiedenen Abteilungen verwaltet werden, unter denen oftmals wenig bis gar keine Kommunikation stattfindet [NKFW19]. Wie und ob diese Daten miteinander in Beziehung zueinander stehen, ist daher oftmals unbekannt.

Die Definition, welcher Kontext für eine überwachte Ressource und die weitere Analyse erforderlich ist, woher die Kontextdaten stammen und wie diese in Beziehung zu anderen Kontextdaten stehen, ist jedoch wichtig, um einen umfassenden Überblick über die industrielle Cloud-Umgebung zu ermöglichen. Beispielsweise wird relevanter Kontext über eine VM von einem ITIM-System (z. B. CPU-Last), der Cloud-Management-Plattform (z. B. IP und Erstellungsdatum) und SLA-Management-Systemen (z. B. welche Mindestverfügbarkeit wurde vereinbart?) geliefert. Für DEAR ist bspw. eine Topologie der überwachten Cloud-Umgebung von Vorteil, auf deren Basis die Verteilung der Alerting-Regeln erfolgen kann (vgl. Kapitel 5, Abschnitt 5.3.1.2). Für die verteilte Situationserkennung hingegen sind detaillierte Informationen über existierende Edge-Clouds obligatorisch, wie z. B. die IP-Adressen der jeweiligen VMs und welche industriellen Assets ihre Sensorwerte dorthin übermitteln (vgl. Kapitel 6, Abschnitt 6.4.3.1).

Um diese Informationen zu modellieren, kommen in der Regel Kontextmodelle zum Einsatz, die alle relevanten Entitäten und deren Beziehungen zueinander beschreiben. Da bisher kein Kontextmodell existiert, welches das Monitoring sowohl in den Domänen Cloud-Computing als auch I4.0 betrachtet, wird in diesem Kapitel daher ein Kontextmodell für das Monitoring industrieller Cloud-Umgebungen vorgestellt. Dieses Kontextmodell soll (i) die zu überwachenden Ressourcen und deren Beziehungen zueinander, (ii) die relevanten Monitoring- und Management-Daten für diese Ressourcen, und (iii) die weiteren Verarbeitungsschritte der Monitoring-Daten darstellen.

Dieses Kapitel ist eine überarbeitete und zusammenfassende Version mehrerer Publikationen des Autors [MHWM17; MS20].

Das restliche Kapitel ist wie folgt gegliedert: in Abschnitt 7.1 werden zunächst verwandte Arbeiten vorgestellt. Daraufhin werden in Abschnitt 7.2 die Anforderungen an das Kontextmodell definiert. In Abschnitt 7.3 wird das Kontextmodell vorgestellt. In Abschnitt 7.4 werden die Ergebnisse in Bezug auf die erfassten Anforderungen diskutiert. Abschnitt 7.5 enthält die Zusammenfassung dieses Kapitels.

7.1 Verwandte Arbeiten

Für die Darstellung von Kontextmodellen kommen oftmals Ontologien zum Einsatz, da sie folgende Vorteile bieten: (i) einheitliche Darstellung von Wissen, (ii) automatisches Reasoning, d. h. logische Schlussfolgerungen von höherwertigen Informationen, und (iii) Wiederverwendung von Wissen durch die Nutzung bereits bestehender Ontologien [WZGP04].

Im Bereich der verarbeitenden Industrie existieren eine Vielzahl von Ontologien, die verschiedene Teilbereiche wie z. B. Stahlproduktion oder Ölindustrie, spezielle Prozesse wie z. B. Produktentwicklung und -wartung, oder I4.0 im Allgemeinen darstellen [DGB08; GSZ18; GW12; HBPH14; ZZL17]. Da die Vision von I4.0 in Richtung Smart Factories geht, die autonom mithilfe von intelligenten Robotern und automatischen Prozessen funktionieren, wurde im Bereich der Robotik dahingehend der IEEE Standard *IEEE 1872-2015* definiert, der eine Basisontologie für den Umgang mit Robotern und entsprechendem Vokabular definiert [SPG+15]. Mithilfe dieser Ontologien können somit zahlreiche Szenarien wie z. B. Rapid-Prototyping oder auf autonomen Drohnen basierte Lieferung unterstützt werden [KKF+19].

Im Bereich des IoT existieren ebenso eine Vielzahl an Kontextmodellen, um verschiedene Aspekte einer IoT-Umgebung zu beschreiben. Beispielsweise wurde mit SensorML, einem Standard des Open Geospatial Consortiums [1], ein Modell entworfen, um Sensoren, deren Messwerte und weiterverarbeitende Prozesse zu beschreiben [OGC21]. Damit soll eine Interoperabilität ermöglicht werden, die es erleichtert, bspw. Sensoren und deren Messwerte

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

automatisiert in einem Workflow zu verwenden. Hirmer et al. [HWBM16b] adaptieren SensorML, um die Registrierung und Anbindung von Sensoren und die Bereitstellung der Sensordaten zu vereinfachen. Franco da Silva und Hirmer vergleichen Ontologien und Modelle im IoT-Kontext [FH20]. Dabei zeigt sich, dass die gängigen Ontologien keine Unterstützung für die Aspekte des IT-Monitoring enthalten.

Im IT-Monitoring existieren Ontologien zum Management von Ereignissen [DSW+18] oder für das Monitoring spezieller Applikationen, die kein gewöhnliches Monitoring unterstützen [FJJG13]. Eine dem IT-Monitoring ähnliche Domäne ist eHealth, in der statt virtueller Maschinen oder Applikationen Patienten überwacht werden und auf Basis der gesammelten Monitoring-Daten, wie z. B. Puls oder Sauerstoffzufuhr, im Notfall Alerts an das Krankenhauspersonal gesendet werden können. Es existieren mehrere Ontologien [ATA+10; GFH+13; PG07], die mittels kontextbezogener Anwendungen Prozesse automatisieren und verbessern sollen.

Dennoch liegen keine Ontologien oder andere Formen von Kontextmodellen vor, die domänenübergreifend sowohl das Monitoring in der Domäne Cloud-Computing als auch in der Domäne I4.0 gemeinsam erfassen und somit als Grundlage für das Monitoring industrieller Cloud-Umgebungen dienen könnten.

7.2 Anforderungen

In Abbildung 7.1 ist eine industrielle Cloud-Umgebung zu sehen, die aus einer Multi-Cloud, d. h., mehrere Public- und Private- sowie Edge-Clouds, und einer industriellen Umgebung besteht. Multi-Clouds sind nützlich, um vor Datenverlusten im Falle eines Ausfalls einzelner Cloud-Anbieter geschützt zu sein, um Kosten zu optimieren, und um die Quality of Service (QoS) zu erhöhen [Pet13]. Umfragen zufolge soll dieser Ansatz von 93 % der Unternehmen eingesetzt werden [Wei20]. In jeder Cloud werden mehrere virtuelle Maschinen (VMs) bereitgestellt, auf denen wiederum verschiedene

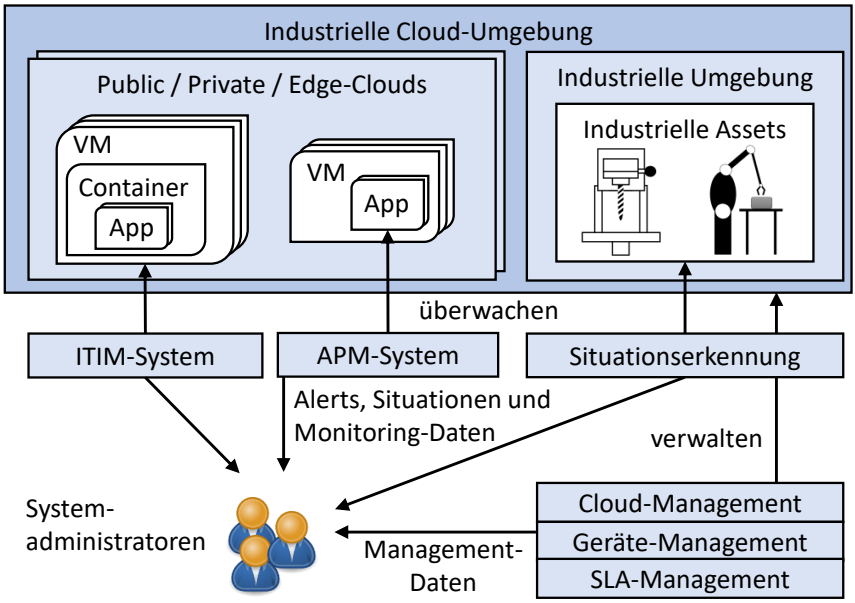

Abbildung 7.1: Abhängigkeiten innerhalb einer industriellen Cloud-Umgebung

Anwendungen bereitgestellt werden, die zum Teil in Containern vorliegen. Gleichermaßen sind in der industriellen Umgebung verschiedene industrielle Assets zu finden.

Weitere Management-Systeme zur Verwaltung der VMs auf Private- und Edge-Clouds sind bspw. Cloud-Management-Systeme, die statische Informationen, wie z. B. zugehörige IP-Adressen und Firewallregeln, enthalten. Für industrielle Assets werden Geräte-Management-Systeme verwendet, um Informationen über deren Sensoren, Datenmodelle und Lokation zu erhalten. Zuletzt werden SLAs bezüglich der QoS von Public-Clouds, wie z. B. die vertraglich vereinbarte Verfügbarkeit der Cloud-Ressourcen, in SLA-Management-Systemen verwaltet.

Letztlich werden die VMs, die darauf installierten Anwendungen, deren Container und auch die industriellen Assets mittels mehrerer, individuel-

ler Systeme fortlaufend überwacht. Dazu gehören etwa IT-Infrastruktur-Monitoring (ITIM)-Systeme, Application-Performance-Monitoring (APM)-Systeme, sowie spezielle Systeme wie die Situationserkennung.

Tritt nun ein einzelner Fehler in der Umgebung auf, kann dieser weitreichende Auswirkungen haben. So kann bspw. ein Ausfall einer VM zum Ausfall der darauf bereitgestellten Container und Anwendungen führen. Eine darauf bereitgestellte Situationserkennung fällt demnach ebenso aus und gefährdet auch das Monitoring der industriellen Assets. Diese Ausfälle werden somit von vielen verschiedenen Monitoring-Systemen, wie z. B. ITIM-, APM - , Hypervisor- und Container-Monitoring-Systemen, erkannt. Jedes dieser Systeme erzeugt Alerts, welche die Aufmerksamkeit der Systemadministratoren fordern. Zur Problembehebung müssen diese je nach Problemfall auf Cloud- und Geräte-Management-Systeme zugreifen oder die vereinbarten SLAs überprüfen, wenn gebuchte Cloud-Ressourcen eines Drittanbieters die Fehlerursache sind.

Obwohl diese Daten alle verfügbar sind, ist es erforderlich, die Beziehungen aller Entitäten einer industriellen Cloud-Umgebung zueinander zu definieren, um ein effizienteres Monitoring zu ermöglichen. Beispielsweise können auf diese Weise bei Ausfall einer VM alle Alerts der kaskadierenden Fehler, d. h. der Ausfall aller darauf bereitgestellten Anwendungen und Container, zusammengefasst werden, um die Anzahl der Alerts und somit den Zeitaufwand für die Systemadministratoren zu reduzieren. Hierfür gilt das in dieser Arbeit vorgestellt Kontextmodell selbstverständlich nur als Grundlage, um darauf aufbauend ein geeignetes Reasoning zu ermöglichen, welches es bspw. für jeden Alert nachvollziehbar macht, welche Analyse mit welchen Daten aus welchen Systemen verwendet wurde und welche überwachten Ressourcen davon betroffen sind.

Es wird ersichtlich, dass ein Kontextmodell sowohl domänenübergreifend (Cloud-Computing und I4.0) als auch systemübergreifend (Monitoring- und Management-Systeme) sein muss, um eine umfassende Sicht auf eine industrielle Cloud-Umgebung zu erlauben. Zusätzlich müssen die jeweiligen Monitoring- und Management-Daten sowie Alerts modelliert werden, um deren Beziehungen zueinander darstellen zu können.

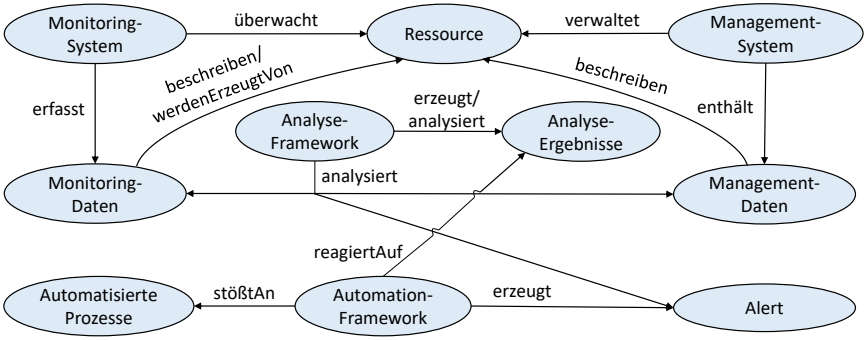

Abbildung 7.2: Übersicht der Ontologie

7.3 Kontextmodell für Monitoring industrieller Cloud-Umgebungen

In diesem Abschnitt wird das Kontextmodell für das Monitoring industrieller Cloud-Umgebungen vorgestellt, um die grundlegenden Entitäten und deren Beziehungen zueinander darzustellen, die für das Monitoring industrieller Cloud-Umgebungen eine Rolle spielen. Dieses wird als Ontologie dargestellt, zu dessen Formalisierung das Resource-Description-Framework-Schema (RDFS) [BGM14] verwendet werden kann. In Abbildung 7.2 ist die Übersicht der Ontologie zu sehen.

Die *Ressource* stellt eine Ressource innerhalb der industriellen Cloud-Umgebung dar, die von einem *Monitoring-System überwacht* und von einem *Management-System verwaltet* wird. Hierfür *erfassen* bzw. *enthalten* die jeweiligen Systeme *Monitoring-Daten* bzw. *Management-Daten*, welche die überwachten *Ressourcen beschreiben* bzw. von ihnen *erzeugt werden*, wenn es sich bei den Ressourcen um Sensoren handelt. Ein *Analyse-Framework* wird genutzt, um die *Monitoring-*, *Management-Daten*, und *Alerts* zu *analysieren* und *erzeugt* daraufhin *Analyseergebnisse*. Die *Analyseergebnisse* können wiederum einer Analyse unterzogen werden. Ein *Automation-Framework erhält* alle *Analyseergebnisse* und *erzeugt* daraufhin *Alerts* oder *stößt auto-*

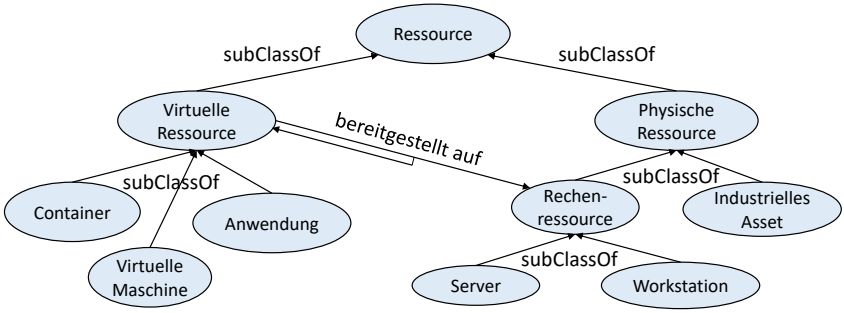

Abbildung 7.3: Detailansicht – Ressource

matisierte Prozesse zur automatischen Problembehebung an. *Alerts* können wiederum vom *Analyse-Framework analysiert* werden und können zu neuen *Analyse-Ergebnissen* führen.

Im Folgenden werden die einzelnen Teilbereiche dieser Ontologie im Detail betrachtet. Es ist zu beachten, dass mit diesen Ontologien kein Anspruch auf Vollständigkeit erhoben wird. Stattdessen werden grundlegende Eigenschaften erfasst, die als sinnvoll oder notwendig erachtet werden und den zugrundeliegenden Nutzen eines Kontextmodells aufzeigen sollen.

7.3.1 Ressourcen

Ressourcen stellen die zu überwachenden Ressourcen in der industriellen Cloud-Umgebung dar. Durch Erstellung von Unterklassen (in RDFS mittels *subClassOf*, vgl. Abbildung 7.3) können Ressourcen in virtuelle oder physische Ressourcen unterteilt werden, welche wiederum in weitere Unterklas-

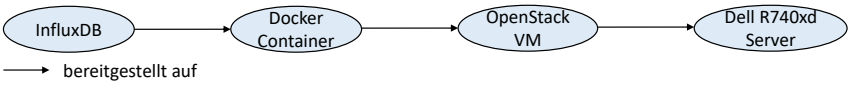

Abbildung 7.4: Instanzen der Klasse *Ressource*

sen unterteilt werden können. Virtuelle Ressourcen können auf physischen Rechenressourcen, wie z. B. einem Server, oder auf weiteren virtuellen Ressourcen bereitgestellt werden. Dies ist eine Möglichkeit zur Beschreibung der Topologie einer industriellen Cloud-Umgebung, die Abhängigkeiten der einzelnen Ressourcen zueinander zu repräsentieren (vgl. Abbildung 7.4). Beim Ausfall des *Dell R740xd Servers* würden die jeweiligen Monitoring-Systeme für den Server als auch für jede darauf bereitgestellte Ressource einen individuellen Alert erzeugen, obwohl die Ursache ausschließlich beim Server liegt. Diese kaskadierenden Fehler erhöhen maßgeblich die Anzahl von Alerts. Mithilfe dieser Topologie ist es einerseits möglich, die Fehlerursache zu finden bzw. einzugrenzen und andererseits kann sie zur Reduktion von Alerts beitragen, indem nur Alerts für die Ressource erzeugt werden, die als Fehlerursache identifiziert wurde.

Ergänzend können weitere Beziehungen modelliert werden, um weitere Anwendungsfälle effizienter zu behandeln. Beispielsweise können die Beziehungen industrieller Assets zueinander innerhalb einer Produktionslinie beschrieben werden. Beim Ausfall einer einzelnen Produktionsmaschine können mittels automatisierter Prozesse die Produktion entsprechend angepasst werden. Dazu kann z. B. der Workflow der Produktion angepasst werden [WSBL15].

Zur weiteren Unterteilung der Unterklassen können bereits existierende Ontologien verwendet werden. Beispielsweise kann bei industriellen Assets die Subklasse *Robotik* eingeführt werden, welche durch die Ontologie *IEEE 1872-2015* (s. Abschnitt 7.1) bereitgestellt wird.

7.3.2 Monitoring- und Management-Systeme

Die Monitoring- und Management-Systeme sind dafür verantwortlich, die aktuellen Zustände der Ressourcen zu erfassen und deren statische Informationen zu verwalten. Unterklassen der Monitoring-Systeme sind ITIM-, APM-, Netzwerk-Monitoring-, Hypervisor-Monitoring-Systeme und weitere spezialisierte Systeme für das Monitoring einer Cloud-Umgebung. Ebenso werden

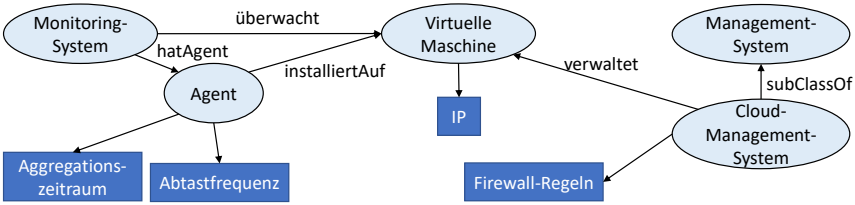

Abbildung 7.5: Detailansicht – Monitoring- und Management-Systeme

die Agenten der Monitoring-Systeme modelliert sowie deren Konfigurationseigenschaften, wie z. B. die Abtastfrequenz und der Aggregationszeitraum (vgl. Abbildung 7.5).

Für das Management der industriellen Cloud-Umgebung kommen zusätzlich Management-Systeme zur Verwaltung der jeweiligen Clouds, physischen Assets, und SLAs zum Einsatz. Beispielsweise werden die statischen Informationen der überwachten Ressourcen (vgl. Abbildung 7.5, *virtuelle Maschine*), wie z. B. die IP, über ein Cloud-Management-System verwaltet. Des Weiteren werden auch Informationen verwaltet, die sich auf mehrere Ressourcen beziehen können (z. B. Firewall-Regeln).

Da ein Ausfall der Monitoring- und Management-Systeme oder derer Agenten zu einer Beeinträchtigung des Monitoring der industriellen Cloud-Umgebung führen würde, können all diese Systeme wiederum als Ressourcen betrachtet werden, die ebenso überwacht werden müssen. Monitoring- und Management-Systeme würden daher als Subklassen der Subklasse *Anwendungen* in Abbildung 7.3 modelliert werden, was in Abbildung 7.2 zur besseren Übersichtlichkeit nicht in dieser Detailstufe dargestellt wurde.

7.3.3 Monitoring- und Management-Daten

Monitoring- und Management-Daten werden von Monitoring- und Management-Systemen erfasst bzw. verwaltet und beschreiben die überwachten Ressourcen. In Abbildung 7.6 ist zu sehen, dass die Monitoring-Daten sehr heterogen sind und bspw. numerische Sensordaten in Form von

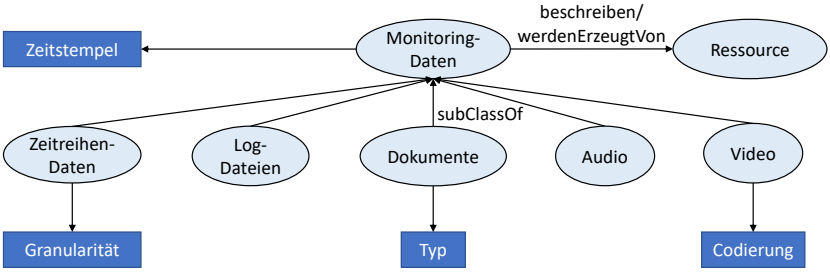

Abbildung 7.6: Detailansicht – Monitoring-Daten

Zeitreihendaten oder alphanumerischen Daten in Form von Log-Dateien oder Dokumenten darstellen. Darüber hinaus werden zukünftig weitere Datenquellen wie Audio und Video für das Monitoring verwendet, um z. B. Probleme in der Produktion mittels einer Anomaliedetektion in Audiodaten zu erkennen. Analog dazu sind Management-Daten ebenso unterteilt und werden z. B. in Form relationaler Daten oder Dokumente verwaltet.

Monitoring-Daten ohne Kontext, wie z. B. der Wert *74*, sind nicht aussagekräftig. Daher muss immer eine Verknüpfung zu der Ressource hergestellt werden, die überwacht wird bzw. die den Wert erzeugt. Auf diese Weise werden dem Wert mehr Informationen hinzugefügt, bspw. *74 °C für die CPU des Dell R740xd Servers*. Damit können bswp. können Alerting-Regeln unter Beachtung üblicher Temperaturen einer CPU modelliert werden.

Während Management-Daten in der Regel über einen längeren Zeitraum gültig sind, wie z. B. die IP einer VM, sind Monitoring-Daten üblicherweise nur für einen begrenzten Zeitraum gültig und relevant. Daher ist eine wichtige, gemeinsame Eigenschaft aller Monitoring-Daten ein Zeitstempel, um die Aktualität der Daten zu erfassen. Während Zeitstempel den Zeitreihendaten inhärent sind, müssen diese z. B. bei Log-Dateien und Dokumenten ausdrücklich erfasst oder spätestens bei der Speicherung der Daten nachträglich hinzugefügt werden. So können Systemadministratoren die Aktualität der Monitoring-Daten überprüfen und darauf basierend Entscheidungen treffen.

Des Weiteren können die individuellen Arten von Monitoring-Daten durch ihre Qualität und weitere Metadaten beschrieben werden. Zeitreihendaten können bspw. durch eine Granularität beschrieben werden, die sich aus der Abtastfrequenz und dem Aggregationszeitraum des überwachenden Agenten erschließt. Weiterhin können Dokumente durch ihren Typ (z. B. JSON) und Videos durch ihre Codierung (z. B. .mpeg) beschrieben werden.

Analog zu Management-Daten können auch Monitoring-Daten für mehrere Ressourcen relevant sein (vgl. *Firewall-Regeln* in Abbildung 7.5). So kann bspw. eine gesamte Produktionslinie bestehend aus mehreren einzelnen industriellen Assets mittels Audio- oder Videoaufnahmen gleichzeitig überwacht werden.

7.3.4 Analyse-Framework und -Ergebnisse

Auf Basis der im vorigen Abschnitt beschriebenen Monitoring- und Management-Daten ermöglicht ein Analyse-Framework eine Analyse und stellt als Resultat Analyseergebnisse bereit. Aufgrund der heterogenen Daten muss das Analyse-Framework auch eine Sammlung an verschiedenen Methoden und Systemen zur Analyse dieser Daten umfassen. Für zeitkritische Anwendungsfälle müssen Streaming-Daten mit möglichst geringer Verzögerung analysiert werden können und zur Analyse historischer Daten muss eine Batch-Verarbeitung möglich sein. Ebenso müssen verschiedene Verfahren zur Datenanalyse, wie z. B. Wenn-Dann-Regeln, neuronale Netze, und Clustering zum Einsatz kommen.

Daher zählt auch die Situationserkennung mit SitOPT als regelbasiertes Analyse-Framework. Wie bereits in Kapitel 6 beschrieben, wird aus einer erkannten Situation heraus ein Situationsobjekt erstellt. Als Grundlage des Situationsobjekts wurde in früheren Arbeiten [MHWM17] ein Situationsmodell definiert, um eine Schnittstelle zwischen der Situationserkennung und situationsbezogenen Anwendungen zu ermöglichen. Anstatt die Situationserkennung direkt in die Anwendungen zu integrieren, wurde mit dem Situationsmodell eine Abkopplung ermöglicht, die eine unabhängige Entwicklung und Ausführung der Anwendungen und der Situationserkennung

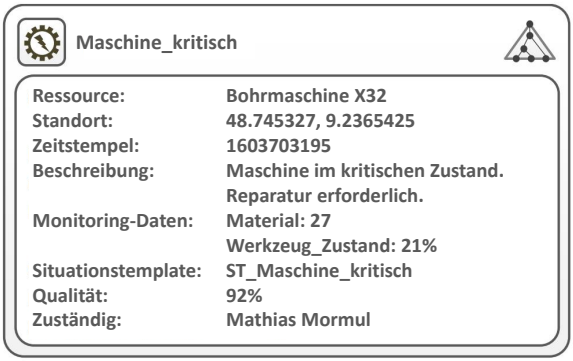

Abbildung 7.7: Situationsobjekt auf Basis des Kontextmodells (angelehnt an [WSBL15])

ermöglicht. Zusätzlich wurde es damit möglich, erkannte Situationen für mehrere, unterschiedliche Anwendungen zu verwenden und auch für weitere Zwecke, wie z. B. Analysen der aufgetretenen Situationen, zu speichern.

Dieses Situationsmodell gilt daher als Grundlage für das Kontextmodell zur Modellierung von Analyseergebnissen. Aus den Monitoring- und Management-Daten erzeugt das Analyse-Framework zunächst Analyseergebnisse, die wiederum analysiert werden können. Zur Analyse verwendet das Analyse-Framework ein Analyse-Modell, dem die jeweiligen Analyseergebnisse entstammen (s. Abbildung 7.8). Hieraus lässt sich bspw. die Situationserkennung als Analyse-Framework ableiten, die als Analyse-Modell Situationstemplates verwendet und damit Situationen erkennt, die wiederum als Input für weitere höherwertige Situationserkennungen verwendet werden können (vgl. Kapitel 6).

Wie bei den Monitoring-Daten spielt auch bei Analyseergebnissen der Kontext eine entscheidende Rolle. Daher muss es zurückverfolgbar sein, aus welchen Monitoring-Daten (und Management-Daten) sowie Analyseergebnissen ein neues Analyseergebnis abgeleitet wurde. Ebenso gilt ein Analyseergebnis für eine oder mehrere Ressourcen. Des Weiteren besitzen Analyseergebnisse weitere Attribute wie z. B. einen Zeitstempel, der den Zeit-

punkt definiert, an dem das Analyseergebnis erzeugt wurde, eine Qualität, die aus der Qualität der verwendeten Monitoring-Daten abgeleitet werden kann, oder eine Beschreibung des Analyseergebnisses. Diese zusätzlichen Informationen wie z. B. Aktualität und Qualität können Auswirkungen auf Folgeaktionen haben. Beispielsweise wird ein automatisierter Prozess nur dann angestoßen, wenn eine gewisse Qualität der Daten und des erzeugten Analyseergebnisses vorhanden ist. Ansonsten wird ein Alert an die Systemadministratoren gesendet, die erkennen können, wie aktuell das Problem ist und welche Ressourcen davon betroffen sind.

In SitOPT entsteht auf diese Weise ein Situationsobjekt (s. Abbildung 7.7), anhand dessen alle relevanten Informationen zu einer erkannten Situation zusammengefasst werden. Relevante Informationen zu der Ressource, wie z. B. der Standort oder das zuständige Personal, können von den Geräte-Management-Systemen abgefragt werden. Auf diese Weise wird eine schnelle Reaktion auf auftretende Probleme ermöglicht.

7.3.5 Automation-Framework

Die Analyseergebnisse dienen dem Automation-Framework als Eingabe und führen zur Erzeugung von Alerts, die an Systemadministratoren geleitet werden. Alternativ können dadurch automatisierte Prozesse angestoßen werden, um auftretende Probleme automatisiert zu beheben. Sowohl Systemadministratoren als auch die automatisierten Prozesse müssen alle erforderlichen Informationen erhalten, um eine schnelle und korrekte Problembehebung zu ermöglichen. Dazu werden die Informationen verwendet, die bereits im Analyseergebnis, wie z. B. einem Situationsobjekt (s. Abbildung 7.7), vorhanden sind. Beispielsweise können bereits über das Situationsobjekt die zuständigen Systemadministratoren ermittelt werden, um das Erstellen von Alerts zu erleichtern.

Es ist zu erkennen, dass eine strikte Unterteilung in Analyse- oder Automation-Framework nicht immer möglich ist. So erlaubt z. B. das Alerting-Framework eines Monitoring-Systems sowohl die Definition und Auswertung regelbasierter Analysen (den Alerting-Regeln), die als Analysemodelle im

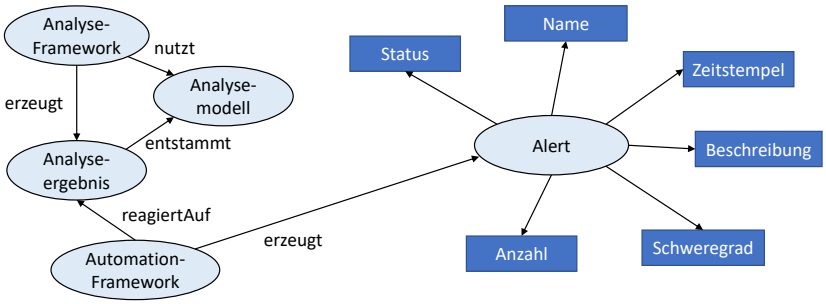

Abbildung 7.8: Detailansicht – Alert

Kontextmodell definiert werden, als auch die Erzeugung von Alerts an Systemadministratoren. Da allerdings auch fortgeschrittenere Analyseverfahren, wie z. B. neuronale Netze, in Betracht gezogen werden müssen, ist eine entsprechende Aufteilung in Analyse- und Automation-Framework vorgenommen worden, um Analysemethoden unabhängig von darauf folgenden, automatisierten Verfahren oder Alerts zu verwalten.

7.3.6 Alerts und automatisierte Prozesse

Zuletzt werden die Alerts und automatisierten Prozesse betrachtet, die auf Basis der Analyseergebnisse erzeugt bzw. angestoßen werden. Für die Reduktion der Anzahl von Alerts ist ein umfangreiches Management der Alerts unumgänglich. Demnach wird in diesem Bereich ein großer Fokus im Kontextmodell gesetzt, um Systemadministratoren so gut wie möglich zu unterstützen. Diese erhalten Alerts über verschiedene Kommunikationswege wie z. B. per E-Mail. Einerseits müssen alle relevanten Informationen zu einem Alert vorliegen, damit die Systemadministratoren Alerts korrekt einordnen und beheben können, andererseits werden Metadaten wie etwa die Anzahl der Instanzen eines bestimmten Alerts aufgezeichnet, um Alerts und deren Auftreten zu analysieren und dauerhaft zu reduzieren.

Die grundlegende Information kommt vom Analyseergebnis selbst. Wie im vorigen Abschnitt beschrieben, enthält das Analyseergebnis bereits viele relevante Informationen, wie z. B. die betroffene Ressource, die Daten, die zu dem Ergebnis geführt haben, und das verwendete Analysemodell (z. B. die verwendete Alerting-Regel oder das Situationstemplate). Zur klaren Identifikation werden Alerts mit einem Namen und einer detaillierten Beschreibung versehen (vgl. Abbildung 7.8). Der Schweregrad gibt an, wie wichtig ein Alert ist und wie schnell darauf reagiert werden muss. Beispielsweise kann eine allgemeine Aufteilung in *Warnung* und *Fehler* erfolgen. Ein Fehler deutet auf ein Problem hin, das unverzüglich behandelt werden muss, während eine Warnung darauf hindeutet, dass es in absehbarer Zeit zu einem Fehler kommt und präventive Maßnahmen ergriffen werden könnten. Zusätzlich sind auch bei Fehlern weitere Unterteilungen hilfreich. Beispielsweise ist dem Ausfall einer gesamten Cloud ein anderer Schweregrad zuzuweisen als dem Ausfall einer einzelnen Anwendung auf einer VM. Ein Zeitstempel gibt die Aktualität des Alerts an. Wie in Kapitel 5 beschrieben, ist auch hier ein sinnvolles Ziel, die Zeit zwischen dem Erkennen eines Problems (definiert durch den Zeitstempel im Analyseergebnis) und dem Versenden des Alerts möglichst zu minimieren. Der Status gibt an, in welchem Zustand sich der Alert befindet, bspw. *offen, in Bearbeitung* und *geschlossen*. Initial beginnt jeder Alert mit dem Status *offen* und wird mit der Zeit die zwei weiteren Status durchlaufen. Der Status *geschlossen* gibt außerdem an, dass Alerts auch nach ihrer Bearbeitung verwaltet werden, um darauf aufbauend weitere Analysen zu ermöglichen. Über die Anzahl eines Alerts kann ermittelt werden, wie oft genau dieser Alert bereits erzeugt wurde und wie oft das zugrundeliegende Problem auftrat.

7.4 Diskussion

Das Kontextmodell für das Monitoring industrieller Cloud-Umgebungen beschreibt alle grundlegenden Entitäten, die für das Monitoring industrieller Cloud-Umgebungen benötigt werden. Es wird ersichtlich, welche Daten wel-

chen Systemen entstammen und wie diese mit überwachten Ressourcen zusammenhängen. Gleichzeitig wird es ermöglicht, eine Topologie der industriellen Cloud-Umgebung zu erstellen und diese mit für das Monitoring hilfreichen Relationen anzureichern. Durch den Einsatz weiterer Ontologien aus den Bereichen IoT und I4.0 kann das in dieser Arbeit vorgestellte Kontextmodell beliebig verfeinert und erweitert werden, um als domänenübergreifendes Kontextmodell für das Monitoring industrieller Cloud-Umgebungen verwendet werden zu können.

Wie bereits beschrieben, sind detaillierte Informationen über die industrielle Cloud-Umgebung sowohl für DEAR (s. Kapitel 5) als auch für die verteilte Situationserkennung (s. Kapitel 6) von Vorteil. Bei DEAR kann die Topologie der Cloud-Umgebung genutzt werden, um die Verteilung der Alerting-Regeln so zu gestalten, um bspw. das Netzwerkverkehrvolumen zu minimieren. Für die verteilte Situationserkennung werden Details über die Edge-Infrastruktur benötigt, wie z. B. die IP-Adresse der VM und welche industriellen Assets ihre Daten dorthin übertragen, um ein automatisches Aufteilen von Situationstemplates zu unterstützen.

Zusätzlich entsteht ein großer Mehrwert für das Monitoring, wenn ein Reasoning auf die vorgestellte Ontologie angewendet wird. Durch diesen Prozess können Systemadministratoren bei vielen Aufgaben des Monitoring unterstützt werden, indem automatisch logische Schlussfolgerungen auf Basis einer modellierten industriellen Cloud-Umgebung gemacht werden können. Beispielsweise führt der Ausfall einer VM zu kaskadierenden Ausfällen von Containern und Anwendungen, die darauf bereitgestellt werden, was daher üblicherweise von mehreren Monitoring-Systemen gleichzeitig erkannt wird. Mit der Relation *bereitgestellt-auf* kann hierbei jedoch die Suche nach der Fehlerursache im Falle eines Alerts schneller eingegrenzt werden. Gleichzeitig können diese Alerts zu einem einzigen Alert zusammengefasst werden, um die Anzahl von Alerts und damit den Arbeitsaufwand von Systemadministratoren zu verringern. Des Weiteren können wiederholte Alerts, die auf das gleiche weiterhin bestehende Problem hinweisen, dedupliziert werden. Stattdessen wird in diesem Fall das Feld *Anzahl* des originalen Alerts erhöht (vgl. Abbildung 7.8). Zusätzlich ist für jeden Alert

nachvollziehbar, welche Analyse mit welchen Daten aus welchen Systemen verwendet wurde und welche überwachten Ressourcen davon betroffen sind. Dies erlaubt eine detaillierte Untersuchung und unterstützt zusätzlich die Suche nach Fehlerquellen.

7.5 Zusammenfassung

In diesem Kapitel wurde ein Kontextmodell für das Monitoring industrieller Cloud-Umgebungen vorgestellt. Das Kontextmodell beschreibt alle Entitäten, die für das Monitoring und Management dieser Umgebungen relevant sind, und wie diese zueinander in Beziehung stehen. Die einzelnen Entitäten wurden detailliert erklärt und sollen als Grundlage für die Erstellung von Topologien industrieller Cloud-Umgebungen dienen. Zusätzlich wurden sowohl die Analyse als auch darauf aufbauende Aktionen, wie z. B. Alerting, im Kontextmodell modelliert. Damit kann ein automatisches Reasoning Systemadministratoren beim Monitoring unterstützen, indem Fehlerursachen schneller eingegrenzt und die Anzahl von Alerts reduziert werden können.

Kapitel 8

Zusammenfassung und zukünftige Arbeiten

In dieser Arbeit wurde Konzept zur Verbesserung des Monitoring in industriellen Cloud-Umgebungen vorgestellt. Dazu wurden domänenspezifische Lösungen sowohl für das Monitoring von Cloud-Umgebungen als auch für das Monitoring industrieller Assets analysiert und in Anbetracht der Anforderungen der jeweiligen Domänen verbessert. Zuletzt wurden diese beiden Domänen im letzten Beitrag in einem Kontextmodell für das Monitoring industrieller Cloud-Umgebungen konsolidiert. Folgende Beiträge sind dabei entstanden:

- Beitrag B1: Generische Agententemplates
- Beitrag B2: Verteilte Auswertung von Alerting-Regeln
- Beitrag B3: Verteilte Situationserkennung
- Beitrag B4: Kontextmodell für das Monitoring industrieller Cloud-Umgebungen

Im ersten Beitrag wurden zunächst die Agentenkonzepte verschiedener Monitoring-Systeme analysiert. Daraus resultierend wurden Anforderungen an die Funktionalitäten aufgestellt, die von generischen Agententemplates erfüllt werden müssen. Auf dieser Basis wurde die Agenten-Pipeline als Abstraktionsstufe für die Modellierung von Agenten eingeführt. Somit wurde von technischen Details abstrahiert und die Modellierung auf das allgemeine Verhalten des Agenten beschränkt und somit vereinfacht. Mittels von Domänenexperten bereitgestellter Transformationslogik können diese in lösungsspezifische Agenten ausgewählter Monitoring-Systeme überführt werden. Dadurch wird der Austausch von Monitoring-Systemen erleichtert und die Gefahr von Vendor-Lock-ins verringert. Nicht alle lösungsspezifischen Agenten unterstützen allerdings die im generischen Agenten modellierten Funktionalitäten, wie z. B. Aggregation der Monitoring-Daten. Für diese Fälle wird ein Outsourcing der fehlenden Funktionalität auf eine CEP-Engine ermöglicht, wodurch generische Agenten auf eine Vielzahl lösungsspezifischer Agenten transformiert werden können. Um die Verwaltung der Agenten zu unterstützen, wurde zusätzlich ein erweiterter Lebenszyklus für Agenten vorgestellt. Dieser enthält die automatische Bereitstellung transformierter, lösungsspezifischer Agenten und die Propagierung von Änderungen in generischen Agenten auf daraus entstammenden, bereitgestellten Agenten.

Im zweiten Beitrag wurde das Plug-in *DEAR* für eine verteilte Auswertung von Alerting-Regeln vorgestellt. *DEAR* stellt einen hybriden Ansatz dar, der die Vorteile der zentralisierten mit denen der dezentralisierten Monitoring-Architekturen vereint. Dadurch wurde ein feingranulares Alerting ermöglicht, ohne jedoch die Nachteile der Aggregation von Monitoring-Daten oder der erhöhten Management-Komplexität dezentralisierter Monitoring-Architekturen zu übernehmen. Die Alerting-Regeln werden weiterhin von den Systemadministratoren auf dem zentralen Monitoring-Server im Alerting-Framework definiert und verwaltet, sodass für diese durch den Einsatz von *DEAR* kein erhöhter Administrationsaufwand entsteht. *DEAR* greift auf diese Alerting-Regeln zu und verteilt sie daraufhin auf die VMs. Es wurden verschiedene Strategien zur Verteilung der Alerting-Regeln präsentiert, die je nach Anwendungsfall eine Verteilung ermöglichen sollen. Die

Verteilung selbst findet automatisiert statt und erfordert somit kein menschliches Eingreifen. Zunächst wurden die Regelbedingungen der Alerting-Regeln auf Auswertungsmodulen der jeweiligen VMs platziert. Anschließend wurden die Agenten des Monitoring-Systems adaptiert, welche konform zu den im ersten Beitrag vorgestellten Agententemplates sind. Die unveränderten und daher feingranularen Monitoring-Daten wurden an das entsprechende Auswertungsmodul zur Auswertung der Regelbedingung gesendet. Die Ergebnisse wurden wiederum über den Agenten an das zentrale Alerting-Framework weitergeleitet. Dort werden weiterhin die Aktionen von den Systemadministratoren verwaltet, die beim Verletzen einer Alerting-Regel ausgeführt werden sollen. Gleichzeitig werden aggregierte Monitoring-Daten direkt an den zentralen Monitoring-Server geschickt, um diese zu historisieren und weitergehende Analysen zu ermöglichen. Da die Granularität dieser aggregierten Monitoring-Daten die Qualität des Alerting nicht beeinträchtigt, kann ein vergleichsweise hoher Aggregationszeitraum gewählt werden, um die Netzwerkbandbreite nicht zu belasten. Zuletzt werden für einen limitierten Zeitraum die feingranularen Monitoring-Daten auf den VMs in einem Puffer vorgehalten. Wird eine Alerting-Regel verletzt, ist es auf diese Weise möglich, bedarfsorientiert auch auf die feingranularen Monitoring-Daten zuzugreifen, um diese für weitere Analysen, wie z. B. eine Fehlerursachenanalyse, zu verwenden. Es wurde auf eine modulare Architektur Wert gelegt, um die Konzepte von *DEAR* auf eine Vielzahl von Monitoring-Systemen anwenden zu können, indem weitere Alerting-Frameworks und Agenten eingebunden werden.

Im dritten Beitrag wurde auf Basis der Verwendung von Situationstemplates eine verteilte Situationserkennung für das Monitoring industrieller Assets vorgestellt. Dazu wurden zunächst verschiedene Anforderungen der Industrie 4.0 erfasst und darauf aufbauend verschiedene Ausführungsumgebungen für die Situationserkennung analysiert. Es wurde gezeigt, dass je nach Anwendungsfall eine Ausführung ausschließlich auf der Public-Cloud oder der Edge, sowie eine Kombination davon praktikabel sein kann. Eine Kombination dieser beiden Möglichkeiten führte zu der Anforderung einer verteilten Situationserkennung. Dazu wurde zunächst eine schichtenbasierte

Modellierung für Situationstemplates vorgestellt, um die Wiederverwendung von bereits modellierten Situationstemplates zu ermöglichen. Ein grafisches Modellierungswerkzeug erleichtert die Modellierung und Verwaltung von Situationstemplates. Werden Situationstemplates zu Beginn nicht separat modelliert, werden Benutzer bei der Instanziierung der Situationstemplates durch ein automatisches Aufteilen der Situationstemplates unterstützt. Durch eine Kontextentfernung werden die zur Erkennung einer Situation beitragenden Kontext-Daten aus den generierten Situationsobjekten entfernt. Dadurch wird bei der Übermittlung des Situationsobjekts von der Edge in die Cloud die Belastung für die Netzwerkbandbreite stark reduziert und weiterhin eine globale Situationserkennung ermöglicht.

Zuletzt wurde im vierten Beitrag ein Kontextmodell für das Monitoring industrieller Cloud-Umgebungen vorgestellt. Das Kontextmodell bildet alle relevanten Aspekte bzgl. des Monitoring dieser Umgebungen ab und unterstützt Systemadministratoren, indem bspw. die Topologie einer überwachten industriellen Cloud-Umgebung erstellt werden kann und die Anzahl von Alerts reduziert wird. Zum Kontextmodell zählen in erster Linie die überwachten Ressourcen und deren Abhängigkeiten zueinander, um diese im Rahmen der überwachten, industriellen Cloud-Umgebung abbilden zu können. Des Weiteren werden die Monitoring- und Management-Systeme, die davon erfassten bzw. verwalteten Monitoring- und Management-Daten, Analyse-Frameworks und deren Analyseergebnisse, und Automation-Frameworks, die Alerting und automatisierte Prozesse verwalten, betrachtet.

8.1 Erfüllung der Ziele

In diesem Abschnitt wird diskutiert, inwiefern die in der vorliegenden Arbeit vorgestellten Beiträge die in Abschnitt 1.2 definierten Ziele erfüllt haben.

8.1.1 Gefahr von Lock-ins bei agentenbasierten Monitoring-Systemen verringern

Das erste Ziel befasste sich mit der Gefahr eines Vendor-Lock-ins bei agentenbasierten Monitoring-Systemen. Dazu wurden in Kapitel 4 generische Agententemplates vorgestellt, um den Austausch der Agenten der Monitoring-Systeme zu automatisieren. Generische Agenten müssen nur einmal modelliert werden, solange die Aufgaben der Agenten sich nicht verändern sollen. Die benötigte Transformationslogik wird von Domänenexperten bereitgestellt und ermöglicht eine automatische Transformation der generischen Agenten in lösungsspezifische Agenten des gewünschten Monitoring-Systems. Dies ist selbst dann möglich, wenn die lösungsspezifischen Agenten nicht alle benötigten Funktionalitäten unterstützen.

Dadurch wurde die Gefahr eines Vendor-Lock-ins reduziert und das Ziel wurde erfüllt. Zusätzlich wurden bei den weiteren Beiträgen möglichst technologieunabhängige Ansätze verfolgt, um diese Gefahr nicht wieder zu erhöhen. *DEAR* ermöglicht als Plug-in für agentenbasierte Monitoring-Systeme mit einer modularen Architektur die Unterstützung mehrerer Alerting-Frameworks. Zusätzlich basiert *DEAR* auf generischen Agententemplates und kann daher ebenso auf eine Vielzahl lösungsspezifischer Agenten angewendet werden.

8.1.2 Akkurate Analyse in agentenbasierten Monitoring-Systemen bei niedrigem Netzwerkverkehrvolumen

Das zweite Ziel beschreibt eine verteilte Auswertung von Alerting-Regeln auf den überwachten VMs, um das Netzwerkverkehrvolumen gering zu halten. In Kapitel 5 wurde *DEAR* als Plug-in für agentenbasierte Monitoring-Systeme zur Verteilung von Alerting-Regeln vorgestellt. *DEAR* ermöglicht es, dass feingranulare Monitoring-Daten für das Alerting verwendet werden können. Um weiterhin Analysen auf historischen Daten zu ermöglichen, werden gleichzeitig grobgranulare Monitoring-Daten an den zentralen Monitoring-Server geschickt, welche aufgrund einer hohen Aggregationsdichte die Netz-

werkbandbreite nur geringfügig belasten. Zusätzlich kann bei auftretenden Fehlern auf feingranulare Monitoring-Daten zugegriffen werden, die über einen kurzen Zeitraum auf den VMs gespeichert werden. Alle von *DEAR* verursachten Veränderungen werden automatisiert ausgeführt und erhöhen die Administrationskomplexität nicht.

Somit sind verschiedene Anwendungsfälle von Analysen abgedeckt und das Ziel ist erfüllt. Vor allem das Alerting kann auf feingranulare Monitoring-Daten zugreifen, ohne das Netzwerkverkehrvolumen zu erhöhen. Eine quantitative Evaluation hat ergeben, dass der Einsatz von *DEAR* im Vergleich zu agentenbasierten Monitoring-Systemen zu weniger Netzwerkverkehrvolumen und einer kürzeren Time-to-Insight führt.

8.1.3 Analyse verschiedener Ausführungsumgebungen und verteilte Ausführung der Situationserkennung

Im dritten Beitrag zur verteilten Situationserkennung wurden in Kapitel 6 verschiedene Ausführungsumgebungen in Anbetracht mehrerer Anforderungen der Industrie 4.0 analysiert. Es wurde gezeigt, dass je nach Anwendungsfall verschiedene Ausführungsumgebungen praktikabel sind. Anschließend wurde eine verteilte Situationserkennung ermöglicht. Dazu wurden Veränderungen sowohl bei der Modellierung von Situationstemplates als auch bei der Ausführung der Situationserkennung vorgenommen. Somit wurde das Ziel erfüllt.

8.1.4 Umfassende Sichtweise auf eine industrielle Cloud-Umgebung ermöglichen

Das letzte Ziel dieser Arbeit war es, ein Kontextmodell zu erstellen, das alle grundlegenden Informationen enthält, die für das Monitoring industrieller Cloud-Umgebungen benötigt werden. Hierzu wurde in Kapitel 7 eine Ontologie präsentiert, welche die überwachten Ressourcen, die Monitoring- und Management-Daten zur Beschreibung dieser Ressourcen, und die jeweiligen Systeme zur Verwaltung bzw. Erfassung dieser Daten enthält. Des

Weiteren wurden auf den Daten aufbauende Analysen und daraus resultierende Aktionsmöglichkeiten, wie z. B. Alerting, im Kontextmodell modelliert. Gemeinsam ermöglicht dies die Modellierung einer Topologie einer industriellen Cloud-Umgebung, in der alle relevanten Aspekte und deren Relationen zueinander beschrieben werden können. Ein automatisches Reasoning unterstützt Systemadministratoren bei der Fehlerursache und ermöglicht eine Reduktion der Anzahl von Alerts.

8.2 Zukünftige Arbeiten

8.2.1 Generische Alerting-Regeln

Analog zu generischen Agententemplates kann in zukünftigen Arbeiten ein Ansatz für die Modellierung generischer Alerting-Regeln verfolgt werden. Dazu wird ebenso eine Abstraktionsstufe für das Modellieren von Alerting-Regeln eingeführt und kann anschließend auf Alerting-Regeln verschiedener Alerting-Frameworks überführt werden und schließlich mittels *DEAR* wiederum auf den Agenten ausgewertet werden. Dadurch entsteht eine weitere Abkopplung lösungsspezifischer Komponenten von Monitoring-Systemen, wodurch die Gefahr eines Vendor-Lock-ins weiterhin reduziert wird.

Des Weiteren kann die Wahl sinnvoller Schwellenwerte für die Alerting-Regeln durch Analysen auf historischen Daten der Metriken der Ressourcen besser eingegrenzt werden bzw. durch dynamische Schwellenwerte ergänzt oder möglicherweise sogar ersetzt werden. Grundlage für diese Analysen kann die im folgenden Abschnitt vorgeschlagene Plattform sein.

8.2.2 Holistisches Monitoring

Im Bereich des Cloud-Monitoring wurde von Gartner das Konzept *AIOps* (Artificial Intelligence for IT Operations) eingeführt, wodurch in erster Linie der Einsatz von fortschrittlichen Analysemethoden in der Domäne des IT-Monitoring forciert werden soll [BMC20]. Hierfür soll eine AIOps-Plattform

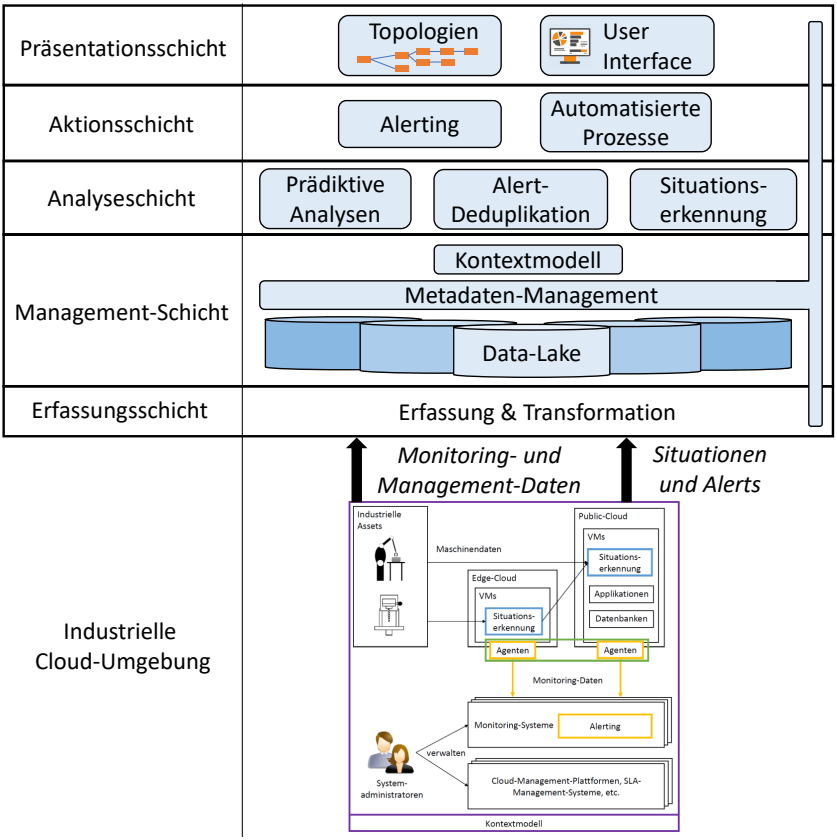

Abbildung 8.1: Architektur der Monitoring-Plattform und Integration in eine industrielle Cloud-Umgebung

verwendet werden, die eine Konsolidierung aller Monitoring-Daten und darauf aufbauend Analysen, wie z. B. Anomaliedetektion oder Fehlerursachenanalyse, ermöglichen soll.

Als Grundvoraussetzung für eine solche Plattform sieht Gartner ein Kontextmodell zur Beschreibung der Cloud-Umgebung vor. Daher kann das in Kapitel 7 vorgestellte Kontextmodell als Grundlage für die Erstellung einer AIOps-Plattform dienen, welche das holistische Monitoring einer industriel-

len Cloud-Umgebung ermöglicht. Eine solche Plattform soll zunächst dafür verwendet werden, um alle relevanten Daten aus den jeweiligen Datensilos der Monitoring- und Management-Systeme zu konsolidieren und für weitere Zwecke verfügbar zu machen. Die Daten können hierfür in einem für den Umgang mit Big-Data geeigneten Speichersystem historisiert werden. Daraufhin können Analysen auf den Daten durchgeführt und entsprechende Aktionen ausgeführt werden.

Mit einer mehrschichtigen Architektur sollen folgende Aufgaben und Funktionen erfüllt werden:

- Erfassung von Monitoring- und Management-Daten aus heterogenen Quellsystemen
- Management der erfassten Daten
- Analyse der erfassten Daten
- Ausführung von Aktionen basierend auf Analyseergebnissen
- Bereitstellung einer grafischen Benutzeroberfläche
- Umfassendes Metadaten-Management

In Abbildung 8.1 sind die jeweiligen Schichten aufgeführt. Zudem ist zu sehen, dass die industrielle Cloud-Umgebung (vgl. Abbildung 3.1) bestehen bleibt. Das bedeutet, dass die Monitoring-Plattform die jeweiligen Systeme nicht ersetzen wird. Stattdessen werden diese Systeme genutzt, um eine Konsolidierung aller relevanten Daten für das Monitoring industrieller Cloud-Umgebungen zu ermöglichen, die als Grundlage für ein holistisches Monitoring dienen.

Im Folgenden wird eine Übersicht der einzelnen Schichten gegeben.

8.2.2.1 Industrielle Cloud-Umgebung

Die industrielle Cloud-Umgebung, wie sie in Kapitel 3 vorgestellt wurde, bleibt weiterhin bestehen und die darin enthaltenen Monitoring- und Management-Systeme liefern die Monitoring- und Management-Daten, die

in der Monitoring-Plattform konsolidiert werden sollen. Zusätzlich werden auch höherwertige Informationen, wie z. B. Situationen oder Alerts, aus den jeweiligen Systemen an die Monitoring-Plattform übertragen. Die zugrundeliegenden Systeme werden demnach weder verändert noch ersetzt, weshalb die Monitoring-Plattform sich in bestehende industrielle Cloud-Umgebungen integrieren lässt, ohne in dieser tiefgreifende Veränderungen vornehmen zu müssen.

8.2.2.2 Erfassungsschicht

Die Erfassung aller relevanten Daten bzgl. Monitoring und Management einer Cloud-Umgebung ist der erste Schritt, der für eine holistische Sicht benötigt wird. Die Daten liegen zunächst in Datensilos der jeweiligen Monitoring- und Management-Systeme vor. Um Zugriff auf diese zu erhalten, werden zunächst individuell für alle *Monitoring- und Management-Systeme* Schnittstellen erstellt, um die Daten zu extrahieren und transformieren, und in entsprechende Datenbanken der Plattform zu laden.

Bei Monitoring-Systemen zählen hierzu die Rohdaten, wie z. B. von Agenten gesammelte Zeitreihendaten, sowie bereits verarbeitete Daten, wie z. B. in der Datenbank aggregierte Durchschnittswerte der Rohdaten über bestimmte Zeiträume. Zuletzt werden alle Alert-Daten von Monitoring-Systemen und Situationen einer Situationserkennung in Form von Situationsobjekten direkt an die Plattform weitergeleitet.

8.2.2.3 Management-Schicht

Um effizient im Umgang mit den heterogenen Daten zu sein, die über die Erfassungsschicht aus den Quellsystemen extrahiert wurden, ist es sinnvoll, für jeden Datentyp das am besten geeignete Speichersystem zu verwenden [GH19]. Eine Möglichkeit zur Umsetzung dieser Anforderung ist ein Data-Lake, welcher als Datenspeicher definiert wird, in dem Daten in ihrem Rohformat gespeichert werden, um für analytische Zwecke genutzt zu werden [HGQ16].

Auf Basis des in Kapitel 7 vorgestellten Kontextmodells können die Daten über ein Metadaten-Management miteinander entsprechend verknüpft werden, um eine effiziente Datenorganisation zu ermöglichen.

8.2.2.4 Analyseschicht

Durch die Verfügbarkeit großer Datenmengen haben fortschrittliche Methoden zur Analyse von Daten Einzug in die Domänen der verarbeitenden Industrie und des IT-Monitoring gefunden. Die Analyseschicht der Plattform soll sämtliche Methoden beinhalten, die zur Analyse der im Data-Lake gespeicherten Daten dienen. Anwendungsfälle im industriellen Bereich sind beispielsweise *Predictive Maintenance* (dt.: Prädiktive Instandhaltung), für die mittels historischer Daten das Verhalten industrieller Assets vorhergesagt wird, bspw. mittels neuronaler Netze des Typs Long-Short-Term-Memory (LSTM; dt.: langes Kurzzeitgedächtnis) oder Methoden wie Autoregressive Integrated Moving Average (ARIMA; dt.: autoregressiver integrierter gleitender Durchschnitt) [STN18]. Diese Vorhersagen ermöglichen u. a. eine effizientere Planung und verringern damit die Stillstandzeiten der industriellen Assets [Has11]. Im Bereich des IT-Monitoring sind Alerting-Regeln und die Ursachenanalyse eines Fehlers die üblichen Anwendungsfälle. Im Rahmen von AIOps sind neue Anwendungsfälle hinzugekommen, wie z. B. die Deduplizierung von Alerts mittels Clustering-Algorithmen.

8.2.2.5 Aktionsschicht

In der Aktionsschicht werden auf Basis der zuvor erstellen Analyseergebnisse Aktionen ausgeführt, um erkannte Probleme entweder manuell oder automatisiert zu lösen. Diese Aktionen werden daher in Alerting und automatisierte Prozesse unterteilt. Zunächst wird versucht, die Probleme automatisiert mit den entsprechenden Prozessen zu beheben. Falls jedoch eine automatisierte Behebung nicht möglich ist bzw. entsprechende Prozesse nicht vorhanden sind, wird mittels Alerting das verantwortliche Personal über individuelle Kommunikationswege über das Problem benachrichtigt.

8.2.2.6 Präsentationsschicht

Die letzte Schicht stellt die grafische Schnittstelle für alle Nutzer der Plattform dar. Es werden verschiedene Visualisierungsmöglichkeiten benötigt, um die heterogenen Daten (z. B. Zeitreihendaten, Dokumente und Clustering-Analyseergebnisse) anzuzeigen. Neben der Erstellung eigener Visualisierungen können hierfür die bereits existierenden, grafischen Benutzeroberflächen der jeweiligen Speichersysteme oder Analyse-Frameworks verwendet werden. Die Metadaten oder die Topologie der überwachten, industriellen Cloud-Umgebung können bspw. über die Benutzeroberfläche einer Graphdatenbank betrachtet werden. Des Weiteren werden alle administrativen Aufgaben, wie z. B. die Definition von Aufbewahrungsrichtlinien, die nach Ablauf eines definierten Zeitraums angefallene Monitoring-Daten aggregieren oder löschen, über diese Schicht verwaltet.

Publikationen des Autors

Publikationen als Hauptautor

- Mormul, Mathias; Hirmer, Pascal; Wieland, Matthias; Mitschang, Bernhard: *Situation model as interface between situation recognition and situation-aware applications*. Computer Science - Research and Development. Vol. 32(3-4), Springer-Verlag, 2017.
- Mormul, Mathias; Hirmer, Pascal; Wieland, Matthias; Mitschang, Bernhard: *Layered Modeling Approach for Distributed Situation Recognition in Smart Environments*. Proceedings of the 7th International Conference on Smart Cities, Systems, Devices and Technologies (SMART), 2018.
- Mormul, Mathias; Hirmer, Pascal; Wieland, Matthias; Mitschang, Bernhard: *Distributed Situation Recognition in Industry 4.0*. International Journal On Advances in Intelligent Systems. Vol. 12(1-2), IARIA, 2019.
- Mormul, Mathias; Stach, Christoph: *A Context Model for Holistic Monitoring and Management of Complex IT Environments*. Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (CoMoRea), 2020.

- Mormul, Mathias; Hirmer, Pascal; Stach, Christoph; Mitschang, Bernhard: *Avoiding Vendor-Lockin in Cloud Monitoring using Generic Agent Templates*. Proceedings of the 23rd International Conference on Business Information Systems (BIS), 2020.
- Mormul, Mathias; Hirmer, Pascal; Stach, Christoph; Mitschang, Bernhard: *DEAR: Distributed Evaluation of Alerting Rules*. Proceedings of the 13th International Conference on Cloud Computing (CLOUD), 2020.

Publikationen als Co-autor

- Zielinski, Erich, et al.: *Secure Real-time Communication and Computing Infrastructure for Industry 4.0—Challenges and Opportunities*. 2019 IEEE International Conference on Networked Systems (NetSys), 2019.
- Petrik, Dimitri; Mormul, Mathias; Reimann, Peter: *Anforderungen für Zeitreihendatenbanken in der industriellen Edge*. HMD Praxis der Wirtschaftsinformatik. Vol 56, Springer-Verlag, 2019.
- Petrik, Dimitri; Mormul, Mathias; Reimann, Peter; Gröger, Christoph: *Anforderungen für Zeitreihendatenbanken im industriellen IoT*. IoT Best Practices, Springer-Verlag, 2021.

Literaturverzeichnis

[ABdP13] G. Aceto, A. Botta, W. de Donato, A. Pescapè. „Cloud monitoring: A survey“. In: *Computer Networks* 57.9 (2013), S. 2093–2115 (Zitiert auf S. 16, 18, 25, 38, 55, 56, 77, 79, 84, 85).

[Ama21a] Amazon Web Services, Inc. *Amazon CloudWatch*. 2021. URL: `https://aws.amazon.com/de/cloudwatch/` (Zitiert auf S. 57).

[Ama21b] Amazon Web Services, Inc. *CloudWatch Agent*. 2021. URL: `https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Install-CloudWatch-Agent.html` (Zitiert auf S. 57).

[AMD+11] F. Azmandian, M. Moffie, J. G. Dy, J. A. Aslam, D. R. Kaeli. „Workload Characterization at the Virtualization Layer“. In: *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. 2011, S. 63–72 (Zitiert auf S. 79).

[Ans73] F. J. Anscombe. „Graphs in statistical analysis“. In: *The american statistician* 27.1 (1973), S. 17–21 (Zitiert auf S. 84).

[ARM+15] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, V. Bhatnagar. „An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art“. In: *Computing* 97.4 (2015), S. 357–377 (Zitiert auf S. 18, 25, 77).

[ASRH13] J. Attard, S. Scerri, I. Rivera, S. Handschuh. „Ontology-Based Situation Recognition for Context-Aware Systems“. In: *Proceedings of the 9th International Conference on Semantic Systems*. New York, NY, USA, 2013, S. 113–120 (Zitiert auf S. 26).

[ATA+10] O. Anya, H. Tawfik, S. Amin, A. Nagar, K. Shaalan. „Context-aware knowledge modelling for decision support in e-health“. In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. 2010, S. 1–7 (Zitiert auf S. 146).

[BBKL14] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. „Vinothek - A Self-Service Portal for TOSCA“. In: *Proceedings of the 6th Central-European Workshop on Services and their Composition (ZEUS 2014)*. Bd. 1140. CEUR Workshop Proceedings. 2014, S. 69–72 (Zitiert auf S. 69).

[BGM14] D. Brickley, R. Guha, B. McBride. *RDF Schema 1.1*. 2014. URL: `https://www.w3.org/TR/rdf-schema/` (Zitiert auf S. 149).

[Big19] BigPanda, Inc. *Future of Monitoring and AIOps*. 2019. URL: `https://www.bigpanda.io/blog/the-results-of-our-2019-future-of-monitoring-and-aiops-survey/` (Zitiert auf S. 17, 143).

[BMC14] BMC Software, Inc. *BMC PATROL Agent overview*. 2014. URL: `https://docs.bmc.com/docs/display/public/PN90/BMC+PATROL+Agent+overview` (Zitiert auf S. 57).

[BMC18] BMC Software, Inc. *TrueSight Infrastructure Management overview*. 2018. URL: `https://docs.bmc.com/docs/display/tsim107/TrueSight+Infrastructure+Management+overview` (Zitiert auf S. 57).

[BMC20] BMC Software, Inc. *What is AIOps? Artificial Intelligence for IT Operations Explained*. 2020. URL: `https://www.bmc.com/blogs/what-is-aiops/` (Zitiert auf S. 27, 50, 56, 85, 86, 167).

[Bow18] D. A. Bowen. „Challenges Archivists Encounter Adopting Cloud Storage for Digital Preservation“. In: *Proceedings of the International Conference on Information and Knowledge Engineering (IKE)*. 2018, S. 27–33 (Zitiert auf S. 18, 51).

[Bun20] Bundesministerium für Bildung und Forschung. *Industrie 4.0 - Innovationen im Zeitalter der Digitalisierung*. 2020. URL: https://www.bmbf.de/upload_filestore/pub/Industrie_4.0.pdf (Zitiert auf S. 18, 19, 35).

[CA15] J. M. A. Calero, J. G. Aguado. „Comparative Analysis of Architectures for Monitoring Cloud Computing Infrastructures“. In: *Future Generation Computer Systems* 47 (2015), S. 16–30 (Zitiert auf S. 17, 49).

[Cam21] Cambridge University Press 2021. *Monitoring*. 2021. URL: https://dictionary.cambridge.org/dictionary/english/monitoring (Zitiert auf S. 36).

[CH09] D. Cohn, R. Hull. „Business artifacts: A Data-centric Approach to Modeling Business Operations and Processes“. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (2009) (Zitiert auf S. 53).

[CHML14] H. Chang, A. Hari, S. Mukherjee, T. Lakshman. „Bringing the cloud to the edge“. In: *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2014, S. 346–351 (Zitiert auf S. 19, 34).

[CM12] G. Cugola, A. Margara. „Complex event processing with T-REX“. In: *Journal of Systems and Software* 85.8 (2012), S. 1709–1728 (Zitiert auf S. 82).

[Dat21a] Datadog, Inc. *Datadog*. 2021. URL: https://www.datadoghq.com/ (Zitiert auf S. 57).

[Dat21b] Datadog, Inc. *Datadog Agent*. 2021. URL: https://docs.datadoghq.com/agent/ (Zitiert auf S. 57).

[DGB08] M. Dobrev, D. Gocheva, I. Batchkova. „An ontological approach for planning and scheduling in primary steel production“. In: *2008 4th International IEEE Conference Intelligent Systems*. Bd. 1. 2008, S. 6–14 (Zitiert auf S. 145).

[DSW+18] M. T. Dharmawan, H. T. Sukmana, L. K. Wardhani, Y. Ichsani, I. Subchi. „The Ontology of IT Service Management by Using ITILv.3 Framework: A Case Study for Incident Management“. In: *2018 Third International Conference on Informatics and Computing (ICIC)*. 2018, S. 1–5 (Zitiert auf S. 146).

[Ecl20] Eclipse. *Winery*. 2020. URL: `https://eclipse.github.io/winery/` (Zitiert auf S. 69).

[EEKS11] T. Eilam, M. Elder, A. V. Konstantinou, E. Snible. „Pattern-based composite application deployment“. In: *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. 2011, S. 217–224 (Zitiert auf S. 25, 53).

[Ela21a] Elasticsearch. *Elasticsearch*. 2021. URL: `https://www.elastic.co` (Zitiert auf S. 57).

[Ela21b] Elasticsearch B.V. *Beats*. 2021. URL: `https://www.elastic.co/de/beats/` (Zitiert auf S. 57, 62).

[Esp21] EsperTech Inc. *Esper*. 2021. URL: `http://www.espertech.com/esper/` (Zitiert auf S. 94, 99).

[FBB+14] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, F. Leymann. „From pattern languages to solution implementations“. In: *Proceedings of the Sixth International Conference on Pervasive Patterns and Applications (PATTERNS)*. 2014, S. 12–21 (Zitiert auf S. 25, 53).

[FH20] A. C. Franco da Silva, P. Hirmer. „Models for Internet of Things Environments—A Survey“. In: *Information* 11.10 (2020), S. 487 (Zitiert auf S. 146).

[FHWM16] A. C. Franco da Silva, P. Hirmer, M. Wieland, B. Mitschang. „SitRS XT-Towards Near Real Time Situation Recognition“. In: *Journal of Information and Data Management* (2016) (Zitiert auf S. 117, 127, 140).

[FJJG13] W. Funika, M. Janczykowski, K. Jopek, M. Grzegorczyk. „An ontology-based approach to performance monitoring of MUSCLE-bound multiscale applications“. In: *Procedia Computer Science* 18 (2013), S. 1126–1135 (Zitiert auf S. 146).

[Flo15] D. Floyer. *The Vital Role of Edge Computing in the Internet of Things*. Okt. 2015. URL: https://wikibon.com/the-vital-role-of-edge-computing-in-the-internet-of-things/ (Zitiert auf S. 127).

[FZYZ08] Q. Fang, Y. Zhao, G. Yang, W. Zheng. „Scalable Distributed Ontology Reasoning Using DHT-Based Partitioning“. In: *The Semantic Web*. 2008, S. 91–105 (Zitiert auf S. 26, 112).

[GBH+05] M. Grossmann, M. Bauer, N. Honle, U.-P. Kappeler, D. Nicklas, T. Schwarz. „Efficiently Managing Context Information for Large-Scale Scenarios“. In: *3rd IEEE International Conference on Pervasive Computing and Communications*. 2005, S. 331–340 (Zitiert auf S. 110).

[GFH+13] H. Guermah, T. Fissaa, H. Hafiddi, M. Nassar, A. Kriouile. „Context modeling and reasoning for building context aware services“. In: *2013 ACS international conference on computer systems and applications (AICCSA)*. 2013, S. 1–7 (Zitiert auf S. 146).

[GH19] C. Gröger, E. Hoos. „Ganzheitliches Metadatenmanagement im Data Lake: Anforderungen, IT-Werkzeuge und Herausforderungen in der Praxis“. In: *BTW 2019*. 2019, S. 435–452 (Zitiert auf S. 170).

[Gra21] Grafana Labs. *Grafana*. 2021. URL: https://grafana.com/ (Zitiert auf S. 99).

[GSZ18] F. Giustozzi, J. Saunier, C. Zanni-Merk. „Context modeling for industry 4.0: An ontology-based proposal“. In: *Procedia Computer Science* 126 (2018), S. 675–684 (Zitiert auf S. 145).

[GW12] R. Guo, J. Wu. „Design and implementation of domain ontology-based oilfield non-metallic pipe information retrieval system“. In: *2012 International Conference on Computer Science and Information Processing (CSIP)*. 2012, S. 813–816 (Zitiert auf S. 145).

[Har17] B. Harzog. *Modern monitoring is a big data problem*. Apr. 2017. URL: https://www.networkworld.com/article/3191479/modern-monitoring-is-a-big-data-problem.html (Zitiert auf S. 18, 25, 77, 85).

[Has11] H. M. Hashemian. „State-of-the-Art Predictive Maintenance Techniques“. In: *IEEE Transactions on Instrumentation and Measurement* 60.1 (2011), S. 226–236 (Zitiert auf S. 171).

[HBPH14] J. Haupert, S. Bergweiler, P. Poller, C. Hauck. „IRAR: smart intention recognition and action recommendation for cyber-physical industry environments“. In: *2014 International Conference on Intelligent Environments*. 2014, S. 124–131 (Zitiert auf S. 145).

[HGQ16] R. Hai, S. Geisler, C. Quix. „Constance: An intelligent data lake system“. In: *Proceedings of the 2016 International Conference on Management of Data*. 2016, S. 2097–2100 (Zitiert auf S. 170).

[HHL+10] K. Häussermann, C. Hubig, P. Levi, F. Leymann, O. Simoneit, M. Wieland, O. Zweigle. „Understanding and designing situation-aware mobile and ubiquitous computing systems“. In: *Proceedings of International Conference on Mobile, Ubiquitous and Pervasive Computing*. 2010, S. 329–339 (Zitiert auf S. 23, 40).

[Hir18] P. Hirmer. *Anforderungsbasierte Modellierung und Ausführung von Datenflussmodellen*. 2018. URL: `http://dx.doi.org/10.18419/opus-9930` (Zitiert auf S. 25, 53).

[HW18] C. B. Hauser, S. Wesner. „Reviewing Cloud Monitoring: Towards Cloud Resource Profiling“. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 2018, S. 678–685 (Zitiert auf S. 18, 25, 77, 81).

[HWBM16a] P. Hirmer, M. Wieland, U. Breitenbücher, B. Mitschang. „Automated Sensor Registration, Binding and Sensor Data Provisioning“. In: *CAiSE Forum*. 2016 (Zitiert auf S. 42, 117).

[HWBM16b] P. Hirmer, M. Wieland, U. Breitenbücher, B. Mitschang. „Dynamic Ontology-Based Sensor Binding“. In: *Advances in Databases and Information Systems*. 2016, S. 323–337 (Zitiert auf S. 146).

[HWS+16] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, S. G. Sáez, F. Leymann. „Situation recognition and handling based on executing situation templates and situation-aware workflows“. In: *Computing* (2016), S. 1–19 (Zitiert auf S. 23, 40, 110, 115).

[Ici21a] Icinga GmbH. *Icinga*. 2021. URL: https://icinga.com/ (Zitiert auf S. 57).

[Ici21b] Icinga GmbH. *Icinga Agent*. 2021. URL: https://icinga.com/docs/icinga2/latest/doc/07-agent-based-monitoring/#agent-based-checks-icinga (Zitiert auf S. 57).

[Inf21a] InfluxData. *InfluxDB*. 2021. URL: https://www.influxdata.com/products/influxdb-overview/ (Zitiert auf S. 99).

[Inf21b] InfluxData. *Telegraf*. 2021. URL: https://www.influxdata.com/time-series-platform/telegraf/ (Zitiert auf S. 57, 62, 73, 99).

[Inf21c] InfluxData. *TICK Stack*. 2021. URL: https://www.influxdata.com/time-series-platform/ (Zitiert auf S. 57, 73).

[KEW+10] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, M. Wolf. „Monalytics: online monitoring and analytics for managing large scale data centers“. In: *Proceedings of the 7th international conference on Autonomic computing*. 2010, S. 141–150 (Zitiert auf S. 25, 80).

[KKF+19] V. R. S. Kumar, A. Khamis, S. Fiorini, J. L. Carbonera, A. O. Alarcos, M. Habib, P. Goncalves, H. Li, J. I. Olszewska. „Ontologies for industry 4.0“. In: *The Knowledge Engineering Review* 34 (2019) (Zitiert auf S. 145).

[Kol17] J. Kolerus. *Zustandsüberwachung von Maschinen: Das Lehr-und Arbeitsbuch für den Praktiker*. expert Verlag, 2017 (Zitiert auf S. 19, 36).

[Kün+11] V. Künzle et al. „PHILharmonicFlows: Towards a Framework for Object-aware Process Management“. In: *Journal of Software Maintenance and Evolution: Research and Practice* (2011) (Zitiert auf S. 53).

[LCW08] D. Lucke, C. Constantinescu, E. Westkämper. „Smart Factory - A Step towards the Next Generation of Manufacturing“. In: *Manufacturing Systems and Technologies for the New Frontier: The 41st CIRP Conference on Manufacturing*. 2008, S. 115–118 (Zitiert auf S. 26, 109, 112).

[Lig12a] S. Ligus. *Effective Monitoring and Alerting*. " O'Reilly Media, Inc.", 2012, S. 2 (Zitiert auf S. 36).

[Lig12b] S. Ligus. *Effective Monitoring and Alerting*. " O'Reilly Media, Inc.", 2012, S. 17 (Zitiert auf S. 38).

[Lig12c] S. Ligus. *Effective monitoring and alerting*. " O'Reilly Media, Inc.", 2012, S. 7 (Zitiert auf S. 22, 38, 39).

[LK18] S. Luber, F. Karlstetter. *Was ist Database as a Service (DBaaS)?* 2018. URL: `https://www.cloudcomputing-insider.de/was-ist-database-as-a-service-dbaas-a-692502/` (Zitiert auf S. 16, 33).

[MDAM11] R. Mehrotra, A. Dubey, S. Abdelwahed, W. Monceaux. „Large scale monitoring and online analysis in a distributed virtualized environment". In: *2011 Eighth IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*. 2011, S. 1–9 (Zitiert auf S. 79).

[MG11] P. Mell, T. Grance. *The NIST definition of cloud computing*. 2011 (Zitiert auf S. 16, 31).

[MHSM20a] M. Mormul, P. Hirmer, C. Stach, B. Mitschang. „Avoiding Vendor-Lockin in Cloud Monitoring using Generic Agent Templates". In: *2020 International Conference on Business Information Systems (BIS)*. 2020 (Zitiert auf S. 28, 52).

[MHSM20b] M. Mormul, P. Hirmer, C. Stach, B. Mitschang. „DEAR: Distributed Evaluation of Alerting Rules". In: *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. 2020, S. 158–165 (Zitiert auf S. 29, 79).

[MHWM17] M. Mormul, P. Hirmer, M. Wieland, B. Mitschang. „Situation model as interface between situation recognition and situation-aware applications". In: *Computer Science-Research and Development* 32.3-4 (2017), S. 331–342 (Zitiert auf S. 30, 117, 144, 154).

[MHWM18] M. Mormul, P. Hirmer, M. Wieland, B. Mitschang. „Layered Modeling Approach for Distributed Situation Recognition in Smart Environments". In: *2018 7th International Conference on Smart Cities, Systems, Devices and Technologies*. IARIA. 2018, S. 47–53 (Zitiert auf S. 30, 112).

[MHWM19] M. Mormul, P. Hirmer, M. Wieland, B. Mitschang. „Distributed Situation Recognition in Industry 4.0“. In: *International Journal On Advances in Intelligent Systems* 12.1-2 (2019), S. 39–49 (Zitiert auf S. 30, 112).

[Mic20] Microsoft Corporation. *Understand the structure and syntax of ARM templates*. 2020. URL: https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-syntax (Zitiert auf S. 53).

[MLB+11] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, A. Ghalsasi. „Cloud computing — The business perspective“. In: *Decision Support Systems* 51.1 (2011), S. 176–189 (Zitiert auf S. 15, 126).

[Moo21a] Moogsoft. *AIOps Platform*. 2021. URL: https://www.moogsoft.com/aiops-platform/ (Zitiert auf S. 57).

[Moo21b] Moogsoft. *Observe Overview*. 2021. URL: https://docs.moogsoft.com/AIOps.7.1.0/Moogsoft-Observe_29953305.html (Zitiert auf S. 57, 82).

[Mor21] M. Mormul. *Schema Agententemplates*. 2021. URL: https://github.com/mormulms/agent-centric-monitoring/blob/master/generic-agent/mona-template-editor/src/assets/schema.json (Zitiert auf S. 66).

[MS20] M. Mormul, C. Stach. „A Context Model for Holistic IT Operations“. In: *2020 16th Workshop on Context Modeling and Activity Recognition (CoMoRea)*. 2020 (Zitiert auf S. 30, 144).

[MZP+17] O. Moll, A. Zalewski, S. Pillai, S. Madden, M. Stonebraker, V. Gadepally. „Exploring big volume sensor data with Vroom“. In: *Proceedings of the VLDB Endowment* 10.12 (2017) (Zitiert auf S. 116).

[Nag21a] Nagios Enterprises. *Nagios*. 2021. URL: https://www.nagios.com/ (Zitiert auf S. 49, 57, 82).

[Nag21b] Nagios Enterprises. *Nagios Cross Platform Agent*. 2021. URL: https://www.nagios.org/ncpa/ (Zitiert auf S. 57, 62).

[Nag21c] Nagios Enterprises. *Nagios Cross Platform Agent - Getting Started*. 2021. URL: https://www.nagios.org/ncpa/getting-started.php (Zitiert auf S. 49).

[NKFW19] S. Niedermaier, F. Koetter, A. Freymann, S. Wagner. „On Observability and Monitoring of Distributed Systems–An Industry Interview Study“. In: *International Conference on Service-Oriented Computing*. Springer. 2019, S. 36–52 (Zitiert auf S. 20, 27, 47, 143, 144).

[OAS13] OASIS Open. *Topology and Orchestration Specification for Cloud Applications Version 1.0*. 2013. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html (Zitiert auf S. 69).

[OGC21] OGC. *Sensor Model Language (SensorML)*. 2021. URL: https://www.ogc.org/standards/sensorml (Zitiert auf S. 145).

[Ope21] OpenStack Foundation. *OpenStack*. 2021. URL: https://www.openstack.org/ (Zitiert auf S. 100).

[Ora21] Oracle Corporation. *Using Monitoring Templates*. 2021. URL: https://docs.oracle.com/cd/E24628_01/doc.121/e24473/mon_temp.htm (Zitiert auf S. 53).

[OST14] J. Opara-Martins, R. Sahandi, F. Tian. „Critical review of vendor lock-in and its impact on adoption of cloud computing“. In: *International Conference on Information Society (i-Society 2014)*. 2014, S. 92–97 (Zitiert auf S. 24).

[Pan21a] Pandora FMS Team. *Pandora FMS*. 2021. URL: https://pandorafms.org/ (Zitiert auf S. 57).

[Pan21b] Pandora FMS Team. *Pandora FMS Software Agent*. 2021. URL: https://pandorafms.com/docs/index.php?title=Pandora:Documentation_en:Configuration_Agents#Pandora_FMS_Software_Agents (Zitiert auf S. 57).

[PB18] P. Prasad, V. Bhalla. *Use This 4-Step Approach to Architect Your IT Monitoring Strategy*. 2018. URL: https://www.gartner.com/en/documents/3882275/use-this-4-step-approach-to-architect-your-it-monitoring (Zitiert auf S. 70).

[Pet13] D. Petcu. „Multi-Cloud: expectations and current approaches“. In: *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. 2013, S. 1–6 (Zitiert auf S. 16, 34, 146).

[PG07] F. Paganelli, D. Giuli. „An Ontology-Based Context Model for Home Health Monitoring and Alerting in Chronic Patient Care Networks“. In: *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*. Bd. 2. 2007, S. 838–845 (Zitiert auf S. 146).

[Pla19] Plattform Industrie 4.0. *Asset Administration Shell: Umsetzung des digitalen Zwillings für Industrie 4.0*. 2019. URL: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/VWSiD%20V2.0.html (Zitiert auf S. 19, 35).

[Pre98] B. R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*. Wiley, 1998 (Zitiert auf S. 90).

[Pro21a] Prometheus Authors. *Exporters and Integrations*. 2021. URL: https://prometheus.io/docs/instrumenting/exporters/ (Zitiert auf S. 57).

[Pro21b] Prometheus Authors. *Prometheus*. 2021. URL: https://prometheus.io/ (Zitiert auf S. 57).

[Rob18] M. Roberts. *Serverless Architectures*. 2018. URL: https://martinfowler.com/articles/serverless.html (Zitiert auf S. 16, 33).

[RSM14] P. Reimann, H. Schwarz, B. Mitschang. „A pattern approach to conquer the data complexity in simulation workflow design“. In: *On the Move to Meaningful Internet Systems - 22nd International Conference on Cooperative Information Systems (CoopIS 2014)*. 2014, S. 21–38 (Zitiert auf S. 25, 53).

[SCZ+16] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu. „Edge computing: Vision and challenges“. In: *IEEE Internet of Things Journal* 3.5 (2016), S. 637–646 (Zitiert auf S. 19, 35, 111, 124).

[Sen21a] Sensu, Inc. *Sensu*. 2021. URL: https://sensu.io/ (Zitiert auf S. 57).

[Sen21b] Sensu, Inc. *Sensu Client*. 2021. URL: https://docs.sensu.io/sensu-core/1.4/installation/install-sensu-client/ (Zitiert auf S. 57).

[SKPR10] B. Schilling, B. Koldehofe, U. Pletat, K. Rothermel. „Distributed Heterogeneous Event Processing: Enhancing Scalability and Interoperability of CEP in an Industrial Context“. In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*. ACM, 2010, S. 150–159 (Zitiert auf S. 26, 113).

[SMP09] N. P. Schultz-Møller, M. Migliavacca, P. Pietzuch. „Distributed complex event processing with query rewriting“. In: *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*. 2009, S. 4 (Zitiert auf S. 26, 82, 113).

[Sol20] Solarwinds MSP. *Create a Monitoring Template*. 2020. URL: `https://documentation.solarwindsmsp.com/remote-management/helpcontents/mt_blank_template.htm` (Zitiert auf S. 53).

[SPG+15] C. Schlenoff, E. Prestes, P. S. Gonçalves, M. Abel, Y. Amirat, S. Balakirsky, M. Barreto, J. Carbonera, A. Chibani, S. R. Fiorini et al. „IEEE Standard Ontologies for Robotics and Automation“. In: (2015) (Zitiert auf S. 145).

[Spl21a] Splunk Inc. *Forwarder*. 2021. URL: `https://www.splunk.com/de_de/products/splunk-enterprise/features/forwarders.html` (Zitiert auf S. 57).

[Spl21b] Splunk, Inc. *Splunk*. 2021. URL: `https://www.splunk.com/` (Zitiert auf S. 57).

[STN18] S. Siami-Namini, N. Tavakoli, A. S. Namin. „A comparison of ARIMA and LSTM in forecasting time series“. In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2018, S. 1394–1401 (Zitiert auf S. 171).

[Stu21] U. Stuttgart. *IC4F*. 2021. URL: `https://www.ipvs.uni-stuttgart.de/de/abteilungen/as/forschung/projekte/IC4F/` (Zitiert auf S. 35, 44).

[Sun+14] Y. Sun et al. „Modeling Data for Business Processes“. In: *Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE), Chicago, USA*. 2014 (Zitiert auf S. 53).

[SWWM10] J. Shao, H. Wei, Q. Wang, H. Mei. „A runtime model based monitoring approach for cloud“. In: *2010 IEEE 3rd International Conference on Cloud Computing*. 2010, S. 313–320 (Zitiert auf S. 25, 82).

[Tec17] R. H. Technology. *All Things Are Digital In Business, But Finding Digital Talent Is A Tall Order*. Nov. 2017. URL: http://rh-us.mediaroom.com/2017-11-01-All-Things-Are-Digital-In-Business-But-Finding-Digital-Talent-Is-A-Tall-Order (Zitiert auf S. 18, 51, 86).

[TJT+18] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, V. Stankovski. „Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review“. In: *Journal of Systems and Software* 136 (2018), S. 19–38 (Zitiert auf S. 18, 69, 80).

[TPD15] D. Trihinas, G. Pallis, M. Dikaiakos. „Monitoring elastically adaptive multi-cloud services“. In: *IEEE Transactions on Cloud Computing* 3 (2015), S. 800–814 (Zitiert auf S. 17, 18, 22, 25, 49, 77, 80).

[Uni19] University of Stuttgart. *OpenTOSCA*. 2019. URL: https://www.opentosca.org/ (Zitiert auf S. 69).

[Uni21] Universität Stuttgart. *SitOPT*. 2021. URL: https://www.ipvs.uni-stuttgart.de/departments/as/research/projects/sitopt/ (Zitiert auf S. 40, 110).

[VT14] A. Varghese, D. Tandur. „Wireless requirements and challenges in Industry 4.0“. In: *2014 International Conference on Contemporary Computing and Informatics (IC3I)*. 2014, S. 634–638 (Zitiert auf S. 18, 80).

[WB14] J. S. Ward, A. Barker. „Observing the clouds: a survey and taxonomy of cloud monitoring“. In: *Journal of Cloud Computing* 3.1 (2014), S. 24 (Zitiert auf S. 18, 25, 37, 69, 78, 80, 82).

[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*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005 (Zitiert auf S. 53).

[Wei20] K. Weins. *Cloud Computing Trends: 2020 State of the Cloud Report*. 2020. URL: https://www.flexera.com/blog/industry-trends/trend-of-cloud-computing-2020/ (Zitiert auf S. 16, 146).

[Woo99] M. Wooldridge. „Intelligent agents“. In: *Multiagent systems* 35.4 (1999), S. 51 (Zitiert auf S. 38).

[WSBL15] M. Wieland, H. Schwarz, U. Breitenbücher, F. Leymann. „Towards situation-aware adaptive workflows: SitOPT – A general purpose situation-aware workflow management system“. In: *Pervasive Computing and Communication Workshops (PerCom Workshops)*. 2015, S. 32–37 (Zitiert auf S. 19, 26, 42, 110, 111, 151, 155).

[WST+11] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, M. Wolf. „A Flexible Architecture Integrating Monitoring and Analytics for Managing Large-scale Data Centers“. In: *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*. 2011, S. 141–150 (Zitiert auf S. 25, 80).

[WZGP04] X. H. Wang, D. Q. Zhang, T. Gu, H. K. Pung. „Ontology based context modeling and reasoning using OWL“. In: *IEEE Annual Conference on Pervasive Computing and Communications Workshops*. 2004, S. 18–22 (Zitiert auf S. 145).

[YHQL15] S. Yi, Z. Hao, Z. Qin, Q. Li. „Fog computing: Platform and applications“. In: *Hot Topics in Web Systems and Technologies (HotWeb)*. 2015, S. 73–78 (Zitiert auf S. 127).

[YWJ+15] O. N. Yilmaz, Y.-P. E. Wang, N. A. Johansson, N. Brahmi, S. A. Ashraf, J. Sachs. „Analysis of ultra-reliable and low-latency 5G communication for a factory automation use case“. In: *2015 IEEE international conference on communication workshop (ICCW)*. 2015, S. 1190–1195 (Zitiert auf S. 111, 116).

[Zab21a] Zabbix LLC. *Zabbix*. 2021. URL: https://www.zabbix.com/ (Zitiert auf S. 57).

[Zab21b] Zabbix LLC. *Zabbix Agent*. 2021. URL: https://www.zabbix.com/zabbix_agent (Zitiert auf S. 57).

[ZZL17] C. Zhang, G. Zhou, Q. Lu. „Decision support oriented ontological modeling of product knowledge“. In: *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. 2017, S. 39–43 (Zitiert auf S. 145).

Alle URLs wurden zuletzt am 05.02.2022 geprüft.

Abbildungsverzeichnis

Tabellenverzeichnis