# Task Allokation für effiziente Edge Computing Systeme

Zur Erlangung des akademischen Grades eines

**DOKTORS DER INGENIUERWISSENSCHAFTEN (Dr.-Ing.)**

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik des
Karlsruher Institut für Technologie (KIT)
genehmigte

**DISSERTATION**

von

**M.Sc. Victor Hugo Pazmino Betancourt**

geboren in Quito - Ecuador

Tag der mündlichen Prüfung:
28.02.2023

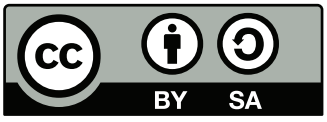
Hauptreferent: Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent: Prof. Dr.-Ing. Jörg Henkel

Version: 1.0.0

Stand: 29.04.2023

**Task Allokation für effiziente**
**Edge Computing Systeme**

1. Auflage: April 2023

*„Nemo nascitur sapiens, sed fit - Keiner wird weise geboren, er muß es erst werden - Nadie nace sabio, sino que se hace“*

*Seneca*

# Abstract

In the field of edge computing, the computational power in close proximity to sensors is constantly increasing. As a result, a growing number of compute- and data-intensive applications are able to run at the edge. At the same time, they are in a constantly changing system environment.

To reduce downtime and long redesign loops self-adaptation capabilities are needed. Automatic reallocation of the executed tasks to the compute nodes is a possible self-adaptation measure. However, the reallocation should be compliant with the different demands, constraints and specifications from the design. At the same time a major challenge is that the allocation decision should be fast enough for calculation at runtime. The focus of this work is to realize efficient allocation that uses demands in the form of policies to compute automatic reallocation at runtime.

In this work, an efficient allocation method was developed that combines consideration of resource availability, application demand, and problem-specific efficiency definition. The approach follows a modular description of these aspects for the allocation in the form of component-specific policies. A special focus is on the allocation due to changes during operation. For this purpose, the allocation problem is modeled at development time and the information is used in operation.

With this concept, two industrial applications were modeled and different allocations were calculated. The scalability of the concept was validated by measurements. The reallocation at runtime was implemented using a container framework. Based on this, the overhead of the allocation calculation at runtime was measured and put into the context of the reallocation time.

Efficient allocation of computational power contributes to the autonomy of computing infrastructures. This increases the ability for self-adaptation and the resilience of these computing networks. This plays a role not only in the industrial edge-cloud context, but also in automotive applications when dynamic operational strategies are to be decided at runtime.

# Zusammenfassung

Im Bereich von Edge Computing nimmt die Rechenleistung in direkter Nähe zu den Sensoren stetig zu. Infolgedessen gibt es immer mehr rechen- und datenintensive Anwendungen, die im Edge Bereich ausgeführt werden können. Gleichzeitig befinden sie sich in einer sich ständig verändernden Systemumgebung.

Um Ausfallzeiten und lange Redesign-Schleifen zu reduzieren, werden Selbstanpassungsfähigkeiten benötigt. Die automatische Reallokation der ausgeführten Aufgaben auf die Rechenknoten ist eine mögliche Selbstanpassungsmaßnahme. Die Reallokation sollte jedoch mit den verschiedenen Anforderungen, Einschränkungen und Spezifikationen des Entwurfs konform sein. Dabei besteht eine große Herausforderung darin, dass die Allokationsentscheidung schnell genug für die Berechnung zur Laufzeit sein sollte. Der Fokus dieser Arbeit ist die Realisierung einer effizienten Allokation, die Bedürfnisse in Form von Policies nutzt, um eine automatische Reallokation zur Laufzeit zu berechnen.

In dieser Arbeit wurde eine effiziente Allokationsmethode entwickelt, die eine kombinierte Betrachtung von Ressourcenverfügbarkeit, Anwendungsbedarf und problemspezifischer Effizienzdefinition realisiert. Der Ansatz verfolgt eine modulare Beschreibung dieser Aspekte für die Allokation in Form von komponentenspezifischen Policies. Ein besonderer Schwerpunkt liegt auf der Allokation aufgrund von Veränderungen im laufenden Betrieb. Hierfür wird das Zuordnungsproblem zur Entwicklungszeit modelliert und die Informationen im Betrieb genutzt.

Mit diesem Konzept konnten zwei industrielle Anwendungen modelliert und unterschiedliche Zuordnungen berechnet werden. Die Skalierbarkeit des Konzepts wurde durch Messungen validiert. Die Reallokation zur Laufzeit wurde mit einem Container Framework implementiert. Darauf aufbauend wurde der Overhead der Allokationsberechnung zur Laufzeit gemessen und in den Kontext der Reallokationszeit gesetzt.

Die Berechnung einer effizienten Allokation trägt zur Autonomie von Recheninfrastrukturen bei. Dadurch erhöhen sich die Fähigkeiten zur Selbstadaption und die Resilienz

dieser Rechennetze. Das spielt nicht nur im industriellen Edge-Cloud Kontext eine Rolle, sondern auch im Automobil, wenn zur Laufzeit über dynamische Betriebsstrategien entschieden werden soll.

# Vorwort

In den zurückliegenden spannenden Jahren als wissenschaftlicher Mitarbeiter und Abteilungsleiter am FZI Forschungszentrum Informatik ist die vorliegende Arbeit entstanden. In dieser Zeit hatte ich die Möglichkeit, viele tolle Menschen kennenzulernen und Kontakte zu knüpfen. Ebenso bekam ich einen wertvollen Einblick in verschiedene Forschungsthemen, Industriekooperationen und Verantwortungsaufgaben. Ich denke sehr gerne an diese schöne und prägende Zeit zurück und möchte mich bei allen bedanken, die mich auf diesem Weg begleitet haben.

Zunächst möchte ich mich ganz besonders bei meinem Doktorvater Prof. Jürgen Becker bedanken, der mir die Promotion am KIT ermöglicht hat und mir in den vergangenen Jahren die Leitung seiner Gruppe am FZI anvertraut hat. Er unterstützte mich immer in meinen Plänen und gab mir den Freiraum, um meine eigenen Ideen zu verfolgen und zu verwirklichen. An seiner Seite habe ich sowohl fachlich als auch menschlich viel gelernt und mich weiterentwickelt. Er ist für mich zu einem Vorbild geworden und ich verstehe nun, was er mit den Worten akademischer Vater meint.

Ebenso möchte ich mich bei Prof. Jörg Henkel für die Übernahme des Korreferats bedanken. Vielen Dank auch an die weitere Prüfungskommission, bestehend aus Prof. Sören Hohmann, Prof. Mike Barth und Prof. Ahmet Cagri Ulusoy.

Ebenfalls bedanken möchte ich mich bei meinen einzigartigen Kollegen am FZI und ITIV, die mich über mehrere Jahre hinweg begleitet haben und jeden Tag mit neuen Ideen, konstruktiver Kritik und anderen Perspektiven bereichert haben. Besonders erwähnen möchte ich meine Abteilung Embedded Computing Systems (ECS) und meine Kollegen, die mich in der letzten Phase der Dissertation auf vielfältige Weise unterstützt haben: Konstantin Dudzik, Maximilian Kirschner, Marius Kreutzer, Sven Nitzsche, Brian Pachideh, und Alexandru Vasilache. Aber auch die Kollegen, die nicht mehr am FZI sind, mich aber in den letzten Jahren begleitet haben: Thomas Glock, Manuel Härdle, Matthias Kern, Nadir Khan, Jochen Kramer. Besonderer Dank gilt Bo Liu als langjährigem Kollegen in Forschungsprojekten am FZI, der mich immer unterstützt hat, wenn angebracht, Kritik geübt

hat und mit dem wir neue Themen erschlossen haben. Ich habe von allen viel gelernt und bin sehr stolz auf das Team, das wir zusammen aufgebaut haben.

Ein weiterer Dank gilt meinen Abteilungsleiterkollegen aus dem Bereich Embedded Systems and Sensors Engineering, die immer Verständnis für die Arbeitsbelastung dieser Doppelrolle hatten und mit denen wir die Herausforderungen sowohl fachlich als auch organisatorisch stets gemeistert haben. Auch ein herzlicher Dank an Thomas Bruckschlögl, der mich nicht nur zu Beginn meiner Promotion unterstützt hat, sondern auch meinen Studienweg im Bachelor und Master geprägt und mir die Möglichkeit gegeben hat, meine Fähigkeiten zu zeigen.

Darüber hinaus möchte ich mich bei allen bedanken, die zur Entstehung dieser Arbeit beigetragen haben, sei es durch Anregungen, Diskussionen oder auch durch Aufmunterungen. Insbesondere möchte ich Elke Weber, Alena Weber und Nicole Pazmino für ihre Unterstützung beim Korrekturlesen und Feinschliff dieser Arbeit danken. Darüber hinaus möchte ich mich auch bei den Studenten bedanken, die ich während meiner Promotionszeit betreuen durfte und die zu verschiedenen Fragestellungen beigetragen haben.

Nicht zuletzt geht ein ganz besonderer Dank an meine Familie, vor allem an meine Eltern Azucena und Patricio, die mich von Ecuador aus immer unterstützen, sowie an meine Geschwister Nicole und Patricio und meine Cousine Camila, die mich jeden Tag fit halten. Außerdem gilt ein spezieller Dank meiner Freundin Dr. Alena Weber und ihren Eltern Elke und Thomas, die der Grund dafür sind, dass ich mich in Deutschland integriert fühle und hier geblieben bin. Mit Alena haben wir gemeinsam jedes Ziel erreicht, das wir uns gesetzt haben, seit wir uns im Elektrotechnikstudium kennengelernt haben. Sie steht mir bei jeder Herausforderung mit unermüdlicher Geduld und Verständnis zur Seite: Sie hat nie an mir gezweifelt, und für ihre unerschöpfliche Unterstützung und Liebe bin ich ihr von ganzem Herzen dankbar!

Deshalb möchte ich mich bei meinem persönlichen Team bedanken, das mich am Laufen hält und für mich Teil dieser Leistung ist. Vor etwa 12 Jahren kam ich aus Ecuador auf ein Abenteuer nach Deutschland, um zu studieren. Heute, mit 29 Jahren, blicke ich zurück, ich habe mehr als ein Drittel meines Lebens in Deutschland verbracht. Nach dem Bachelor, dem Master und jetzt der Promotion am KIT habe ich nicht vergessen, woher ich komme. Ich habe meinen Eltern zu danken, vor allem für die Bildung, die sie mir gegeben haben, und das Lebensmotto, das mich durch alle Herausforderungen bringt: Nadie nace sabiendo - Niemand wird wissend geboren. Das ist der innere Antrieb, der mich von einem Ziel zum nächsten bringt.

Karlsruhe, im April 2023

Victor Hugo Pazmino Betancourt

# Inhaltsverzeichnis

# Teil I

# Einführung

# Kapitel 1

# Einleitung

In diesem Kapitel werden die Motivation und die Zielsetzung für diese Arbeit erläutert sowie der Aufbau der Arbeit beschrieben.

## 1.1 Motivation und Umfeld

Im Bereich von Edge Computing nimmt die Rechenleistung in direkter Nähe zu den Sensoren stetig zu. Infolgedessen können im Edge Bereich mehr und mehr rechen- und datenintensive Anwendungen ausgeführt werden [1]. Die Edge Computing Infrastruktur unterscheidet sich von der Cloud Infrastruktur. So sind beispielsweise die Recheneinheiten in der Regel mit einem geringeren Energiebudget oder einer geringeren Leistung ausgestattet und hinsichtlich der Art ihrer Prozessorkomponenten heterogenen.

**Anwendungsspezifische Effizienz:** Im Edge Kontext sind Aspekte wie Energieeffizienz, Echtzeitfähigkeit, Zuverlässigkeit, Verfügbarkeit oder Ausfallsicherheit von zunehmender Bedeutung. Die Ausprägung dieser Aspekte hängt jedoch von der spezifische industriellen Anwendung und dem Betriebskontext ab [2]. Zum Beispiel liegt bei der Objekterkennung in der Robotik der Fokus auf einer möglichst geringen Latenzzeit, während bei batteriebetriebenen, fahrerlosen Transportsystemen der Fokus auf einem geringen Stromverbrauch liegt. Das bedeutet, dass die Effizienz eines Edge Computing Systems anwendungsspezifisch definiert werden muss.

**Veränderungen:** Zudem sind dies keine abgeschlossenen Anwendungen und Systeme, vielmehr werden sie im Laufe der Zeit weiterentwickelt und unterliegen Veränderungen. Die dazugehörige Datenverarbeitungsinfrastruktur muss in der Lage sein, auf die Veränderungen zu reagieren und sich anzupassen. Änderungen treten beispielsweise auf, wenn neue Funktionen im Laufe der Zeit zum System hinzugefügt werden, wenn Funktionen ausfallen oder aufgrund von Securityproblemen oder Betriebssituationen verlagert werden müssen [3], wie in Abbildung 1.1 dargestellt.

Aus der analysierten Literatur geht hervor, dass selbstadaptive Recheninfrastrukturen im Edge für verschiedene Bereiche wie die industrielle Fertigung, autonome Maschinen oder intelligente Verkehrsinfrastrukturen zunehmend an Bedeutung gewinnen. Da im Edge Kontext viele Anwendungen auf heterogenen Recheneinheiten ausgeführt werden müssen, ist der Aspekt der effizienten Zuweisung von Funktionen eine große Herausforderung. Besonders, wenn es darum geht, im laufenden Betrieb autonom auf Veränderungen zu reagieren.

## 1.2 Zielsetzung und Beitrag der Arbeit

### Thema und Zielsetzung

Im Rahmen dieser Arbeit wurde daher der Frage nachgegangen, wie eine automatische und effiziente Allokation realisiert werden kann. Das wurde in die folgenden Teilfragestellungen untergliedert:

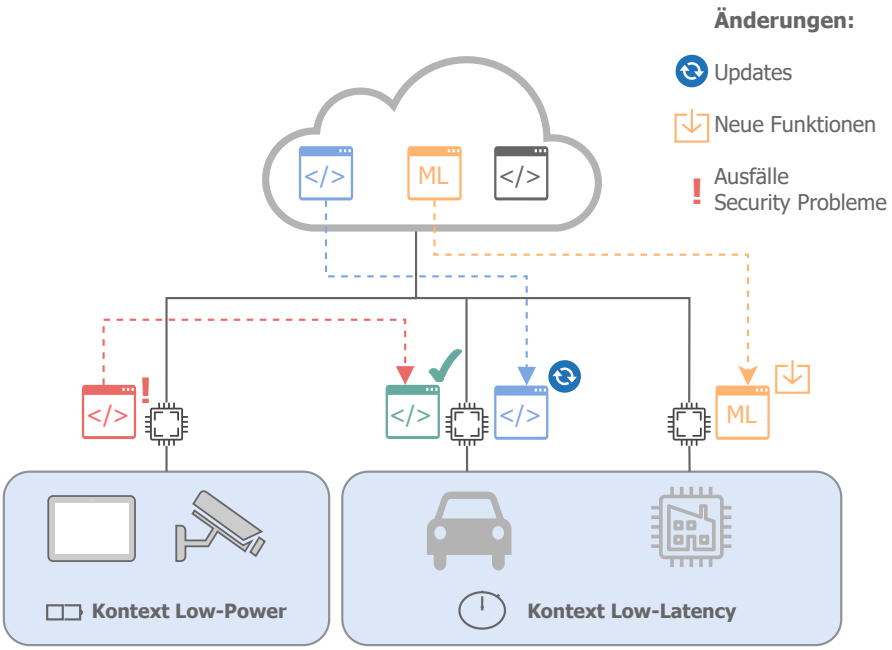



Abbildung 1.1: Veränderungen im Edge Computing Kontext. Schematische Darstellung von Updates (blau), neuen Funktionen (gelb) und Ausfälle (rot).

1. Welche Informationen sind für die Allokationsberechnung notwendig?

2. Welche Algorithmen können eingesetzt werden, um die Allokation leichtgewichtig im Betrieb zu berechnen?

3. Wie kann eine effiziente Allokation in den Betrieb integriert werden?

Das Ziel dieser Arbeit ist somit die automatische und effiziente Zuweisung zur Laufzeit von Anwendungen auf Recheneinheiten im Edge Computing Kontext. Auf der Basis kann die automatische Reaktion und Selbstadaption der Datenverarbeitungsinfrastruktur ermöglicht werden, wodurch flexible und resiliente Systeme in Edge implementiert werden können. Die folgenden Punkte wurden bearbeitet:

- Ein *modellgetriebener Entwicklungsprozess für Edge Computing Systeme*, der die kombinierte Berücksichtigung von Ressourcenverfügbarkeit, Anwendungsbedarf und problemspezifischer Effizienzdefinition während des Entwurfs und Betriebs ermöglicht.

- Eine *leichtgewichtige Allokationsmethode*, die anhand von Informationen aus Design und Betrieb auf Veränderungen reagieren kann, um eine effiziente Allokation in einem dynamischen Betrieb zu berechnen.

- Ein *containerbasiertes Edge Framework* zur Integration der Allokationsmethode in ein laufendes System sowie zur Durchführung von Neuzuordnungen.

### Beitrag der Arbeit

Diese Arbeit leistet mit der entwickelten Methode zur effizienten Allokation einen Beitrag zur Entwicklung von autonomen und resilienten Recheninfrastrukturen im Edge Computing Kontext. Die Recheninfrastruktur kann so auf Veränderungen reagieren und sich selbst adaptieren, so dass die Verfügbarkeit der Systeme im Edge Bereich erhöht wird und gleichzeitig die manuelle Verwaltung der Infrastruktur reduziert wird. Das spielt nicht nur im industriellen Edge-Cloud Kontext eine Rolle, sondern auch bspw. in intelligenten Smart City Infrastrukturen oder in E/E-Architekturen für autonome Fahrzeuge.

Die in dieser Arbeit untersuchten industriellen Anwendungsfälle sind: (i) das Offloading der Perzeption von fahrerlosen Transportfahrzeugen in der Fabrikumgebung und (ii) die Veränderungen im System bei der Personenerkennung in der Umgebung von kollaborativen Arbeitsmaschinen.

Insgesamt wurden die folgenden Beiträge geleistet:

- *Erweiterung eines Modellierungstools und Prozesses* für kontinuierliche Entwicklung von Edge Systemen durch die Integration von Konzepten für die Allokationsbeschreibung, Adaption und Reaktion bei Veränderungen im System.
- *Entwicklung eines Policy Beschreibungskonzepts* zur Modellierung der notwendigen Informationen für eine effiziente Allokation, welches eine komponentenspezifische Beschreibung der Bedarfe, Optimierungsziele und Beschränkungen auf unterschiedlichen Ebenen in einer Schichtenarchitektur umfasst.
- *Entwicklung einer leichtgewichtigen Allokationsmethode* für die automatische Entscheidung über die Zuordnung von Funktionen auf Recheneinheiten im laufenden Betrieb, welche die Information des Policy Beschreibungskonzeptes und einen leichtgewichtigen Constraint Programming Solver verwendet. So kann die Allokationsmethode eine realisierbare und effiziente Lösung mit einem minimalen Rechenaufwand finden. Die Effizienz der Lösung wird durch die Einhaltung und Optimierung der beschriebenen komponentenspezifischen Policies sichergestellt. Die Skalierbarkeit der entwickelten Allokationsmethode wird durch den geringem Rechenaufwand und einen mehrstufigen Ressourcenverwaltungsansatz ermöglicht. Die Stufen werden von verteilter über dezentraler bis hin zu zentraler Ressourcenverwaltung durchlaufen.
- *Erweiterung eines Edge Frameworks* für die Verschiebung von Funktionen als Reaktion auf Veränderungen. Hierbei wird die Allokationsmethode unter Verwendung einer servicebasierten Architektur, Container Virtualisierung und Orchestrierungslösungen integriert. So kann das Edge Framework mittels der entwickelten leichtgewichtigen Allokationsmethode über die Verschiebung von Funktionen im laufenden Betrieb effizient entscheiden und diese umsetzen.

## 1.3 Aufbau der Arbeit

Die vorliegende Arbeit untersucht die automatische und effiziente Zuordnung von Anwendungen zu Recheneinheiten im Edge Computing Kontext und ist in drei Teile mit neun Kapiteln gegliedert.

Der erste Teil gibt eine Einführung in die Arbeit. In Kapitel 1 wird das Thema motiviert und die Zielsetzung sowie die Beiträge der Arbeit werden aufgeführt. In Kapitel 2 werden die zum Verständnis notwendigen Grundlagen erläutert. Es werden Begriffe definiert und Grundlagen zu selbstadaptiven Systemen, modellgetriebener Entwicklung, Ressourcenmanagement, Optimierungsalgorithmen und Technologieplattformen für Edge Computing erklärt. In Kapitel 3 werden aktuelle Ansätze zur Ressourcenverwaltung, Funktionsverteilung und Technologien im Kontext von Edge Computing analysiert.

Der zweite Teil beschreibt die in dieser Arbeit entwickelten Ansätze für ein effizientes und leichtgewichtiges Allokationsverfahren. In Kapitel 4 wird die Erweiterung eines Modellierungswerkzeugs und Prozesses zur kontinuierlichen Entwicklung von Edge Systemen durch die Integration von Konzepten zur Allokationsbeschreibung erläutert. In Kapitel 5 wird die Entwicklung des Policy Beschreibungskonzepts zur Modellierung der notwendigen Informationen für eine effiziente Allokation dargestellt. Es beinhaltet die Analyse und Formalisierung des Allokationsproblems, der Effizienz sowie den Entwurf eines Policy Metamodells. In Kapitel 6 wird die Entwicklung der leichtgewichtigen Allokationsmethode für die automatische Entscheidung über die Zuordnung von Funktionen zu Recheneinheiten im laufenden Betrieb beschrieben. Es beinhaltet die Analyse des Tradeoffs zwischen Optimierung und Rechenaufwand sowie die Bedeutung von Leichtgewichtigkeit und Ansätze zur Modularisierung und Skalierung der Allokationsberechnung. In Kapitel 7 wird der Entwurf eines Edge Frameworks für die Verlagerung von Funktionen als Reaktion auf Veränderungen im System vorgestellt. Hierbei wird die Allokationsmethode unter Verwendung einer servicebasierten Architektur, Container Virtualisierung und Orchestrierungslösungen integriert.

Der dritte Teil beinhaltet die Evaluation und Schlussfolgerung der Arbeit. Kapitel 8 zeigt die untersuchten industriellen Anwendungsfälle, die zur Evaluierung der entwickelten Allokationsmethode herangezogen wurden. Die erste Anwendung beschäftigt sich mit den Veränderungen im System bei der Personenerkennung im Umfeld von kollaborativen Arbeitsmaschinen. Die zweite Anwendung beschäftigt sich mit dem Offloading der Perzeption von fahrerlosen Transportfahrzeugen in einer Fabrikumgebung. In Kapitel 9 werden die erzielten Ergebnisse zusammengefasst und diskutiert.

# Kapitel 2

# Grundlagen

In diesem Kapitel werden die zum Verständnis notwendigen Grundlagen erläutert. Es werden Begriffe definiert und die Grundlagen von selbstadaptiven Systemen, modellgetriebener Entwicklung, Ressourcenmanagement und Technologieplattformen für Edge Computing erklärt.

## 2.1 Edge Computing und selbstadaptive Systeme

### 2.1.1 Edge Computing

Verbesserte Verarbeitungsmöglichkeiten in kleinsten Rechenknoten, zunehmende Konnektivität und Konzepte wie das Internet der Dinge (IoT) eröffnen Chancen für ein neues Datenverarbeitungsparadigma, das Edge Computing, das in seinem Kern die Verarbeitung von Daten am Rande des Netzes stärkt. Edge Computing hat das Potenzial, Anforderungen in Bezug auf Reaktionszeit, Batterielebensdauer, Reduzierung der Bandbreitenkosten sowie Datensicherheit und Datenschutz zu erfüllen [1] . In diesem Abschnitt werden die Grundlagen und notwendigen Definitionen zum Thema Edge Computing aufgezeigt, sowie Anwendungsbereiche, die das Konzept des Edge Computing veranschaulichen.

Der Begriff Internet der Dinge (engl. Internet of Things, IoT) wurde erstmals 1999 für das Lieferkettenmanagement vorgestellt [4] und wurde auf das Konzept ausgeweitet, einen Computer dazu zu bringen, ohne menschliches Zutun Informationen in anderen Bereichen wie Gesundheitswesen, Haushalt, Umwelt und Verkehr zu sammeln. Seitdem sind verschiedene Entwicklungen zu beobachten, die eine Post-Cloud-Ära einläuten, in der eine große Menge an Daten von Dingen erzeugt wird, die überall eingebettet werden können und zahlreiche Anwendungen am Rande des Netzes einsetzen. Schätzungsweise 50 Milliarden Dinge sollten 2020 mit dem Internet verbunden sein, berichtete die Cisco Internet Business Solutions Group.

Zunehmend werden Daten am Rande des Netzes erzeugt, so dass es effizienter wäre, die Daten auch dort zu verarbeiten. Daher wurden in der Vergangenheit andere Datenverarbeitungsparadigmen wie Micro Datacenter, Cloudlet und Fog Computing eingeführt, da Cloud Computing für die Datenverarbeitung nicht immer geeignet ist. Shi et al [1] erläutern einige Gründe, warum das Edge Computing Paradigma notwendig ist:

*Grenzen des Cloud-Computing-Paradigmas:* Die Verlagerung aller Datenverarbeitungsaufgaben in die Cloud ist zwar ein möglicher Weg der Datenverarbeitung, da die Rechenleistung in der Cloud enorm und skalierbar ist. Die Netzwerkbandbreite skaliert jedoch nicht mit. Wenn die Datenmenge am Rande des Netzes wächst, werden sowohl die Geschwindigkeit als auch die Energiekosten des Datentransports zu einem Engpass. So wird beispielsweise erwartet, dass ein autonomes Fahrzeug jede Sekunde ein Gigabyte an Daten erzeugt, die in Echtzeit verarbeitet werden müssen, um korrekte Entscheidungen treffen zu können. Wollte man die Daten zur Verarbeitung in die Cloud schicken, wäre die Reaktionszeit zu lang und die Netzwerkbandbreite und -zuverlässigkeit nicht ausreichend, um eine große Anzahl von Fahrzeugen zu unterstützen. Stattdessen bietet es sich an, die Daten am Rande des Netzes zu verarbeiten, um die Reaktionszeit zu verkürzen, die Verarbeitung effizienter zu gestalten und die Netzlast zu verringern.

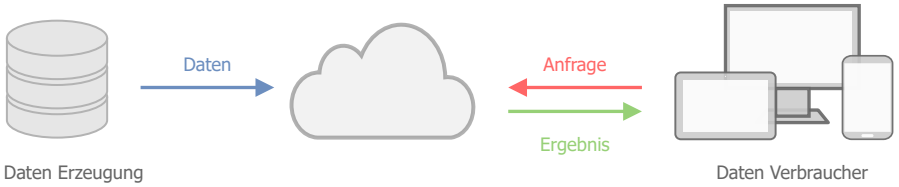



Abbildung 2.1: Cloud Computing Struktur

*(Industrial) IoT-Aspekte:* Fast alle Arten von elektrischen Geräten werden Teil des IoT, und sie werden sowohl die Rolle von Datenproduzenten als auch -konsumenten spielen, wie z. B. Straßenkreuzungen, Luftqualitätssensoren und insbesondere Geräte im industriellen Sektor, wie Kameras, Prozesssensoren usw. Die von ihnen erzeugten Rohdaten sind enorm, so dass die meisten der erzeugten Daten, insbesondere im industriellen Umfeld, nicht in die Cloud übertragen werden können, sondern im Edge verarbeitet werden, wodurch neue Möglichkeiten für verschiedene Anwendungen im Edge entstehen.

Abbildung 2.1 zeigt die traditionelle Cloud-Computing-Struktur, bei der Datenquellen Rohdaten generieren und in die Cloud übertragen. Anwendungen, die diese Daten benötigen, senden eine Anfrage zur Datennutzung an die Cloud und das Ergebnis wird zurückgemeldet.

Edge Computing umfasst somit ein Datenverarbeitungsparadigma und Technologien, die es ermöglichen, Berechnungen am Rande des Netzes durchzuführen. Dabei werden sämtliche Rechen- und Netzwerkressourcen auf dem Weg zwischen Datenquellen und Cloud-Rechenzentren als Edge definiert [1]. Ein Kamerasystem ist zum Beispiel die Grenze zwischen der Umgebung und der Cloud, dazwischen wird die Umgebung erfasst, Objekte werden erkannt und Anwendungen können darauf basierend Entscheidungen für das Handeln von Arbeitsmaschinen in dieser Umgebung ableiten. Grundidee von Edge Computing ist, dass die Datenverarbeitung in der Nähe der Datenquellen stattfinden soll. Andere Paradigmen wie Fog Computing sind ähnlich , wobei Edge Computing sich mehr auf die Seite der Datenquellen konzentriert, während Fog Computing sich mehr auf die Kommunikationsinfrastruktur in der Nähe der Cloud konzentriert.

Abbildung 2.2 zeigt die Datenflüsse und Ebenen beim Edge-Computing, wobei die Geräte sowohl Datenkonsumenten als auch Datenproduzenten sind. Im Edge können die Geräte nicht nur Dienste und Inhalte aus der Cloud anfordern, sondern auch Rechenaufgaben ausführen. Dies bedeutet, dass die Datenverarbeitung und -speicherung in den Edge Bereich verlagert werden kann. Edge Systeme müssen bedarfsgerecht konzipiert und konfiguriert werden, um die Anforderungen an die Services wie Zuverlässigkeit, Sicherheit und Schutz der Privatsphäre effizient zu erfüllen.

*Vorteile von Edge Computing:* Die Datenverarbeitung wird in die Nähe der Datenquellen verlegt. Dies hat mehrere Vorteile, wie verschiedene Forschungsarbeiten aufgezeigt haben.

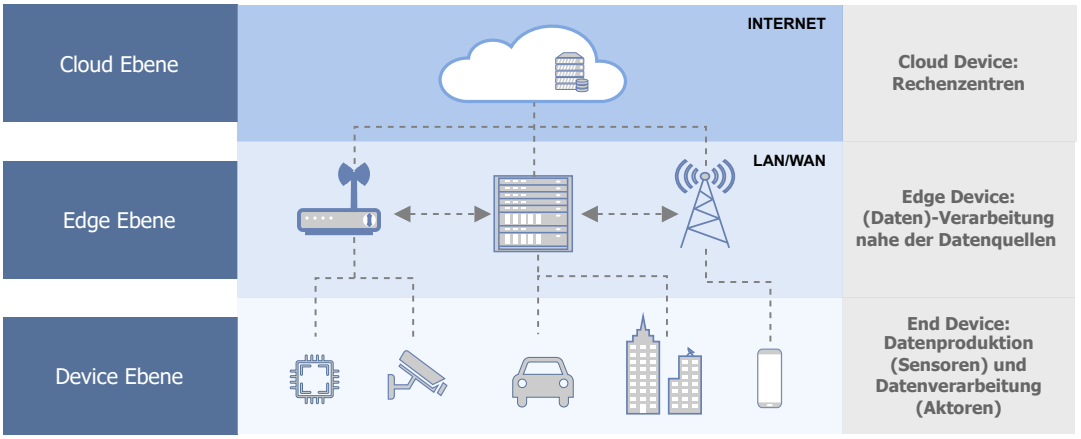



Abbildung 2.2: Edge Computing Struktur

So konnte beispielsweise die Rechenzeit einer Gesichtserkennungsanwendung in [5] von 900 ms auf 169 ms reduziert werden, indem die Berechnung in den Edge-Bereich verlagert wurde. In [6] wurde eine kombinierte Partitionierung, Migration und on-demand Instanziierung von Rechenaufgaben zwischen mobilen Geräten und der Cloud getestet, wodurch Laufzeit und Energieverbrauch um den Faktor 20 reduziert werden konnten.

Einsatzgebiete für Edge Computing sind unter anderem Cloud Offloading, Videoanalyse, Smart Home, Smart City oder vernetztes autonomes Fahren. Im industriellen Bereich spielen intelligente Umgebungen eine zentrale Rolle, wie z.B. intelligente Produktionsanlagen, Fabrikgelände, Baustellen und Smart Grids. Daher werden in dieser Arbeit die Konzepte anhand von zwei industriellen Anwendungen analysiert und evaluiert. Die Funktionsauslagerung von Arbeitsmaschinen im Kontext einer intelligenten Baustelle und das Offloading von Perzeptionsfunktionen für vernetzte Mikromobile im Kontext eines intelligenten Werksgeländes.

### 2.1.2 Selbsadaptive Systeme

Seit einigen Jahren nimmt die Komplexität von vernetzten Systemen, z.B. im Edge Bereich, zu, was zu einem erhöhten Aufwand für Wartung und Konfiguration führt. Selbstadaptive Systeme (engl. self-adaptive systems, SAS) adressieren dieses Problemfeld, indem die Geräte in der Lage sind, sich an den jeweiligen Kontext anzupassen. In diesem Abschnitt wird die Definition von SAS für den Kontext dieser Arbeit erläutert. Es werden die Dimensionen der Selbstadaption sowie Engineering Ansätze aufgezeigt.

Mit der zunehmenden Zahl eingebetteter Geräte und drahtloser Breitbandnetze erwarten die Betreiber und Nutzer, dass die Systeme jederzeit und überall funktionieren. Systeme im Edge Bereich sind hochgradig verteilt und müssen verfügbare, hochspezialisierte und heterogene Geräte (von eingebetteten Sensorknoten bis zu On-Premise Edge Servern)

und Datenströme (einschließlich IIoT-Dienste und Echtzeit-Sensordaten) integrieren. Dabei befinden sie sich in einer sich ständig verändernden Umgebung mit schwankenden Netzwerkressourcen und Verfügbarkeiten [3] . Die Entwicklung, Konfiguration und Wartung solcher Systeme ist eine anspruchsvolle, fehleranfällige und zeitaufwändige Aufgabe. Selbstadaption ist ein vielversprechender Ansatz, um diesen Aufwand zu reduzieren. Ein selbstadaptives System (SAS) ist in der Lage, sich selbst automatisch zu modifizieren, um auf Veränderungen in seiner Betriebsumgebung zu reagieren [7], [8]. Dazu gibt es verschiedene Dimensionen und Herangehensweisen; im Allgemeinen geht es um die Anpassung von Attributen oder Artefakten des Systems als Reaktion auf Veränderungen. Selbstadaptive Systeme bieten so genannte Self-x- oder Selbstmanagement-Merkmale wie Selbstkonfiguration, Selbstheilung im Falle von Ausfällen, Selbstoptimierung und Selbstschutz gegen Bedrohungen [8], [9].

Somit kann ein selbstadaptives System wie folgt definiert werden: „Ein selbstanpassendes System ist in der Lage, sein Verhalten oder seine Umgebung als Reaktion auf Veränderungen im Betriebsumfeld zu ändern. Das Betriebsumfeld umfasst alles, was beobachtbar ist, wie z. B. Eingaben des Endbenutzers, Hardware-Geräte, den umgebenden Kontext oder Programminstrumentierung“ [3].

Die Aspekte und Dimensionen der Selbstadaption wurden im Laufe der Jahre in verschiedenen Arbeiten analysiert und eigene Klassifikationen und Taxonomien erstellt. Für diese Arbeit folgen wir den Definitionen aus [3]. Dies beinhaltet Dimensionen, die für Systeme im Edge Bereich relevant sind, wie in Tabelle 2.1 zu sehen ist. Diese Fragen werden im Folgenden aufgelistet und im Zusammenhang mit der vorliegenden Arbeit kommentiert:

- *Wann soll die Adaption erfolgen?* Nach Änderungen im System oder im Kontext sogenannte On-Demand Ereignisse, siehe Kapitel 4.1.
- *Warum soll die Adaption erfolgen?* Um die Verfügbarkeit zu erhöhen und Ausfallzeiten aufgrund von Änderungen zu reduzieren.
- *Wo soll die Adaption erfolgen?* Der Schwerpunkt dieser Arbeit liegt auf der Zuweisung von Anwendungen an Rechenknoten im Edge Bereich.
- *Welche Art von Adaption wird benötigt?* Wenn Funktionen oder Rechenknoten ausfallen, müssen die Anwendungen auf verfügbare Rechenknoten verschoben werden, und es muss zur Laufzeit eine Entscheidung über die neue Zuordnung getroffen werden.
- *Wer muss die Adaption durchführen?* Da es sich um ein selbstadaptives System handelt, sollte diese Anpassung automatisch und zur Laufzeit erfolgen.
- *Wie wird die Adaption durchgeführt?* Für die Entscheidung über neue Zuweisungen nach dem Auftreten von Änderungen im System wurde im Rahmen dieser Arbeit eine leichtgewichtige Allokationsmethode entwickelt, die eine entwurfskonforme Neuzuweisung von Funktionen im Betrieb ermöglicht.

Tabelle 2.1: Dimensionen von selbstadaptiven Systemen (SAS) [3].

| **Frage** | **SAS Dimension** |
|---|---|
| Wann? | Zeit (Reaktiv vs Proaktiv) |
| Warum? | Gründe (Kontext, technische Ressourcen, Benutzer) |
| Wo? | Ebene (Anwendungen, Systemsoftware, Kommunikation, technische Ressourcen, Kontext) |
| Was? | Technik (Parameter, Struktur, Kontext) |
| Wer? | N/A (Die Natur eines SAS führt zu einer automatischen Art der Adaption) |
| Wie? | Anpassungssteuerung (Ansatz, Entscheidungskriterien für die Anpassung, Grad der Dezentralisierung) |

#### 2.1.2.1 Analyse der Dimensionen in selbstadaptiven Systemen

**Dimension Zeit:** Der zeitliche Aspekt hängt mit der Frage nach dem Wann zusammen. Die traditionelle Perspektive ist die reaktive Anpassung, nachdem ein Ereignis, das einen Anpassungsbedarf auslöst [10], z. B. die Veränderung von Ressourcen oder der Einbruch der Leistung. In [11] wurden zwei weitere Perspektiven für die Planung einer Anpassung vor einem tatsächlichen Ereignis aufgezeigt: (i) vorausschauend und (ii) proaktiv. Bei der prädiktiven Variante geht es darum, den Anpassungsbedarf zu ermitteln, bevor ein Problem auftritt, während bei der proaktiven Variante die Anpassung zur Verbesserung der Leistung erfolgt, ohne dass ein Problem antizipiert wird. Eine dritte Perspektive kann im Zusammenhang mit Selbstoptimierungseigenschaften einbezogen werden [8]. Die Selbstoptimierung ist eine Teilmenge der Selbstadaption und ist daher implizit in der proaktiven Variante enthalten [12]. Für diese Arbeit ist die Unterscheidung von Adaption vor oder nach dem Anpassungsbedarf ausreichend, um den zeitlichen Aspekt der Anpassung zu beschreiben. Aus Sicht der Allokationsmethode dieser Arbeit könnte der zeitliche Aspekt jedoch als sekundär betrachtet werden, da die Allokationsmethode genauso gut vor oder nach dem Anpassungsbedarf eingesetzt werden kann.

**Dimension Grund:** Im Allgemeinen ist die Adaption eine Reaktion auf eine Veränderung, wobei die Reaktion kostspielig sein kann. Dementsprechend sollten die Art und die Auswirkungen einer Veränderung bestimmt werden. Die Frage nach dem Warum beeinflusst also die Reaktion, denn unterschiedliche Gründe können zu unterschiedlichen Adaptionsmaßnahmen führen. In einem selbstadaptiven System kann der Grund für eine Anpassung eine Veränderung in einem oder mehreren Systemelementen [3] sein: (i) eine Veränderung der technischen Ressourcen, z.B. ein Defekt in einer Hardwarekomponente, ein Softwarefehler oder die Verfügbarkeit einer alternativen Netzwerkverbindung, (ii) eine Veränderung der Umgebung, z.B. die Erweiterung des Aktionsradius oder das Hinzufügen von Geräten. In

dieser Arbeit werden die Gründe für eine Reaktion als On-Demand Ereignisse definiert und eine detaillierte Analyse sowie Auflistung dieser Ereignisse findet sich im Kapitel 4.1.

**Dimension Ebene:** Anpassungen können auf verschiedenen Ebenen des Systems vorgenommen werden. Um die Frage nach dem Wo zu betrachten, können die verschiedenen Ebenen eines selbstadaptiven Systems analysiert werden. Ein selbstadaptives System besteht in der Regel aus den verwalteten Elementen und der Anpassungslogik. Während die Anpassungslogik als Steuerungseinheit der technischen Ressourcen oft stabil bleibt, können die verwalteten Elemente adaptiert werden. Die verwalteten Elemente setzen sich aus verschiedenen Ebenen zusammen. In der Edge-Domäne sind die technischen Ressourcen hauptsächlich die Hardware und die Assets wie Computer, Smartphones, Roboter, Arbeitsmaschinen oder Produktionsanlagen. Die Recheneinheiten werden von einem Software-Stack kontrolliert, der in der Regel aus einer Betriebssystemschicht und Middleware besteht. Die Anwendung wird auf dem Software-Stack ausgeführt. Bei der Anwendung kann es sich um eine einzige Anwendung handeln, die auf einem einzigen Gerät oder verteilt ausgeführt wird. Außerdem können verschiedene Anwendungen gleichzeitig laufen und interagieren, was zu Interferenzen führen kann, d. h. zu unerwünschten Überschneidungen und Abhängigkeiten bei der Nutzung von Ressourcen [13]. Die Kommunikation ist eine weitere Ebene und kann aus zwei Perspektiven betrachtet werden. Die physikalische Netzwerkinfrastruktur, z. B. WLAN, und der logische Kommunikationsmechanismus wie z. B. die ereignisbasierte oder Pub/Sub-Kommunikation. Die Anpassungslogik eines selbstadaptiven Systems muss diese verschiedenen Ebenen und mögliche Anpassungsalternativen einbeziehen. In dieser Arbeit liegt der Fokus auf den Ebenen der technischen Ressourcen, d.h. den verfügbaren Hardware-Recheneinheiten und den Anwendungen im Edge-Netz einschließlich der Kommunikationskosten. Auf diesen Ebenen wird die Policy-Beschreibung genutzt, um die Kriterien für eine entwurfskonforme Allokation von Anwendungen auf Recheneinheiten automatisch zur Laufzeit zu ermitteln.

**Dimension Technik:** Es müssen nicht nur die Adaptationsebenen bestimmt werden, sondern auch die spezifischen Adaptationsmaßnahmen, die auf diesen Ebenen durchgeführt werden. In der Literatur finden sich verschiedene Techniken für die Adaption, die mit der Welche Frage zusammenhängen. Für adaptive Software können zwei Ansätze unterschieden werden: (i) Parameteranpassung und (ii) kompositionelle Anpassung [14]. Die Parameteranpassung ist denkbar einfach, da der Anpassungsmechanismus nur die Parameter kontrollieren und verändern muss. Allerdings können die Parameter voneinander abhängig sein, was eine hohe Komplexität mit sich bringt. Die dynamische Integration neuer Algorithmen zur Laufzeit ist jedoch nicht möglich, wie auch die dynamische Integration von neuen Komponenten. Die kompositionelle Anpassung ermöglicht den dynamischen Austausch von Algorithmen oder Systemkomponenten zur Laufzeit. So ist es möglich, defekte Komponenten zu ersetzen, um Systemausfälle zu verhindern, das System an neue Bedingungen anzupassen oder die Leistung durch Hinzufügen neuer Komponenten zu verbes-

sern. Eine dritte Technik aus dem Bereich Pervasive Computing ist die Kontextanpassung, wodurch der Kontext, in dem Systeme laufen, verändert werden kann. Während die Kontextüberwachung und die Erkennung von Kontextänderungen von vielen selbstadaptiven Ansätzen unterstützt werden, ist die Kontextanpassung oft nicht integriert. Außerdem können Adaptionstechniken in Verhalten, Zusammensetzung bzw. Struktur und Kontext kategorisiert werden [10], [3]. In diesem Zusammenhang ist die Parameteranpassung eine Verhaltensanpassung. Änderungen in der Struktur umfassen den Austausch von Komponenten, die neue Zusammensetzung von Komponenten oder das Entfernen bzw. Hinzufügen von Komponenten. In dieser Arbeit kann die Allokationsmethode als strukturelle Adaption bezeichnet werden, da die Zuweisung von Aufgaben an Edge Nodes den Austausch von Hardware- und Softwarekomponenten impliziert und somit die Struktur des Edge Systems verändert.

**Dimension Steuerung:** Ein selbstadaptives System kann in Anpassungsmechanismen und verwaltete Ressourcen unterteilt werden. Die Anpassungsmechanismen sind für die Verwaltung der Anpassung zuständig. Dazu gehören die Überwachung des verwalteten Systems, die Planung der Adaption und die Ausführung der Adaptionspläne. Die Anpassungsmechanismen sind also mit der Wie-Frage gekoppelt. Zur Umsetzung der Anpassungsmechanismen findet man in der Literatur zwei Gruppen von Ansätzen. Der interne Ansatz mit der Verflechtung der Anpassungslogik mit den Systemressourcen oder der externe Ansatz mit der Trennung in Anpassungslogik und verwaltete Ressourcen [12], [15]. Letzterer erhöht durch Modularisierung die Wartbarkeit. Die Steuereinheit benötigt eine Metrik, um zu entscheiden, wie die Adaption erfolgen soll. Die Metriken können aus Modellen, Regeln und Richtlinien, Zielen oder Nutzenfunktionen [16] abgeleitet werden, um die verschiedenen Adaptionsoptionen zu analysieren und die beste auszuwählen. Ein weiterer Aspekt der Anpassungsmechanismen ist der Grad der Dezentralisierung. Eine zentralisierte Anpassungslogik ist geeignet, um eine Lösung für Systeme mit einer geringen Anzahl von zu verwaltenden Ressourcen zu finden. Für große Systeme mit vielen zu verwaltenden Komponenten kann ein dezentraler Ansatz die Adaptionseffizienz verbessern. In dieser Arbeit wird eine externe Adaption verfolgt, bei der die Adaptionsmechanismen, in diesem Fall die Allokationsmethode, von den zu verwaltenden Ressourcen getrennt sind. Die Metriken für die Entscheidungen basieren auf Policies, die in der Entwurfsphase modellbasiert definiert werden (siehe Kapitel 5.2). Für die Ausführung der Allokationsmethode wird ein gestufter Managementansatz verwendet, der die Vorteile einer verteilten und einer zentralen Allokation für die Adaption zur Laufzeit nach Änderungen im System integriert.

#### 2.1.2.2 Analyse der Engineering Ansätze für selbstanpassende Systeme

Ein selbstadaptives System kann in verwaltete Ressourcen und Anpassungslogik unterteilt werden. Die Anpassungslogik steuert die Adaption und kann in Untereinheiten aufgeteilt

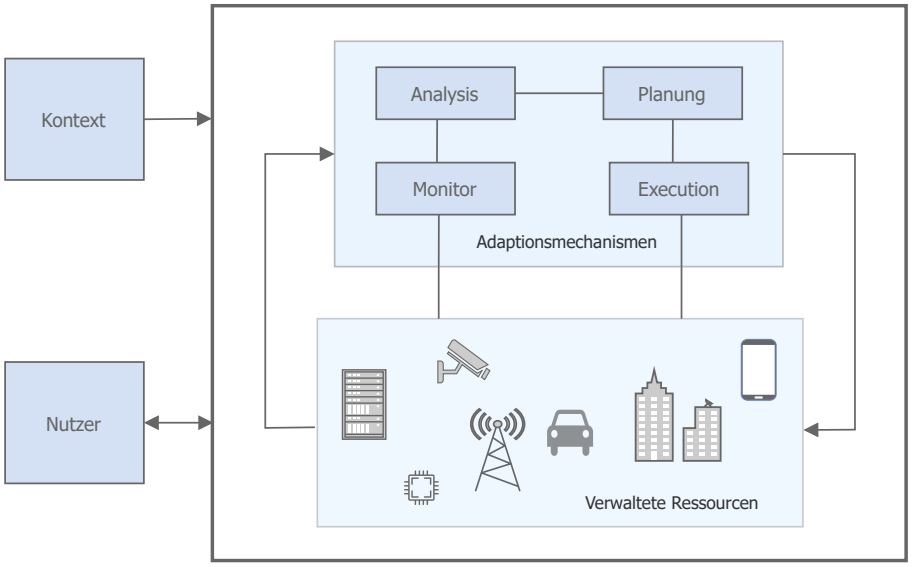



Abbildung 2.3: Aufbau eines selbstadaptiven Systems

werden: Monitoring (M) des Systems und der Umgebung, Analyse (A) der Daten auf Veränderungen, Planung (P) der Adaption und Ausführung (engl. Execution) der Adaption (E). Diese Aktivitäten sind auch als MAPE-Zyklus oder -Funktionalität [8] bekannt und können zum Aufbau eines selbstadaptiven Systems verwendet werden, wie in Abbildung 2.3 dargestellt, wobei die gestrichelte Linie die Systemgrenze darstellt.

Es gibt verschiedene Ansätze für die Implementierung eines selbstadaptiven Systems, ein Auszug ist unten aufgeführt. Eine detaillierte Analyse sowie eine Zuordnung der Adaptionsdimensionen Zeit, Grund, Ebene und Technik zur MAPE-Funktionalität kann in [3] nachgelesen werden.

**Modellgetriebene Ansätze:** Model-Driven Engineering (MDE) konzentriert sich auf die Verwendung von Modellen als Entitäten zur Beschreibung von Software und ihrer Umgebung [17]. Sich ändernde Bedingungen zur Laufzeit führen zu einer Verlagerung der Verwendung von Modellen nicht nur während des Entwurfs, sondern auch zur Laufzeit [18]. Modelle sind ein unterstützendes Instrument für Monitoring und Algorithmen, die durch zusätzliche Techniken für die Veränderung der verwalteten Ressourcen ergänzt werden müssen, wie z.B. serviceorientierte Ansätze (z.B. [19], [20])

**Serviceorientierte Ansätze:** Services bzw. Dienste sind kleine, gekapselte und autonome Softwareeinheiten, die eine bestimmte Aufgabe erfüllen. Entscheidend für den Einsatz von Diensten ist eine serviceorientierte Architektur (SOA), die es ermöglicht, Dienste zu finden, zu nutzen und zu verbinden. So wird der Service-Ansatz auf selbstadaptive Systeme übertragen, indem die Funktionalität der verwalteten Ressourcen als Services modelliert und die SOA für die Kommunikation genutzt wird. Die Anpassungslogik entscheidet, welche Dienste ausgeführt werden sollen. Serviceorientierte Ansätze konzentrieren sich also

auf die strukturelle Adaption durch den Austausch von Diensten oder die Änderung der Zusammensetzung von Diensten. Die Ebene der Anpassung ist in der Regel die Anwendungsebene, wobei in dieser Arbeit nicht nur die Software, sondern auch die Hardwarekomponenten im Edge System für die Adaption berücksichtigt werden.

## 2.2 Modellgetriebene Entwicklung

Modellgetriebene Entwicklung (engl. Model-Drive-Engineering, MDE) [17], [21], [22] ist ein zentrale Entwicklungsparadigma in dieser Arbeit und benutzt Modelle als Artefakt [23], das zur Entwicklung softwareintensiver Systeme wie Edge Systemen verwendet wird. Das wesentliche Merkmal eines Modells ist, dass es ein entsprechendes Original beschreibt. Daher hängt das Konzept der Modelle untrennbar mit einer Modellierungssprache, in der diese Beschreibung gegeben wird. Ein weiteres Merkmal von Modellen ist, dass sie von ihren entsprechenden Originalen abstrahieren. Ein Modell beschreibt also nicht alle Aspekte des Originalsystems in allen Einzelheiten, sondern reduziert die Beschreibung auf eine bestimmte Ebene. Dementsprechend kann Modell folgendermaßen definiert werden: Ein Modell ist eine Beschreibung eines (Teil-) Systems, die in einer wohldefinierten Sprache beschrieben ist.

Im Folgenden werden Entwicklungsprozesse, -werkzeuge und -frameworks, die für die Entwicklung von Edge Systemen von Bedeutung sind, aufgeführt und im Zusammenhang mit den Konzepten dieser Arbeit erläutert.

### 2.2.1 Entwicklungsprozesse

#### 2.2.1.1 Prozessmodelle

**System- und Softwareengineering – Architekturbeschreibung (ISO/IEC/IEEE 42010)**

In der internationalen Norm ISO/IEC/IEEE 42010 wird ein Vorschlag zur Vereinheitlichung der Architekturbeschreibung bei der Systementwicklung gemacht. Es geht nicht darum eine bestimmte Architektur zu beschreiben, sondern die Anforderungen an die bestimmten Anliegen (engl. Concerns) in Bezug auf die Stakeholder des Systems zu stellen. Ein wichtiger Aspekt sind die Architektursichten und -sichtweisen (engl. architecture views and viewpoints) sowie die Architekturmodelle (engl. architecture models). Die konzeptionelle Struktur der Architekturbeschreibung ist in Abbildung 2.4 nach [24] dargestellt. Diese Grundgedanken flossen sowohl in die Erstellung des Modellierungswerkzeugs als auch in die Entwicklung des Policy Beschreibungskonzepts und die Verknüpfung mit dem Systemmodell ein.

**V-Modell (VDI 2206)**

Das V-Modell ist in der Richtlinie VDI/VDE 2206 für die Entwicklung von mechatronischen Systemen formalisiert. Dieser Prozess beginnt mit der Definition von Änderungen, gefolgt vom Systementwurf. Nach dem Systementwurf werden die domänenspezifischen

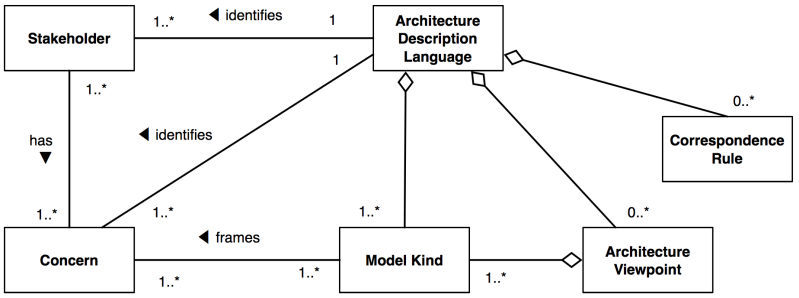



Abbildung 2.4: Überblick der Architekturbeschreibung nach ISO/IEC/IEEE 42010 [24]

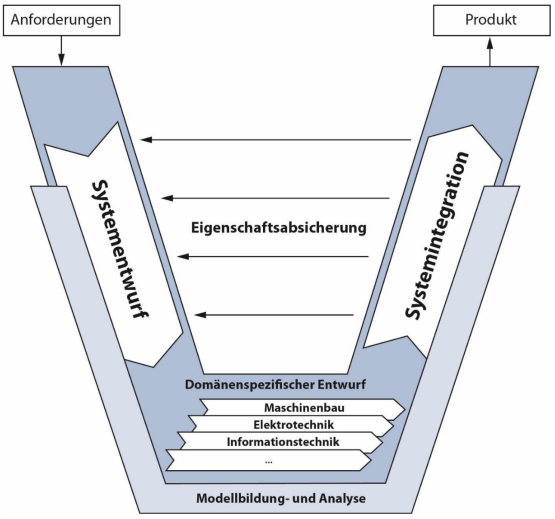



Abbildung 2.5: Übersicht des V-Modells nach VDI/VDE 2206 [27], [25]

Aktivitäten in verschiedene Disziplinen aufgeteilt: Maschinenbau, Elektrotechnik und Informationstechnik. Der letzte Schritt ist die Systemintegration, bei der die Lösungen aller Domänen zu einem System kombiniert werden. In diesem Prozess werden die zuvor definierten Anforderungen mit zunehmender Systemreife verifiziert. Es ist ein iteratives Verfahren von der Verifikation bis zum fertigen Produkt. Der gesamte Prozess wird durch Modellbildung und -analyse unterstützt. Eine Übersicht ist in Abbildung 2.5 zu finden [25], [26], [27].

### Kontinuierliche Entwicklungsprozesse (DevOps)

Die kurzen Zyklen bei der System- und Softwareentwicklung, die für moderne Edge Systeme charakteristisch sind, erfordern kontinuierliche Entwicklungsansätze, was zur Verbreitung von DevOps-Ansätzen und agilen Methoden geführt hat. Diese zielen darauf ab, die Zeit zu verkürzen, die für die Entwicklung, Bereitstellung, Implementierung und Wartung funktionierender, stabiler Systeme benötigt wird. DevOps stammt aus dem Software Engineering Umfeld und begann mit der Idee, Softwareentwicklung (Dev) und Softwarebetrieb (Ops) zu vereinen. Das Hauptmerkmal von DevOps ist die Automatisierung und Überwachung aller Schritte der Softwareentwicklung, von der Integration, den Tests

und der Freigabe bis hin zur Bereitstellung und dem Infrastrukturmanagement. Während die DevOps-Prinzipien in erster Linie in der Softwareindustrie angewandt werden, können dieselben Prinzipien auf der Modellebene für die Entwicklung von Edge Systemen angewendet werden. Hierbei werden die verschiedenen domänenspezifischen Entwicklungsmodelle nahtlos in den Betrieb integriert, entweder über Modelle zur Laufzeit (z. B. modellbasierte MAPE-K-Schleife oder digitale Zwillinge) oder über eine Kombination von Software- und Hardwarekomponenten innerhalb einer bestimmten Betriebsumgebung [28]. Ursprünglich für die Entwurfsphasen in der Softwareentwicklung eingeführt, decken Model-Driven-Engineering Ansätze (MDE) mittlerweile den gesamten Lebenszyklus ab. So können Modelle intensiv genutzt werden, um die Entwicklung und Analyse komplexer Systeme zu automatisieren und um dynamische Rekonfigurationen in selbstadaptive Systeme zu unterstützen.

Die Beschreibung eines kontinuierlichen Entwicklungsprozesses mit Adaption auf Basis von modellgetriebener Entwicklung und DevOps wird im Kapitel 4.2 erläutert.

#### 2.2.1.2 Entwicklungsmethoden

Eine Entwicklungsmethode umfasst in der Regel eine strukturierte und konsistente Vorgehensweise für die Systementwicklung. In diesem Kapitel wird ein Überblick über nichtkommerzielle Entwicklungsmethoden im Kontext von vernetzten eingebetteten Systemen und Industrie 4.0 gegeben, die für die Entwicklung von Edge Systemen relevant sind.

**Architecture Analysis and Design Integrated Approach (ARCADIA)**

Die ARCADIA-Methode ist eine modellgetriebene Engineering Methode für System- und Softwarearchitekturen, die vorwiegend von Thales entwickelt und im Open Source Eclipse Projekt und Tool Capella implementiert wurde. Arcadia trennt die Entwicklung in zwei Phasen, wie in Abbildung 2.6 zu sehen: das Verstehen des Bedarfs und den Entwurf der Lösungsarchitektur. Zu diesem Zweck existiert eine domänenspezifische Beschreibungssprache ARCADIA DSL, die von UML/SysML inspiriert ist. In der ersten Phase wird eine operationale Analyse durchgeführt, d.h. es wird untersucht, was die Benutzer des Systems erreichen wollen. Dann werden die funktionalen und nicht-funktionalen Ziele identifiziert. Also das, was das System für den Benutzer leisten soll. In der zweiten Phase wird zunächst die logische Architektur entworfen, um zu definieren, wie das System die ermittelten Bedürfnisse realisieren soll. Im letzten Schritt wird die physikalische Architektur erstellt, die festlegt, wie das System aufgebaut sein soll, beispielsweise aus der Sicht der Hardware und Software. Insbesondere die Trennung von logischer und physikalischer Architektur wurde bei der Entwicklung des Modellierungswerkzeugs dieser Arbeit berücksichtigt.

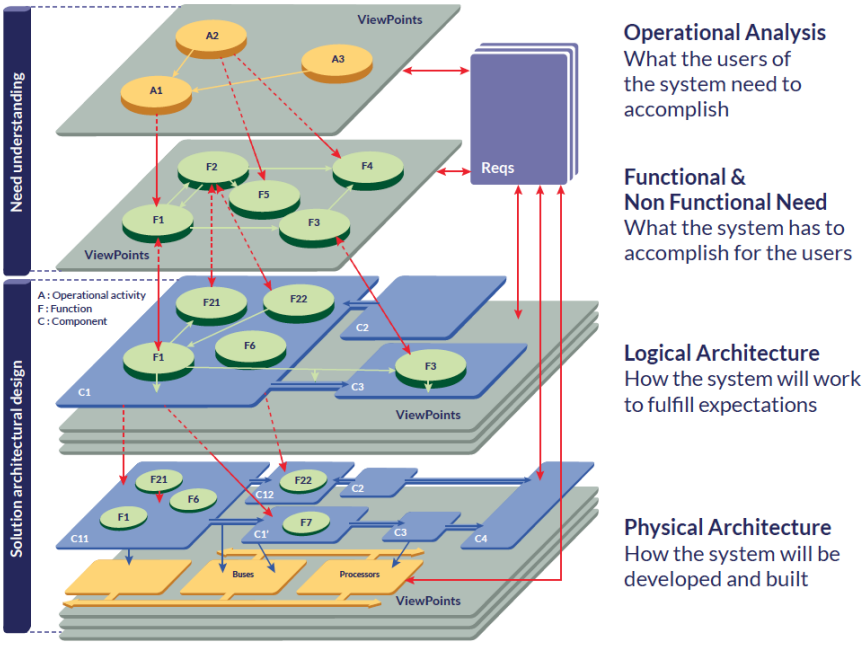



Abbildung 2.6: Überblick über die ARCADIA-Methode

**Conceptual design Specification technique for the Engineering of complex Systems (CONSENS)**

CONSENS ist eine modellbasierte Spezifikationstechnik für das Systems Engineering, die am Heinz Nikdorf Institut in Paderborn entwickelt wurde. Consens stellt Systementwicklern eine DSL für den Entwurf intelligenter Systeme zur Verfügung, die entsprechend ihres situativen Kontextes ihren eigenen Betrieb autonom steuern und optimieren. Daher wird die Software dieser Systeme als selbstadaptiv bezeichnet [29], [30], [31]. Das Ziel von CONSENS ist es, eine umfassende und interdisziplinäre Modellierung mechatronischer Systeme zu ermöglichen. In Abbildung 2.7 ist eine Übersicht über das Consens-Verfahren dargestellt [32], [33], [34]. Die Methode gliedert sich in sieben Teilmodelle. Im Umfeld werden alle externen Einflüsse, die auf das System wirken, berücksichtigt. In den Anwendungsszenarien werden die relevanten Aktivitäten und Interaktionen mit dem System definiert, sowie die relevanten Elemente, die aus dem System resultieren. Anschließend werden die Anforderungen an den Systementwurf formell erfasst. Für die Funktionen wird eine hierarchische Dekomposition durchgeführt. Das Wirkstrukturmodell stellt den grundsätzlichen Aufbau und die Funktionsweise des Systems dar, das in Komponenten unterteilt wird. Das Verhalten umfasst die Modellierung von Aktivitäten, Zuständen und der Interaktion mit der Wirkstruktur. Schließlich wird beim Gestaltmodell die physische Struktur in der Regel als 3D-CAD-Modell erstellt.

Diese Entwicklungsmethode konzentriert sich weitgehend auf die Eigenschaften eines mechatronischen Systems, einschließlich seiner mechanischen Konstruktion. Da der Schwer-

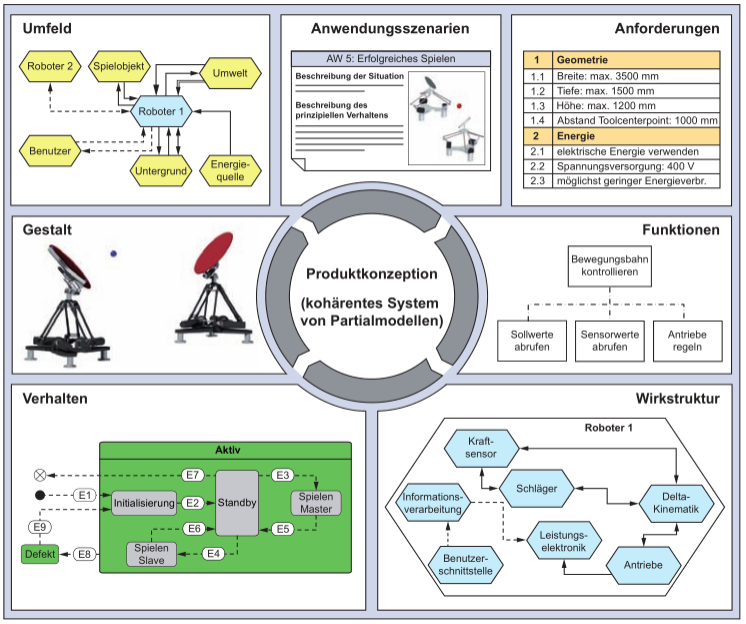



Abbildung 2.7: Übersicht der Spezifikationstechnik CONSENS [27]

punkt dieser Arbeit auf der Edge Infrastruktur liegt, sind nicht alle Teilmodelle relevant. Die Grundgedanken der kontinuierlichen und adaptiven Entwicklung sowie die Teilmodelle der Anwendungsszenarien und Funktionen sind jedoch in das Konzept der Modellierungswerkzeuge und der Entwicklungsmethode dieser Arbeit eingeflossen.

### Functional Architectures for Systemes (FAS)

Die FAS Methode ist sprachunabhängig, wobei sie meist mit SysML verwendet wird. Mit Hilfe der FAS Methode werden funktionale Architekturen aus Use Cases abgeleitet, unter Verwendung sogenannter Use Case Activities [35]. Eine detaillierte Beschreibung kann aus [36] entnommen werden.

Die Grundideen der FAS Methode sind teilweise in die Modellierung der logischen Architektur dieser Arbeit eingeflossen.

### Software Platform Embedded Systems 2020 (SPES2020)

Die SPES-Methode ermöglicht die domänenübergreifende modellbasierte Entwicklung eingebetteter Systeme. Die SPES-Methode führt die Idee ein, die Domänen eines Systems in zwei Dimensionen zu betrachten. Zum einen die Sichtweisen (engl. viewpoints) und zum anderen die Abstraktionsebenen (engl. abstraction layers), wie in Abbildung 2.8 dargestellt. Die Sichtdimension geht von den Anforderungen über die funktionale Sicht, die logische Sicht bis hin zur technischen Sicht und das kann nach der SPES-Methode auf verschiedenen Abstraktionsebenen des Systems oder von Teilsystemen betrachtet werden.

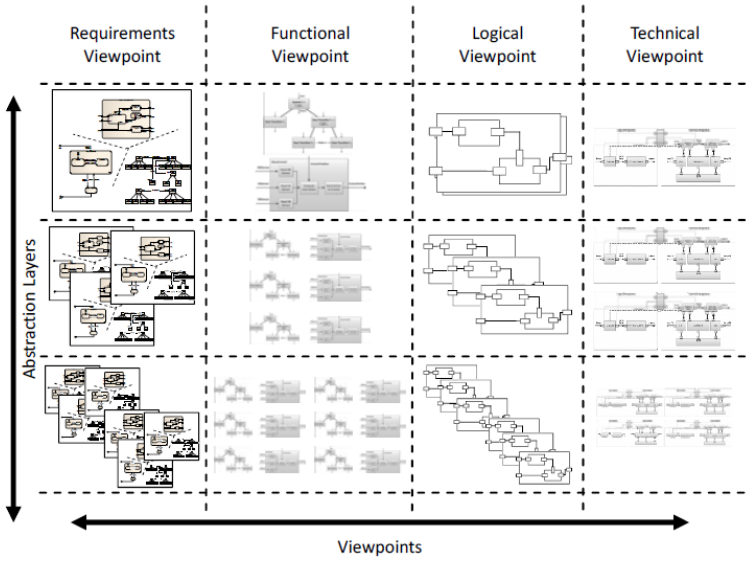



Abbildung 2.8: SPES Modellierungsframework mit Dimensionen [37]

In dieser Arbeit werden ähnliche Konzepte zur Trennung von logischen Funktionen und technischen Sichten in verschiedenen Ebenen, sowie zur Verfeinerung der Komponenten in Subsystemen angewendet.

### Systems Modeling Toolbox (SYSMOD)

Die SYSMOD Methode wird zur Modellierung von Anforderungen, funktionaler und physischer Architektur komplexer Systeme eingesetzt. SYSMOD kann mit Hilfe eines SysML-Profils implementiert werden und die Methode kann je nach Anwendungsbereich angepasst werden, da sie als Toolbox aufgebaut ist.

Im Analyseprozess werden zunächst die Stakeholder identifiziert und die Systemziele beschrieben. Daraus wird die Systemskizze (Base Architecture) erstellt, für die der Systemkontext und die Anwendungsfälle identifiziert werden. Auf dieser Basis wird das Systemverhalten (system processes) in Form von Aktivitätsdiagrammen oder Zustandsdiagrammen beschrieben. Im Architekturentwurfsprozess werden die zuvor definierten Modelle einbezogen und darauf aufbauend wird zunächst die logische Architektur modelliert, die die Systemkomponenten auf einer abstrakten Ebene mittels Blockdiagrammen darstellt. Hieraus werden die spezialisierte Produktarchitektur und die Systemzustände abgeleitet. Schließlich wird diese physikalische Architektur anhand der zuvor definierten Szenarien verifiziert.

Für die Konzepte dieser Arbeit bezüglich des Modellierungswerkzeugs und des Entwicklungsprozesses wurden die Ideen des Analyseprozesses und des Architekturentwurfsprozesses berücksichtigt und an die Idee der Policy Beschreibungen entlang des Systemmodells angepasst.

### 2.2.2 Modellierungssprachen und frameworks

#### 2.2.2.1 Modellierungssprachen

Für die Anwendung von modellgetriebenen Ansätzen ist die Beschreibung der Modellierungssprache unerlässlich. Im Gegensatz zu traditionellen Modellierungssprachen bei der Softwareentwicklung, wie der standardisierten Unified Modeling Language (UML) , können sich Sprachen auf eine bestimmte Anwendungsdomäne konzentrieren [17]. Eine Modellierungssprache für eine bestimmte Domäne wird als Domain Specific Language (DSL) bezeichnet [38], [39]. Wenn die Definition einer Modellierungssprache selbst als Modell gegeben ist, nennt man sie ein Metamodell [40], [41].

Ein Metamodell kann als ein Modell definiert werden, das zur Spezifikation einer Sprache verwendet wird. Somit ist ein in einer Modellierungssprache gegebenes Modell eine Instanz des zugehörigen Metamodells. In dieser Arbeit wird das Eclipse Modeling Framework (EMF) als Basisframework für die Metamodellierung verwendet.

Die Modelltransformation ist ein wesentlicher Bestandteil der modellgetriebenen Entwicklung, um automatisch Entwicklungsartefakte aus Modellen zu erzeugen oder zu analysieren [42], [43], [44], [45], [46]. In dieser Arbeit werden sowohl Modell-zu-Modell-Transformationen einbezogen, bei denen die generierten Artefakte wiederum Modelle zur Analyse sind, als auch Modell-zu-Text-Transformationen, bei denen Konfigurationen in Austauschdateien generiert werden.

Wenn keine eigene domänenspezifische Sprache (DSL) für die Modellierung entwickelt wird, wird im Bereich des modellbasierten Systems Engineering häufig die Systems Modeling Language (SysML) verwendet, die auf der Unified Modeling Language (UML) basiert. Aus diesem Grund wird im Folgenden ein kurzer Überblick über die beiden Sprachen gegeben.

**Unified Modeling Language (UML)**

Die UML hat ihren Ursprung im Software Engineering und wurde 2017 von der Object Management Group (OMG) in der Version 2.5.1 veröffentlicht. Abbildung 2.9 zeigt eine Übersicht der UML Diagramme, die sich in Struktur- und Verhaltensdiagramme unterteilen lassen.

**Sytems Modeling Language (SysML)**

SysML ist eine für das Systems Engineering zugeschnittene Modellierungssprache, die ebenfalls von der OMG veröffentlicht wurde. Dafür wurden Teile der UML, die für die Systementwicklung nicht relevant sind, herausgenommen und durch einige weitere Aspekte erweitert, wie zum Beispiel die Anforderungsdiagramme oder die Blockdefinitionsdiagramme. In Abbildung 2.10 ist eine Übersicht der SysML Diagramme zu sehen.

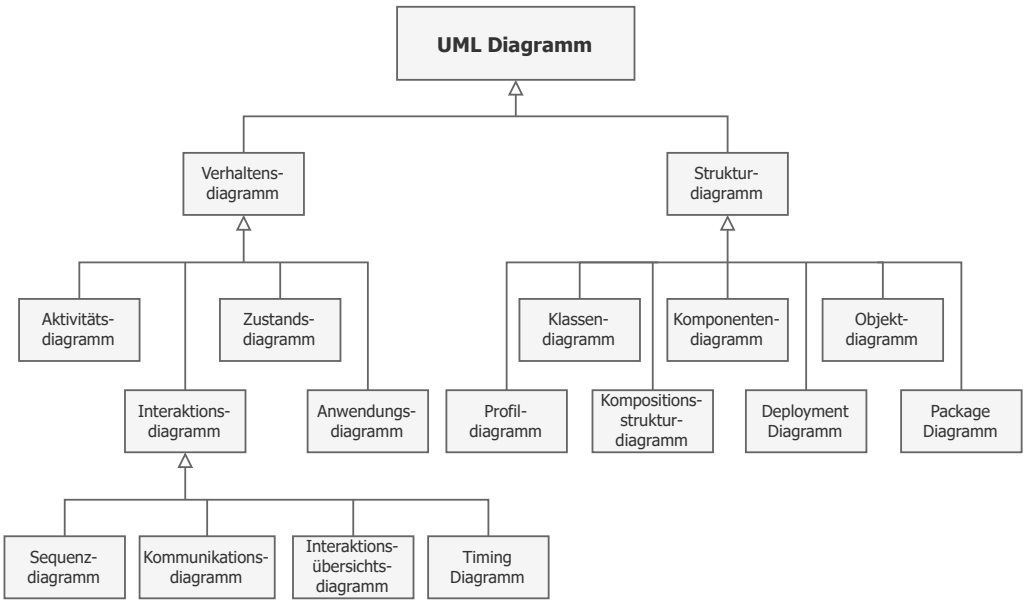



Abbildung 2.9: Übersicht der UML Diagramme

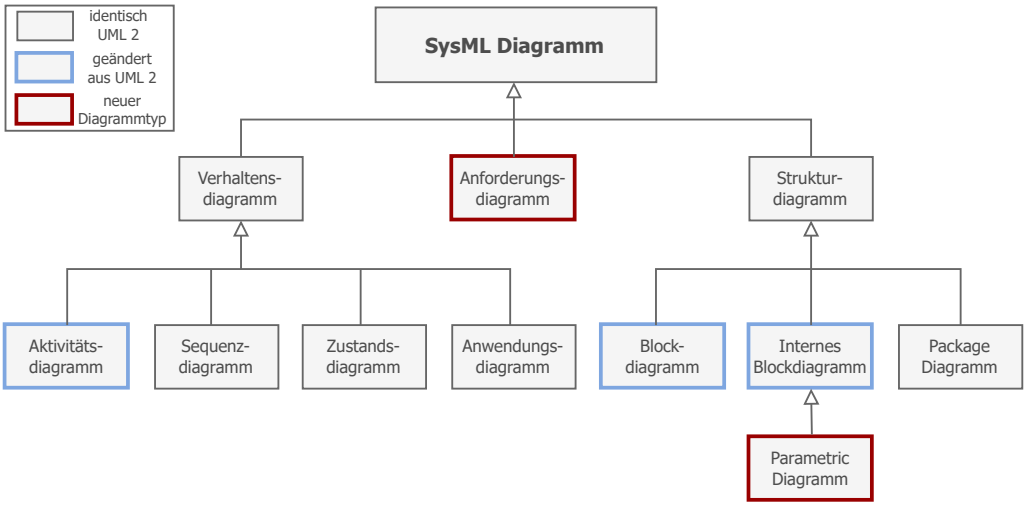



Abbildung 2.10: Übersicht über die SysML Diagramme

Im Rahmen dieser Arbeit wurden für das Modellierungswerkzeug verschiedene domänenspezifische Modellierungssprachen für die Systemmodellierung und insbesondere für das Policy-Beschreibungskonzept entwickelt. Die Metamodelle sowie Diagramme im Entwicklungswerkzeug sind jedoch basierend auf dem Konzept der SysML Sprache entwickelt worden, so dass die Konzepte dieser Arbeit leicht übertragbar sind.

#### 2.2.2.2 Modellierungsframework

Für die Modellierung wurde das Open Source Framework der Eclipse Community verwendet, wie in Kapitel 4.3 erläutert, da in dieser Form die Konzepte und Methoden dieser Arbeit demonstriert und validiert werden können.

**Eclipse Modeling Framework (EMF)**

Das Eclipse Modeling Framework (EMF) [47] ist eines der am weitesten verbreiteten Frameworks für die modellgesteuerte Entwicklung. Es handelt sich um ein von der Eclipse Foundation entwickeltes open source Framework. EMF kann sowohl zur Modellierung von Modellen als auch von Metamodellen verwendet werden. Ein EMF-Modell definiert die Daten eines Systems. Es spezifiziert die Attribute eines Objekts, die Beziehungen zwischen Objekten, mögliche Operationen für jedes Objekt sowie Bedingungen für Objekte und Beziehungen. Zu diesem Zweck gibt es drei verschiedene Arten der Darstellung: 1. Java-Schnittstelle 2. UML-Klassendiagramm 3. XML-Schema. Im EMF wird ein Meta-Metamodell namens Ecore verwendet. Dieses Meta-Metamodell kann verwendet werden, um domänenspezifische Metamodelle in EMF zu entwickeln und zu beschreiben.

**Ecore**

Das Ecore-Modell basiert auf dem EMOF-Standard (Essential Meta Object Facility). Dabei stellt das Element EClass eine Klasse wie in der Programmiersprache Java dar. Eine Klasse enthält verschiedene Attribute und Referenzen auf andere Klassen. Attribute und Referenzen werden in Ecore mit EAttribute und EReference beschrieben. Vererbungsbeziehungen werden durch die Eigenschaft eSuperTypes dargestellt. Ein Attribut hat einen Typ, der in Ecore mit EDataType beschrieben wird. Eine Referenz in Ecore ist standardmäßig eine Assoziation. Aggregationen und Kompositionen können durch die Eigenschaft containment dargestellt werden. Abbildung 2.11 zeigt einen Auszug aus dem Ecore Meta-Metamode

**Ecore Tools**

Ecore Tools ist ein Framework zur grafischen Erstellung, Bearbeitung und Verwaltung von Ecore-Modellen. Ecore Tools ist mit dem grafischen Modellierungsframework Sirius entwickelt worden. Damit wird eine übersichtliche Darstellung eines Modells ermöglicht, ähnlich wie bei einem UML-Klassendiagramm.

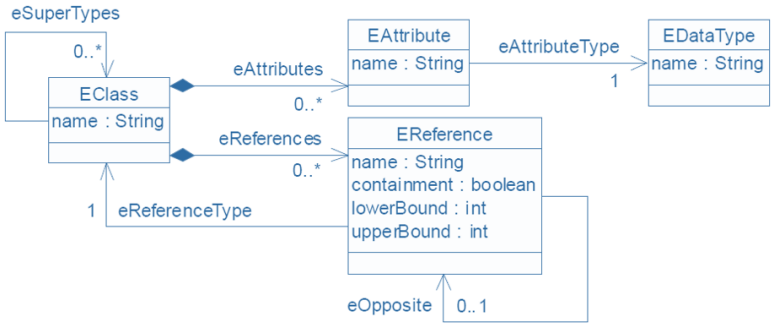



Abbildung 2.11: Auszug aus dem Ecore-Modell [47]

### EMF Sirius

Sirius ist ein Framework, das eine einfache Möglichkeit zur Erstellung eines grafischen Editors für domänenspezifische Modelle bietet. Das Sirius Framework basiert auf EMF. Um einen Editor zu entwickeln, werden Struktur, Aussehen und Verhalten in einem Viewpoint Specification Project (VSP) definiert. Innerhalb dieses Projekts enthält das Viewpoint Specification Model (VSM) die modellbasierte Definition des Editors. Mit Sirius können Modelle als Diagramme, Strukturbäume und Tabellen visualisiert und bearbeitet werden. Die grundlegenden Darstellungen des Editors können vollständig ohne Code realisiert werden. Für komplexes Editorverhalten können Java-Erweiterungen entwickelt und in Sirius verwendet werden.

In der Eclipse Community gibt es weitere Werkzeuge und Frameworks, die mit EMF kompatibel sind. Mit Xtext können textbasierte Modellierungssprachen definiert werden. Mit Acceleo kann eine Model2Text Transformation definiert werden, z.B. zur Code- oder Dateigenerierung.

## 2.3 Ressourcenmanagement in Edge Computing

Die Terminologie in diesem Abschnitt lehnt sich an die Beschreibungen in [48] an, das einen Überblick über die Forschung zum Ressourcenmanagement im Edge Computing gibt. Im Zuge der Entwicklung von Internet of Things (IoT) können nicht nur Computer oder Smartphones an das Netz angeschlossen werden, sondern auch eine Vielzahl von Dingen wie Autos, Sensoren, Drohnen, Roboter oder Arbeitsmaschinen. Daher werden die Objekte, die sich auf der Anwenderseite des Netzwerks befinden und Daten erzeugen, als Endgeräte bezeichnet. Zu den Edge-Geräten gehören die Geräte, die Recheneinheiten enthalten und die Endgeräte mit dem Rest des Netzes verbinden, z. B. Heimrouter, Gateways, Access Points oder Basisstationen, die immer leistungsfähiger werden [38]. In dieser Arbeit liegt der Schwerpunkt auf Geräten mit Recheneinheiten bzw. Ressourcen, auf denen Aufgaben ausgeführt werden können. Verwaltete Ressourcen werden zur Ausführung von Aufgaben auf einer bestimmten Ebene der Architektur verwendet und können so aufgebaut werden, dass sie letztendlich einen Service bereitstellen.

### 2.3.1 Ressourcen und Management in Edge

Um einen Überblick über das Thema Ressourcenmanagement in Edge zu geben, werden vier Hauptaspekte aufgezeigt: Ressourcentyp, Ziel der Ressourcenverwaltung, Ressourcenort und Ressourcennutzung [48], wie in Abbildung 2.12 dargestellt.

**Ressourcentyp**

Für die Verwaltung von Edge Systemen ist es wesentlich zu bestimmen, welche Arten von Ressourcen im Vergleich zu einem zentralisierten System vorhanden sind. Eine naheliegende Rechtfertigung für den Einsatz von Edge Architekturen ist die Verringerung der Reaktionszeit unter Berücksichtigung der Ressourcenarten *Rechenzeit* und *Kommunikation*. Auch die *Speicherung* ist ein wichtiger Aspekt, da eine lokale Speicherung mit angepassten sicheren Speichermechanismen die Sicherheit verbessern kann. Eine andere Art von Ressource ist der Zugriff auf spezifische *Daten*, z. B. von Sensoren, die lokale Vorteile in einer Anwendung bieten. Die Menge und Art der gesammelten Daten hat wiederum Auswirkungen auf die Rechen- und Kommunikationsressourcen und damit auf die Entscheidung, wo und wie viele andere Ressourcen verwendet werden sollen. Eine weitere Kategorie ist die *Energie*, die von der Menge an Berechnungen, Kommunikation, Speicherung und Datenerfassung beeinflusst wird. Schließlich können die Ressourcentypen auch auf *generische Art* und Weise betrachtet werden.

**Ziele des Ressourcenmanagements**

Die Ziele des Ressourcenmanagements sind orthogonal zu den im vorherigen Abschnitt vorgestellten Ressourcentypen.

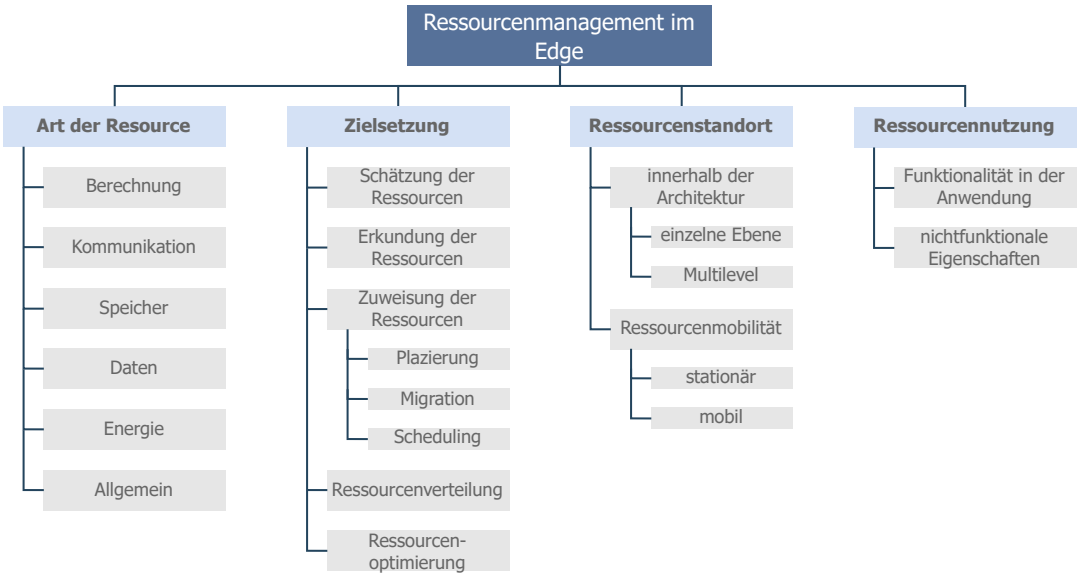



Abbildung 2.12: Aspekte des Ressourcenmanagements in Edge Systemen [48]

Die *Ressourcenabschätzung (engl. resource estimation)* ist die Fähigkeit, abzuschätzen, wie viele Ressourcen zur Realisierung einer Aufgabe erforderlich sein werden. Auf diese Weise können Schwankungen in der Ressourcennachfrage antizipiert und die Dienstgüte (Quality of Service, QoS) sichergestellt werden.

Die *Ressourcenerkennung (engl. resource discovery)* bezieht sich im Gegensatz zum Schätzungsproblem auf die Nutzungsseite. Dabei muss ein Managementsystem wissen, welche Ressourcen verfügbar sind, wo sie sich befinden und wie lange sie verfügbar sind.

Die *Ressourcenzuweisung (engl. resource allocation)* kann aus zwei Sichtweisen betrachtet werden: Wo soll zugeteilt werden und wann soll wie viel zugeteilt werden. Die Ansätze lassen sich in drei Perspektiven unterteilen [48]: Placement, Migration und Scheduling. Beim Placement geht es darum, wo die Aufgabe ausgeführt werden soll und welche Ressourcen für eine optimale Ausführung zugewiesen werden sollen. Die Definition der bestmöglichen Ausführung variiert je nach dem betrachteten System. Die Migration befasst sich zusätzlich mit der Frage, wie virtualisierte Dienste während der Ausführung verschoben werden. Beim Scheduling steht die Frage im Vordergrund, wann und wie viele Ressourcen im Edge Netzwerk zugewiesen werden sollen, wobei es sich in der Regel um einen feingranularen Zuweisungsmechanismus handelt.

Die *gemeinsame Nutzung von Ressourcen (engl. resource sharing)* wird meist bei heterogenen und ressourcenbeschränkten Edge Geräten eingesetzt. Damit lassen sich beispielsweise folgende Probleme lösen: Die benötigte Ressource ist auf dem Gerät, auf dem die Aufgabe gestartet wird, nicht oder nicht ausreichend verfügbar, oder es können Ressourcen von anderen Geräten verwendet werden, um die Aufgabe effizienter zu erledigen. Diese Probleme sind jedoch implizit in das Ressourcenplacement integriert.

Die *Ressourcenoptimierung (engl. resource optimization)* ist in der Regel mit einem oder mehreren der zuvor beschriebenen Ziele des Ressourcenmanagements verbunden. Welcher Aspekt zu optimieren ist und welche Einschränkungen damit verbunden sind, variiert je nach Edge-System. Die drei beliebtesten sind QoS (oft als Latenzzeit verstanden), Energie und Betriebskosten.

**Ressourcenort**

Die beim Edge Computing verwendeten Ressourcen können verschiedenen Ebenen angehören. Neben der primären Nutzung von Ressourcen auf der Edge Ebene können bei Bedarf auch Ressourcen auf der Cloud Ebene genutzt werden. End- und Edge-Geräte können auch nicht ortsgebunden sein. Der Standort innerhalb der Architektur kann danach unterschieden werden, auf wie vielen Ebenen die Ressourcen verwaltet werden können. Bei der Ressourcenmobilität geht es darum, ob die Ressourcen im Edge Netzwerk stationär oder mobil sind.

**Ressourcennutzung**

Der letzte Aspekt ist die Analyse des Zwecks, für den die Ressourcen verwendet und verwaltet werden. Dabei kann zwischen funktionalen und nicht-funktionalen Eigenschaften unterschieden werden. Die *funktionalen Eigenschaften* beziehen sich auf den Zugriff auf einen bestimmten Dienst, also auf die Erfüllung von Funktionen in einer Anwendung. Die *nicht-funktionalen Eigenschaften* werden meist zusätzlich bei der Gestaltung der Edge Architektur berücksichtigt, um den zu erbringenden Dienst auf eine bestimmte Weise zu realisieren. Dabei beziehen sich die nicht-funktionalen Eigenschaften auf die Zielkategorien für das Ressourcenmanagement. Exemplarisch werden Metriken der Zielkategorien verwendet, die repräsentativ für die Messung der nicht-funktionalen Eigenschaften verwendet werden [48]: (i) die Antwortzeit als Maß für zeitgerechtes Verhalten (ii) der Energieverbrauch als Maß für die Energieeffizienz (iii) die Zugriffsrate oder die zugehörige Blockiergefahr, als Maß für die Verfügbarkeit (iv) die CPU- oder Netzauslastung als Maß für die effiziente Nutzung von Rechen-/Kommunikationsressourcen (v) die finanziellen Kosten als Maß für die Kosteneffizienz. Die Liste der Metriken ist keinesfalls abschließend, sondern zeigt vielmehr die häufig verwendeten Effizienzmetriken.

Die konzipierte Allokationsmethode berücksichtigt unterschiedliche Ressourcentypen, da prinzipiell jede Art von Ressource im Policy Beschreibungskonzept definiert werden kann. In den für die Evaluation betrachteten Anwendungsfällen werden explizit die Ressourcen Rechenleistung, Kommunikation, Speicher und Energie modelliert. Die Arbeit konzentriert sich beim Ziel der Ressourcenverwaltung auf die Ressourcenallokation und dabei primär auf den Bereich des Placements, da die Allokationsmethode Aufgaben auf Edge-Geräte platziert bzw. zuweist. Dennoch spielen auch andere Ziele bei der Integration und Evaluation eine Rolle. Die betrachteten Ressourcen befinden sich hauptsächlich auf der

Edge-Ebene und sind teilweise mobil. Als nicht-funktionaler Aspekt ist die Verfügbarkeit in dieser Arbeit besonders wichtig, da die automatische Reallokation von Ressourcen die Verfügbarkeit erhöhen und Ausfallzeiten reduzieren soll.

### 2.3.2 Allokationsproblem

Die Grundversion des Allokationsproblems ist fester Bestandteil der Einführung in Operations Research oder Produktions- und Betriebsmanagement. Das Problem wird in der Regel als eine eins-zu-eins-Zuweisung zwischen n Aufgaben und n Agenten definiert, wobei das Ziel die Minimierung der Gesamtkosten der Zuweisungen ist. Übliche Beispiele sind die Zuordnung von Aufgaben zu Maschinen, von Aufgaben zu Arbeitern oder von Arbeitern zu Maschinen. Das mathematische Modell für das klassische Allokationsproblem kann wie in (2.1)dargestellt werden. Dabei ist $x_{ij} = 1$, wenn der Arbeiter $i$ der Aufgabe $j$ zugewiesen ist, 0, wenn nicht. $c_{ij}$ sind die Kosten der Zuweisung des Arbeiters $i$ zur Aufgabe $j$. Die erste Bedingung stellt sicher, dass jede Aufgabe nur einem Bearbeiter zugewiesen wird. Die zweite Bedingung stellt sicher, dass jeder Bearbeiter einer Aufgabe zugewiesen wird. Zudem gibt es mehrere Varianten des klassischen Allokationsproblems, eine detaillierte Aufzählung ist in [49] enthalten. Die Analyse des Allokationsproblems im Kontext von Edge Systemen für die Allokationsmethode dieser Arbeit wird im Kapitel 5.1 erläutert.

$$
\begin{aligned}
\text{Minimieren} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{Unter Beachtung von:} \quad & \sum_{i=1}^{m} x_{ij} = 1 \quad j = 1,\ldots,n, \\
& \sum_{j=1}^{n} a_{ij}x_{ij} \leq b_i \quad i = 1,\ldots,m, \\
& x_{ij} = 0 \text{ oder } 1
\end{aligned}
\tag{2.1}
$$

Das Allokationsproblem kann für die Zuweisung von einer oder mehreren Aufgaben pro Agent gelöst werden. Es gibt Varianten, bei denen die Fähigkeiten der Agenten einbezogen oder die Anforderungen kategorisiert werden. Darüber hinaus gibt es multikriterielle Allokationsprobleme, bei denen mehr als eine Eigenschaft optimiert werden soll. Im Allgemeinen gibt es zwei Methoden, um mehrere Kriterien im Entscheidungsmodell zu berücksichtigen: die Kombination zu einem Kriterium oder die separate und sequentielle Optimierung der Kriterien. Darüber hinaus gibt es Allokationsprobleme mit mehreren Nebenbedingungen, wie die Berücksichtigung mehrerer Merkmale als Einschränkungen bei der Problemfindung. Eine weitere Variante sind die mehrdimensionalen Allokationsprobleme. Hierbei wird die Allokation im Vergleich zum klassischen Problem aus mehr als zwei

Mengen bestimmt. Ein beliebtes Beispiel ist die Zeit in Perioden als dritte Dimension, wie bei der Integration von Scheduling in das Entscheidungsproblem.

Für verschiedene Anwendungen und Systeme, bei denen das Allokationsproblem relevant ist, sind Merkmale wie geringerer Stromverbrauch, niedrigere Latenzzeiten und geringer Verbrauch von Rechen- und Speicherressourcen wichtig. Bei IoT-Systemen beispielsweise ermöglicht ein niedriger Energieverbrauch, dass das System auf Edge Geräten und mobilen Plattformen implementiert werden kann. Geringe Latenzzeiten sind für Systeme relevant, bei denen sich die Komponenten wie Fahrzeuge oder Maschinen bewegen. Darüber hinaus verfügen Edge Systeme nur über begrenzte Rechen- und Speicherressourcen, so dass ein möglichst effizienter Umgang mit diesen Ressourcen gewährleistet sein muss.

**Analyse der mathematischen Modelle**

Die Ressourcenallokation zwischen mehreren an einem Edge Computing Szenario beteiligten Einheiten ist eine komplexe Aufgabe mit vielen Parametern. Es gibt diverse etablierte mathematische Modelle und Berechnungsmethoden, um das Problem zu modellieren und zu lösen. Im Folgenden wird daher eine Liste einiger mathematischer und Berechnungsmodelle, Algorithmen und Techniken vorgestellt. Eine detaillierte Diskussion dieser Modelle kann in [50] nachgelesen werden.

Der Ansatz der *Spieltheorie (engl. game theory)* besteht darin, die Interessen vieler Spieler oder Einheiten unter bestimmten Bedingungen zu maximieren. Ein Spielmodell besteht in der Regel aus drei Elementen: der Auszahlungsfunktion, der Strategie und den Spielern. Die Spieler können einzeln oder als Team über ihre eigene Strategie entscheiden. Spielmodelle werden in zwei Kategorien unterteilt: Klassische Spiele in kooperativer oder nicht-kooperativer Form und evolutionäre Spiele in statischer oder dynamischer Form [51], [52].

Ein *bipartiter Graph oder Bi-Graph* in der mathematischen Graphentheorie ist eine Menge von Graphknoten, die in zwei disjunkte Mengen aufgeteilt sind. Das bedeutet, dass sie kein gemeinsames Element haben und keine zwei Knoten innerhalb desselben Graphen benachbart sind. Beim Edge Computing eignen sich bipartite Graphen zum Beispiel für die Lösung von Energieoptimierungsproblemen, obwohl die Skalierbarkeit problematisch sein könnte.

*Markov-Modelle* sind stochastische Methoden, die zur Modellierung dynamischer Systeme verwendet werden. Die vier gängigen Markov-Modelle sind Markov-Kette, Markov-Entscheidungsprozess, Semi-Markov-Prozess und Hidden-Markov-Modell [53]. Dabei wird angenommen, dass zukünftige Zustände nur vom aktuellen Zustand abhängen, das heißt nicht von den Ereignissen, die davor eingetreten sind. Zum Beispiel kann ein Markov-Entscheidungsprozess verwendet werden, um das Problem der Ressourcenzuweisung mit einer Lohnfunktion zu formulieren [54].

*Heuristische Algorithmen* sind so konzipiert, dass sie Probleme schnell und effektiv lösen können, ohne dabei große Kompromisse bei der Genauigkeit einzugehen. Ein erster heuristischer Algorithmus kann eine kurzfristige Lösung für ein Problem liefern und anschließend kann er mit den Reinforcement Learning Algorithmen für langfristige Lösungen kombiniert werden, wodurch die Ressourcenzuweisung effizienter wird. Beim Reinforcement Learning handelt es sich um einen Algorithmus, bei dem intelligente Agenten in einer Umgebung geeignete Aktionen ausführen, um die Vergütung in einer iterativen Weise zu maximieren.

Die *Mehrzieloptimierung (engl. multiobjective optimization)* spielt in der Forschung und Entwicklung eine entscheidende Rolle, insbesondere bei Anwendungen, bei denen mehrere Zielfunktionen involviert sind. Es werden optimale Lösungen vorgeschlagen, wobei Kompromisse zwischen den verschiedenen Zielen eingegangen werden müssen. Es gibt keine eindeutigen Lösungen für die Mehrzieloptimierung oder das Pareto-Optimierungsproblem. Diese Lösungen werden als nicht-dominiert oder Pareto-optimal bezeichnet und werden nach einigen Iterationen gefunden. Eine Möglichkeit, diese Probleme zu lösen, sind genetische Algorithmen wie der Non-Dominated Sorting Genetic Algorithm III (NSGA-III). Die Analyse von *multikriteriellen Entscheidungen (engl. Mutiple Criteria Decision Analysis)* befasst sich hingegen mit der Suche nach verschiedenen optimalen Lösungen und nicht nach eindeutigen Lösungen von Problemen mit mehreren Kriterien.

Zusammenfassend kann festgestellt werden, dass es mehrere Ansätze zur mathematischen Modellierung und Lösung des Ressourcenallokationsproblems gibt. Die verschiedenen Ansätze unterscheiden sich stark in Bezug auf Performance, Effektivität und Effizienz. Eine geeignete Methode muss je nach Kontext und Ziel des Problems analysiert werden. Diese Arbeit konzentriert sich auf die automatische Lösung des Allokationsproblems während des Betriebs, um die Selbstadaption der Edge Infrastruktur nach Änderungen im System zu ermöglichen. Bei der Auswahl eines geeigneten Modells und Solvers stehen daher Aspekte wie Effektivität und schnelle Lösung im Vordergrund. Eine ausführliche Analyse und Tradeoff für diesen Einsatzzweck findet sich im Kapitel 6.1.

## 2.4 Infrastuktur und Technologien für Edge Computing Systemen

Im Unterschied zur Nutzung zentralisierter Cloud Rechenzentren ist die Verwendung dezentraler Ressourcen im Randbereich eines Netzwerks für die Verarbeitung von Daten näher an den Maschinen und Geräten bspw. in der Produktion ein aufkommendes Paradigma, das als Edge- oder Fog-Computing bezeichnet wird [55], [56]. Edge Ressourcen sind im Vergleich zur Cloud in der Regel ressourcenbeschränkt, heterogen und dynamisch, wodurch die Ressourcenverwaltung als große Herausforderung gilt. Ein Überblick über diese Herausforderungen und weiterführende Arbeiten findet sich in [57]. Die Edge Computing Infrastruktur umfasst Hardware- und Softwarekomponenten zur Verwaltung der Rechen-, Netzwerk- und Speicherressourcen [58].

### 2.4.1 Edge Hardware

Im Edge Computing werden energieeffiziente mobile Geräte, Gateways, Router, Autos und sogar Drohnen als Rechengeräte eingesetzt. Diese Geräte verfügen heutzutage über zunehmende Rechen- und Konnektivitätsfähigkeiten. Die Kombination dieser Rechengeräte ermöglicht eine Rechenumgebung für die Verarbeitung von Internet of Things (IoT) Anwendungen und cyber-physischen Systemen (CPS). Dabei kann die für Edge Computing verwendete Hardware in Rechen- und Netzwerkgeräte unterteilt werden.

Rechengeräte: Einplatinencomputer und so genannte lokale Edge Server werden zur Verarbeitung von Edge Computing Anwendungen eingesetzt. Einplatinencomputer wie der Raspberry Pi oder die i.MX-Serie werden häufig als Edge Nodes eingesetzt [59]. Solche Einplatinencomputer integrieren CPU, Speicher, Netzwerk- und Speichergeräte und andere Komponenten. Sie sind entweder direkt oder über Gateways mit Endgeräten wie Kameras, Maschinen, Drohnen usw. verbunden. Lokale Edge Server verfügen über mehr Rechenkapazität, indem sie z. B. GPUs oder FPGAs integrieren, und können vor Ort in die Werkshalle von Produktionsstätten integriert werden.

Netzwerkgeräte: Darunter fallen Gateways, Router, drahtlose Zugangsstationen usw., die sich am Rande des Netzes befinden und hauptsächlich den Netzverkehr verarbeiten. Netzwerk-Gateways und -Router integrieren zunehmend Rechenkapazitäten, um Rechenleistung zu bieten und gleichzeitig den Kommunikationspfad zwischen Endgeräten, dem Edge-Netzwerk und dem Internet zu ermöglichen. Intelligente Gateways könnten z. B. entscheiden, ob die von IoT-Geräten erhaltenen Daten lokal verarbeitet oder an Rechenzentren gesendet werden sollen [60], was zu einer besseren Nutzung der Netzwerkbandbreite beiträgt.

### 2.4.2 Edge Software System

Die Systemsoftware läuft direkt auf der Edge Hardware und verwaltet die Ressourcen für verteilte Edge Anwendungen. Die Systemsoftware umfasst Betriebssysteme, Virtualisierungssoftware und Kommunikationssoftware [57]. Die Systemsoftware muss mehrere Anwendungen von verschiedenen Agenten unterstützen und isolieren.

Bei der Systemvirtualisierung können mehrere Betriebssysteme auf einem einzigen physischen Rechner ausgeführt werden. Dies ermöglicht die Isolierung zwischen mehreren Agenten und die Partitionierung von Ressourcen. Dadurch kann sich bspw. der Ausfall eines Agenten nicht auf andere auswirken. Zur Unterstützung der Systemvirtualisierung können klassische virtuelle Maschinen, moderne Container und Migrationsmechanismen eingesetzt werden.

Eine virtuelle Maschine (VM) ist eine Menge von virtualisierten Ressourcen, die zur Emulation eines physischen Computers verwendet werden. Zu den virtualisierten Ressourcen gehören CPUs, Arbeitsspeicher, Netzwerke, Speichermedien [61] und sogar GPUs und FPGAs [62]. Virtualisierungssoftware, auch Hypervisor genannt z. B. [63], [64] virtualisiert die physischen Ressourcen und stellt sie in Form einer VM zur Verfügung. Der Agent installiert ein Betriebssystem und führt Anwendungen innerhalb der VM aus. Eine VM-Architektur ist auf der linken Seite der Abbildung 2.13 dargestellt.

Containers: Containers sind eine moderne Technologie, die eine leichtgewichtige Virtualisierung auf Prozessebene ermöglicht [65]. Container arbeiten mit einem einzigen Linux-Kernel, so dass sie im Vergleich zu virtuellen Maschinen keine zusätzliche Virtualisierungsschicht benötigen. Obwohl sie sich denselben Betriebssystemkern teilen, bieten sie dennoch die Prinzipien der Betriebssystemvirtualisierung, bei der jeder Benutzer eine isolierte Umgebung für die Ausführung von Anwendungen hat [66]. Die Architektur von Containern ist in der Mitte der Abbildung 2.13 dargestellt.

In Linux bieten Namespaces Containern eine eigene Sicht auf das System, und cgroups sind für die Ressourcenverwaltung wie die CPU-Zuweisung an Container zuständig. Diese leichtgewichtige Virtualisierung ermöglicht es Containern, schnell zu starten und zu stoppen und eine mit der nativen Umgebung vergleichbare Performance zu erreichen. Darüber hinaus werden Container in der Regel mit einer vorgefertigten Anwendung und ihren zugehörigen Libraries bereitgestellt, was das Deployment und die Orchestrierung von containerbasierten Anwendungen erleichtert. Zu den repräsentativen Container-Tools gehören LXC und Docker [67] sowie Kubernetes [68] für die Orchestrierung.

Eine Middleware bietet ergänzende Dienste zur Systemsoftware. Middleware Mechanismen im Edge Computing bieten Performance Monitoring, Koordination und Orchestrierung, Kommunikationsmöglichkeiten, Protokolle und so weiter. Die Architektur einer Middleware ist auf der rechten Seite der Abbildung 2.13 dargestellt.

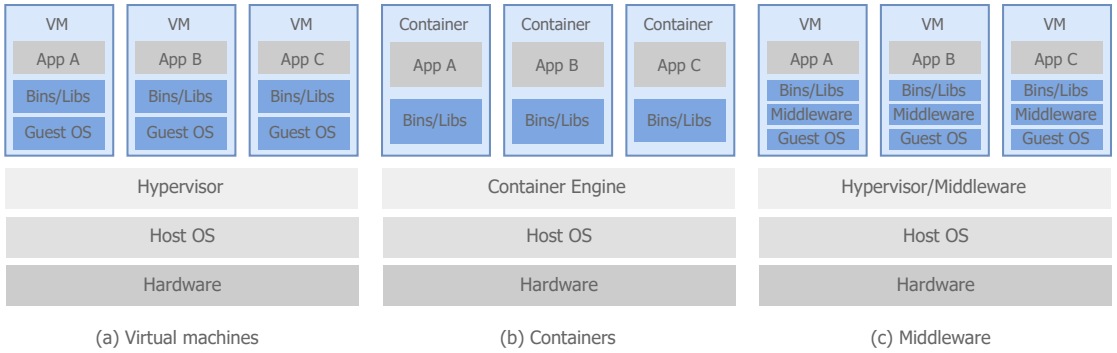



Abbildung 2.13: Architekturen von virtuellen Maschinen, Containern und Middleware für das Ressourcenmanagement im Edge Computing [57]

Zusammenfassend lässt sich sagen, dass an einer Edge Computing Infrastruktur verschiedene Hardware- und Softwarekomponenten beteiligt sind. Für den Fokus dieser Arbeit sind insbesondere Virtualisierungs- und Orchestrierungstechnologien relevant. Mit diesen kann die entwickelte Allokationsmethode im Betrieb integriert werden. Eine Analyse und der Aufbau eines Edge Frameworks zur Demonstration der entwickelten Ansätze ist in Kapitel 7.1 zu finden.

# Kapitel 3

# Stand der Technik

In diesem Kapitel werden aktuelle Ansätze zur Ressourcenverwaltung, Funktionsverteilung und Technologien im Kontext des Edge Computing analysiert. Darüber hinaus wird die Abgrenzung zu verwandten Arbeiten beschrieben.

Die vorliegende Arbeit beschäftigt sich mit autonomen und resilienten Recheninfrastrukturen für Edge Computing. Zu diesem Zweck wurden verschiedene wissenschaftliche Beiträge untersucht, um die Ausgangsfrage dieser Arbeit, wie eine effiziente Verteilung von Funktionen in Edge-Computing Systemen realisiert werden kann, zu analysieren. Mehrere aktuelle wissenschaftliche Veröffentlichungen befassen sich mit dem Problem der Ressourcenverwaltung in Edge Computing Systemen. Während es zahlreiche Forschungsarbeiten zur Ressourcenallokation mit einzelnen Optimierungszielen wie Energie, Latenz, Leistung usw. gibt, haben sich nur wenige Forscher mit Ansätzen zur Beschreibung verschiedener Effizienzziele für ein Allokationsproblem sowie mit der Integration von Allokationsmethoden zur Laufzeit beschäftigt.

## 3.1 Edge Computing und Ressourcenmanagement

Aus den Forschungsarbeiten der letzten Jahre geht hervor, welche Vorteile und welches Potenzial es hat, die Datenverarbeitung näher an die Datenquellen zu bringen. Shi et al. [1] gibt einen Überblick über die Visionen und Herausforderungen des Edge Computing. Einige Fallstudien konzentrieren sich auf Videoanalyse, Smart Home, Smart City Anwendungen und kollaborative Maschinen im Edge-Bereich. Unter anderem werden die Herausforderung des dynamischen Service-Managements und die Optimierungsmetriken für effiziente Systeme im Edge-Bereich erläutert.

Mehrere systematische Studien befassen sich mit Ansätzen für selbstadaptive Systeme [3] [69] und mit den Herausforderungen in Bezug auf Architekturen und Ressourcenmanagement, die für eine selbstadaptive Recheninfrastruktur in Edge Computing Systemen erforderlich sind [70] [71]. Es gibt in der Literatur übereinstimmende Hinweise darauf, dass die Zuweisung von Aufgaben und Ressourcen bei der Weiterentwicklung von Edge Computing Systemen eine entscheidende Rolle spielt.

In einer Ära sich schnell verändernder digitaler Technologiesysteme reicht jedoch eine statische Zuweisung und Verteilung von Ressourcen nicht mehr aus, um zu entscheiden, wie Funktionen in einem Edge Computing Netzwerk deployed werden sollten. Die Erkenntnisse aus älteren Studien müssen möglicherweise überarbeitet oder erweitert werden, um automatisch auf Veränderungen im System durch eine dynamische Umverteilung der Ressourcen im Edge Bereich zu reagieren.

## 3.2 Funktionsverteilung und Allokationsansätze

***Definition des Allokationsproblems:***

Die automatische Verwaltung von Ressourcen ist entscheidend für resiliente Edge Computing Systeme. Einer der Teilbereiche ist das Problem der Zuweisung von Aufgaben an Recheneinheiten im Edge Bereich. Die Theorie der Zuweisungsprobleme der letzten Jahrzehnte wird in [49] diskutiert, und in [2] wird die Anwendung dieser Theorie auf die Aufgabenzuweisung in Edge Computing Systemen erläutert.

Die in mehreren Studien verwendeten Entscheidungskriterien konzentrierten sich auf die Komponenten- und Systemebene, wobei hauptsächlich statische Ressourceninformationen verwendet wurden. Der Schwerpunkt lag dabei auf optimalen Lösungen für bestimmte Aspekte wie Zeit [72] [73] [74] [75] oder Energie [76] [77] [78]. Die Studien zeigen durchweg, dass die Definition des Zuordnungsproblems eine systematische Festlegung der Entscheidungskriterien oder Parameter und der Entwurfsziele erfordert. Außerdem hängen diese Ziele von der Art der Anwendung ab, die im Edge Netzwerk genutzt werden soll, und die Optimierung wird auf das gesamte System angewendet.

Daher ist die Übertragung spezifischer Optimierungsziele möglicherweise nicht für alle Anwendungen im Edge Bereich relevant. Vielmehr wäre es wichtig, einen Ansatz zur generischen Definition von Optimierungszielen auf verschiedenen Systemebenen bereitzustellen. Auf diese Weise könnten verschiedene Optimierungsziele auf der Komponenten- und Systemebene entsprechend den Anforderungen der Anwendungen kombiniert werden.

***Allokationsberechnung:***

Während die Wahl der Entscheidungskriterien und des Ziels das Zuteilungsproblem definieren, bestimmen die Lösungsalgorithmen und die Managementebene, auf der sie ausgeführt werden, die Ergebnisse sowie den Overhead der Allokation. In [79] werden verschiedene Konzepte und Scheduling Modelle zur Lösung der Ressourcenallokation im Edge Computing theoretisch dargestellt.

Tatsächlich gibt es, wie in [80] gezeigt, mehrere algorithmische Ansätze zur Lösung der Zuweisung, und nicht nur der Algorithmus selbst, sondern auch die Managementebene und der Umfang des Problems wurden im Detail untersucht [57] [2]. In einer systematischen Übersicht von [2] werden die Ansätze in zentralisierte [72] [73] [74], dezentralisierte [75] [77] [81] und verteilte Verwaltung [76] unterteilt. Aus der Literatur lässt sich keine einheitliche Aussage darüber ableiten, welcher Lösungsalgorithmus oder welche Managementebene grundsätzlich am besten für das Aufgabenzuweisungsproblem geeignet ist. Gleichzeitig zeigen die Studien, dass meist schnelle und realisierbare Allokationslösungen gegenüber rechenaufwändigen Algorithmen, die optimalere Allokationen finden, zu bevorzugen sind.

Aus diesem Grund wird es im Zeitalter sich schnell verändernder Edge-Computing-Systeme zunehmend wichtiger, automatisch realisierbare Allokationen zu finden und diese leicht an definierte Entscheidungskriterien und Ziele anzupassen. Dies impliziert auch einen reduzierten Umfang der an der Allokation beteiligten Komponenten, wie es beim dezentralen Managementansatz der Fall ist.

## 3.3 Technologien für die Reallokation im laufenden Betrieb

Neben der Wahl des Lösungsalgorithmus und der Verwaltungsebene, die sich auf die Allokationsergebnisse auswirken, tragen auch die Architekturen und Technologien, die für die Berechnung und das Deployment der Reallokationen zur Laufzeit verwendet werden, wesentlich dazu bei, auf Veränderungen in einem realen System zu reagieren. Eine Studie über die Verwendung von Containern für das selbstadaptive Deployment von Funktionen und deren Vergleich mit anderen Technologien wurde in [69] präsentiert.

In [70][79] und [82] werden verschiedene Technologien wie Hypervisoren und Container sowie Computing-Ansätze wie Edge und Fog Computing verglichen. Da Container für Edge Computing Systeme nicht nur für industrielle, sondern auch für Fahrzeuganwendungen zunehmend verbreitet sind [50], konzentrieren sich mehrere Studien auf das Task Offloading als Fallbeispiel für die Verschiebung von Funktionen mit Containern zur Laufzeit [73] [75] [77] [81] [78].

Es gibt in der Literatur einen Trend zur Verwendung von Containern anstelle von Hypervisor Virtual Machines, um Änderungen in Edge Computing Netzwerken schnell umzusetzen. Außerdem wird dies mit Orchestrator-Ansätzen in Verbindung gebracht, die Ressourcenmanagement-Funktionen umfassen, welche die Funktionsbereitstellung vereinfachen könnten. Die derzeit verfügbaren Ansätze müssen jedoch um intelligente Allokationsmethoden erweitert werden, die zur Laufzeit bedarfsgerechte Entscheidungen treffen können, um selbstadaptive Edge-Computing-Netze zu verwirklichen.

## 3.4 Abgrenzung

Aus der analysierten Literatur geht hervor, dass sich Forscher zunehmend mit selbstadaptiven Recheninfrastrukturen für Edge Computing beschäftigen. Die Zuweisung von Funktionen an Recheneinheiten ist eine Herausforderung, insbesondere wenn es darum geht, automatisch auf Veränderungen zu reagieren, wie es bei dem Fallbeispiel des Task Offloading vorgesehen ist. Im Rahmen dieser Arbeit wurden folgende Forschungslücken identifiziert und adressiert:

1. Die generische Beschreibung verschiedener Optimierungsziele, die eng mit dem Systemmodell für ein Zuordnungsproblem verzahnt sind.

2. Automatische Suche nach machbaren vor optimalen Allokationslösungen für eine schnelle und leichte Ausführung im laufenden Betrieb.

3. Die Integration intelligenter Allokationsmethoden mit Technologien zur Reallokation von Funktionen im laufenden Betrieb.

Damit trägt diese Arbeit zur Entwicklung von Allokationsansätzen für die Realisierung von autonomen und resilienten Recheninfrastrukturen für Edge Computing bei.

### *Abgrenzung von verwandten Arbeiten*

Es wurden verschiedene verwandte Arbeiten zur Allokationsberechnung im Edge Computing Kontext analysiert. Ein Auszug aus der Analyse ist in Tabelle 3.1 dargestellt. Dabei wurde die Analyse in vier Kategorien gruppiert. Für jede Kategorie wurde eine Leitfrage aufgestellt sowie der Ansatz der vorliegenden Arbeit im Vergleich beschrieben.

Tabelle 3.1: Abgrenzung von verwandten Arbeiten

| Verwandte Arbeiten | Entscheidungskriterien | | | | Allokationsziel | | | Managementebene | | | Allokationsberechnung | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Komponentenebene | Systemebene | Statische Ressourcen | Online Ressourcen | Zeit | Energie | Generisch | Zentral | Dezentral | Verteilt | Entwurfszeit | Betriebszeit |
| CH18 [72] | ✓ | | ✓ | | ✓ | | | ✓ | | | ✓ | |
| Liu+18 [73] | ✓ | | ✓ | | ✓ | | | ✓ | | | | ✓ |
| CL17 [74] | | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | ✓ | |
| JD19a [75] | ✓ | ✓ | | ✓ | ✓ | | | | ✓ | | | ✓ |
| XLL15 [76] | ✓ | | ✓ | | | ✓ | | | | ✓ | ✓ | |
| Ch+17 [77] | | ✓ | | ✓ | | ✓ | | | ✓ | | | ✓ |
| SW18 [81] | ✓ | | ✓ | | ✓ | ✓ | (✓) | | ✓ | | | ✓ |
| CZ17 [78] | | ✓ | ✓ | | ✓ | ✓ | | ✓ | | | ✓ | ✓ |
| **Ansatz** | ✓ | ✓ | ✓ | (✓) | (✓) | (✓) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Entscheidungskriterien:
Welche Informationsbasis wird für die Allokation verwendet?
Ansatz: Allokationsinformationen entlang der Systemebenen.

Allokationsziel:
Welches (Effizienz-)Ziel wird bei der Allokation verfolgt?
Ansatz: Komponentenspezifische Effizienz.

Management Ebene:
Auf welcher Ebene wird die Allokation durchgeführt?
Ansatz: Mehrstufige Ressourcenverwaltung.

Allokationsberechnung:
Wann wird die Allokation berechnet?
Ansatz: Fokus Ausführung im Betrieb nach Änderungen.

# Teil II

# Entwicklung einer effizienten und leichtgewichtigen Allokationsmethode für Edge Computing Systeme

# Kapitel 4

# Kontinuierlicher modellgetriebener Entwicklungsprozess für Adaption und Reallokation zur Laufzeit

In diesem Kapitel wird die Erweiterung eines Modellierungswerkzeugs und -prozesses für die kontinuierliche Entwicklung von Edge Systemen durch die Integration von Konzepten zur Allokationsbeschreibung erläutert.

Für intelligente und sich verändernde Systeme wird der Spalt zwischen notwendigen und etablierten Entwicklungsmethoden immer größer [83]. Die Zusammenarbeit verschiedener, oft weltweit verteilter Komponenten, Domänen und Entwicklungsteams ist unumgänglich. Damit steigt die Komplexität der Anforderungen an Komponenten, Teil- und Gesamtsystem. Der Zusammenhang ist in Abbildung 4.1 dargestellt.

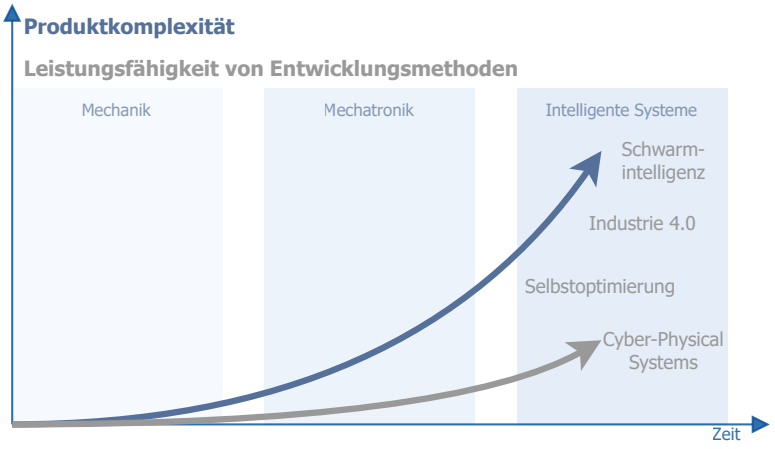



Abbildung 4.1: Lücke zwischen notwendigen und etablierten Entwicklungsmethoden [84]

Darüber hinaus müssen diese Systeme in der Lage sein, mit Veränderungen während des Betriebs umzugehen [3]. Für eine effiziente Entwicklung und Optimierung müssen nicht nur geeignete Methoden für die Entwurfsphase gefunden werden [84]; es sind auch flexible und kontinuierliche Entwicklungsprozesse notwendig, um Informationen und Aktivitäten zwischen Entwurf und Laufzeit näher zusammenzubringen.

## 4.1 Konzept der kontinuierlichen Adaption und Reallokation

Bei der Betrachtung kontinuierlicher Entwicklungsprozesse von sich verändernden Systemen konzentriert sich diese Arbeit auf die Aspekte der Adaption und Reallokation. Unter dem Begriff *Adaption* werden verschiedene Verfahren zusammengefasst, mit denen sich ein reaktionsfähiges System an äußere Ereignisse oder Störungen anpasst. In der Biologie wird dies auch als die Fähigkeit bezeichnet, sich perfekt an seine Umgebung und Situation anzupassen. Übertragen auf die Informationstechnik geht es darum, sich an veränderte Gegebenheiten anzupassen, um die Effizienz und Verfügbarkeit der Systeme nicht zu gefährden.

Im Fokus dieser Arbeit steht die Datenverarbeitungsinfrastruktur für Edge Systeme. Das heißt, es geht nicht direkt um die Adaption einer Anwendung selbst, sondern vielmehr um die Adaption der zugrunde liegenden Infrastruktur, auf der die Anwendung ausgeführt wird. Die Anpassungsfähigkeit der Datenverarbeitungsinfrastruktur ist von großer Bedeutung, denn ohne eine widerstandsfähige Infrastruktur können selbst die intelligentesten

Anwendungen nicht sicher, zuverlässig und robust ausgeführt werden. Wenn beispielsweise die Recheninfrastruktur aufgrund von Störungen wie dem Ausfall von Rechenknoten nicht in der Lage ist, die Anwendungen auszuführen, ist die Verfügbarkeit der Anwendungen ebenso gefährdet wie bei Fehlern in der Implementierung der Anwendung.

Diese Arbeit befasst sich mit der Fähigkeit, bei der Adaption der Datenverarbeitungsinfrastruktur zu entscheiden, wo welche Anwendungen oder Aufgaben am effizientesten ausgeführt werden sollen. Der Vorgang wird als Task Allokation oder, im Falle der Adaption einer bereits bestehenden Allokation, als *Reallokation* bezeichnet.

### 4.1.1 Analyse der Adaption und Reallokation in der Entwurfs- und Betriebsphase

Um die Reallokationsschritte im Entwicklungsprozess zu lokalisieren, ist es nicht nur notwendig, die Entscheidung über die Aufgabenzuweisung zu treffen, sondern es ist auch wichtig zu analysieren, wann die Entscheidung getroffen wird. Betrachtet man den gesamten Produktlebenszyklus, so kann die Entscheidung in der Entwurfsphase oder im Betrieb getroffen werden. Im Folgenden werden Alternativen aufgezeigt (siehe Abbildung 4.2) und deren Vor- und Nachteile im Hinblick auf die Verfügbarkeit des Systems diskutiert.

Im ersten Fall, welches der einfachste Prozess ist, wird das System, nachdem es in der Entwurfsphase entwickelt wurde, im Betrieb eingesetzt. Wenn das System aufgrund eines Ereignisses oder einer Störung nicht mehr verfügbar ist, bedeutet dies eine Ausfallzeit von Wochen bis Monaten.

Im zweiten Fall, der eine ***manuelle Reallokation*** vorsieht, wird nach der Störung im System eine Meldung übermittelt, im besten Fall mit den Informationen über die Störung und die betroffenen Komponenten. Mit diesen Informationen kann ein Redesign des Systems zurück in die Entwurfsphase erfolgen. Anschließend erfolgt ein Redeployment des Systems. Das kann in der Regel die Dauer des Ausfalls verkürzen, etwa auf Tage oder Wochen. Allerdings bedeutet das Redesign einen erhöhten Aufwand und Kosten für den Entwurf.

Im dritten Fall, der ***vorprogrammierten Reallokation***, erfolgt zusätzlich in der Entwurfsphase die Planung und Definition der vorprogrammierten Reallokationen. Im Betrieb kann die vorprogrammierte Reallokation dann automatisch nach dem Auftreten eines zuvor bekannten Ereignisses oder einer Störung durchgeführt werden. Bei der vorprogrammierten Reallokation werden beispielsweise beim Ausfall eines Rechenknotens die Aufgaben, die auf dem Knoten liefen, einem anderen vordefinierten Knoten neu zugewiesen. Die Dauer des Ausfalls hängt von dem intern definierten Mechanismus für die Funktionsverschiebung ab und liegt im Minutenbereich. Das ist bereits ein großer Vorteil gegenüber einer manuellen Umgestaltung oder Rekonfiguration. Gleichzeitig wird die Verfügbarkeit erhöht und

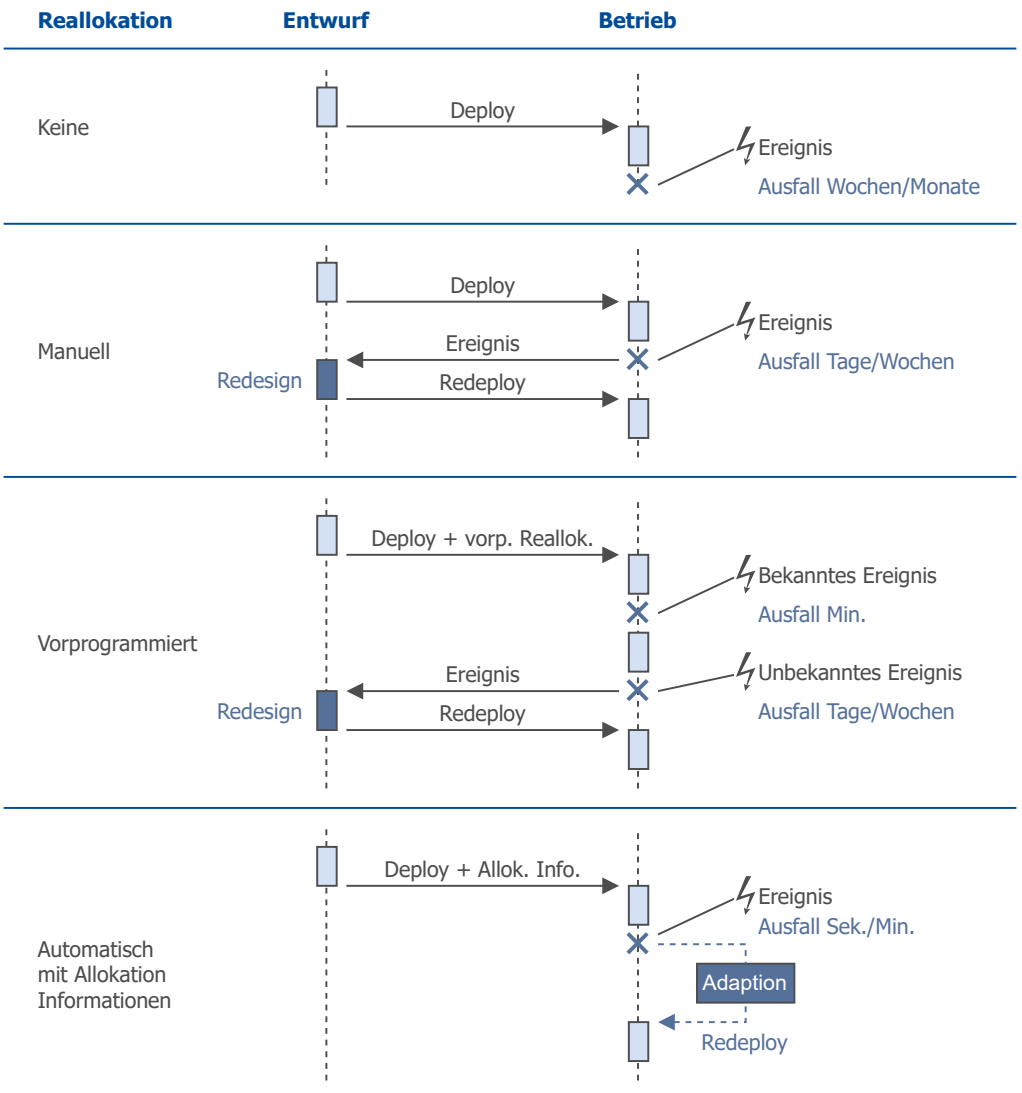



Abbildung 4.2: Reallokationsschritte während der Entwurfs- und Betriebsphase

die Ausfallzeit drastisch reduziert, während die Kosten und der Aufwand für ein Redesign oder eine Neukonfiguration vermieden werden. Tritt jedoch eine unbekannte Kombination von Ereignissen auf, verhält sich der Prozess wie im zweiten Fall und es muss ein manuelles Redesign durchgeführt werden.

Im vierten Fall, der ***automatischen Adaption*** in einem vordefinierten Rahmen, ist der erste Schritt nicht die Vorprogrammierung der Reallokation in der Entwurfsphase, sondern die Beschreibung von Informationen, damit die Allokation im Betrieb entschieden werden kann. Im Betrieb nach dem Auftreten von Ereignissen findet dann die Adaptionsphase statt. In dieser Phase kann das System eine neue Aufgabenzuweisung auf der Grundlage der notwendigen Informationen aus der Entwurfsphase und der Informationen über das eingetretene Ereignis berechnen. Das heißt, es wird automatisch entschieden, welche Aufgaben auf welchen Rechnerkomponenten ausgeführt werden sollen. Diese wird dann automatisch umgesetzt bzw. umverteilt. So kann die Ausfalldauer auf einen Bereich von Sekunden bis Minuten reduziert werden, abhängig von der Dauer des Zuweisungsalgorithmus und den Mechanismen zur Umverteilung. Wie bei der vorprogrammierten Reallokation wird dadurch die Dauer der Ausfälle reduziert und die Kosten für die manuelle Umgestaltung vermieden.

Der letzte Fall wurde in dieser Arbeit gewählt, da er die Verfügbarkeit des Systems durch eine automatische Adaption erhöhen kann. Dieses Verfahren wurde der vorprogrammierten Reallokation vorgezogen, da insbesondere bei der zunehmenden Komplexität von Rechnerinfrastrukturen die Definition einer begrenzten Anzahl von Ereignissen nicht ausreichend ist. Bei der Komplexität der Systeme kann schnell eine manuell nicht kontrollierbare Kombination von Ereignissen auftreten. Es ist daher unmöglich, alle möglichen Kombinationen von Reallokationen im Voraus zu definieren.

Aus diesem Grund wurde ein Konzept entwickelt, mit dem die Reallokation automatisch im laufenden Betrieb durchgeführt werden kann, indem die notwendigen Informationen im Entwurf definiert werden. Wie die notwendigen Informationen im Entwurf definiert werden können, wird im Kapitel 5 diskutiert. Wie die Allokation auf dieser Basis im Betrieb durchgeführt werden kann, wird im Kapitel 6 erläutert.

### 4.1.2 Kontinuierlicher Prozess mit Schritten zur Adaption und Reallokation

Der Ansatz für einen kontinuierlichen Entwicklungsprozess in dieser Arbeit ist die explizite Berücksichtigung von Adaption und Reallokation in der Betriebsphase, sowie die Definition der notwendigen Informationen in der Entwurfsphase. Für eine Darstellung des Prozesses lehnt sich das Konzept an das kontinuierliche Entwicklungsparadigma DevOps an.

Das DevOps Paradigma [85] eignet sich für die iterative und agile Entwicklung neuer Funktionen. Bei dem Prozess wird eine enge Verknüpfung von Entwicklung und Betrieb

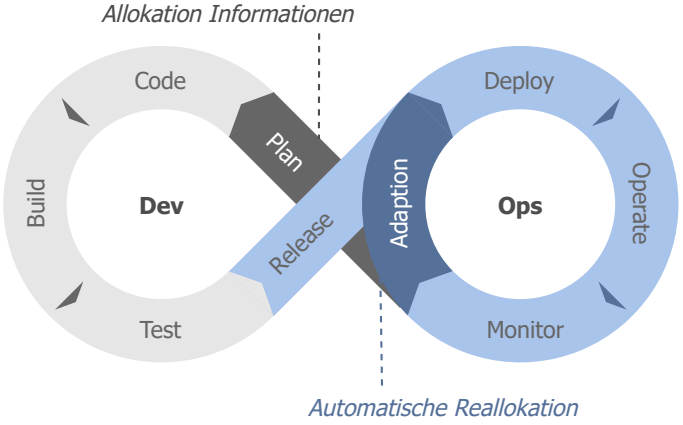



Abbildung 4.3: DevOps Prozess mit relevanten Phasen für Adaption und Reallokation

angestrebt. Mit einer zusätzlichen modellbasierten Unterstützung des Prozesses können Modelle während der gesamten Entwicklungs- und Betriebsphase eingesetzt werden, um den gesamten Prozess effizienter zu gestalten [86]. Darauf aufbauend ist ein Teilkonzept dieser Arbeit die Integration der notwendigen Schritte für eine Reallokation in DevOps, um einen kontinuierlichen Prozess zu erreichen (siehe Abbildung 4.3).

Im ***Entwurfszyklus (Dev)*** ist die Phase Plan besonders relevant. Hier wird das System geplant und entworfen. In dieser Phase sollten zusätzlich zu den Modellen der Systemarchitektur die benötigten Informationen für die Allokation definiert werden. Während des Entwurfs oder beim Übergang von Dev zu Ops kann die Zuordnung von Aufgaben zu Hardwarekomponenten automatisch auf der Grundlage der zuvor definierten Informationen berechnet werden.

Im ***Betriebszyklus (Ops)*** ist der Übergang von Monitor zu Deploy besonders wichtig. Diese Phase ist eine implizite Planungsphase, die vom System automatisch gesteuert werden sollte. Das wird in dieser Arbeit als Adaption bezeichnet, um die notwendigen Schritte zur automatischen Entscheidung über das Deployment zur Laufzeit zu integrieren. Zuvor werden in der Monitorphase Änderungen im System und im Kontext überprüft. Wenn eine Anpassung des Systems erforderlich ist, wird eine neue Allokation für das System berechnet. Die Berechnung erfolgt auf der Grundlage der in der Planungsphase definierten Informationen, so dass sie mit den Entwurfsdefinitionen übereinstimmen. Die neu berechnete Allokation wird dann in der Errichtungsphase angewendet. Auf diese Weise bleibt das System in der Betriebsphase, indem es sich selbst anpasst, wodurch Ausfallzeiten verringert und die Verfügbarkeit erhöht werden.

### 4.1.3 On-Demand Ereginisse für Adaption

Für die Reallokation im Betriebszyklus (Ops) sind das Monitoring und die anschließende Adaption zwei wichtige Bestandteile. In diesem Kapitel wird analysiert, wann das gesche-

hen kann, und es wird eine Liste von On-Demand Ereignissen gezeigt, um die Relevanz der Adaption zu verdeutlichen. Die Liste der Ereignisse basiert auf den vom Autor in [87] vorgestellten Reallokationsszenarien und einer Übersichtsstudie über dynamische Ereignisse in selbstadaptiven Internet of Things (IoT) Systemen [88], in der analysiert wurde, welche dynamischen Ereignisse im Edge Kontext die häufigsten Ursachen für die Auslösung von Adaptionen in einem IoT System sind.

On-demand Ereignisse, auch dynamische Events genannt, sind Ereignisse, die im System auftreten und eine Veränderung bewirken. In dieser Arbeit werden sie als on-demand bezeichnet, weil damit die Fähigkeit beschrieben wird, Anforderungen oder Änderungen zeitnah zu erfüllen. Es handelt sich also um Ereignisse, auf die das System reagieren muss, um die Verfügbarkeit des Systems nicht zu gefährden.

Die On-Demand Ereignisse wurden nach den Komponenten klassifiziert, die davon betroffen sind: die Recheneinheiten, hier Nodes genannt, oder die auszuführenden Funktionen, hier Tasks genannt. Außerdem wurden die Ereignisse danach klassifiziert, ob eine Komponente hinzugefügt wird +, ausfällt bzw. ein Problem hat ! oder geändert wird ↻. Eine Liste der On-Demand Ereignisse ist in den Tabellen 4.1 und 4.2 zu sehen.

Tabelle 4.1: On-Demand Ereignisse für Tasks
*Komponenten hinzugefügt +, ausgefallen ! oder geändert ⟳*

| **Ereignis** | **Task** | **Beschreibung** | **Beispiel** |
|---|---|---|---|
| Neue Funktionen | + | Neue Funktionen müssen auf das bestehende Edge Netzwerk verteilt werden | Aufgrund neuer UseCases oder Sensoren können im Laufe der Zeit neue Funktionen hinzukommen. |
| Zusätzliche Funktionen | + | Für eine Anwendung können je nach Situation mehrere Instanzen einer Funktion erforderlich sein. | Wenn bspw. bei der Kameraüberwachung ein größerer Anwendungsbereich abgedeckt werden soll oder ein Bereich nach dem Auftreten von Warnmeldungen genauer kontrolliert werden soll. |
| Softwareausfall und Alterung | ! | Ausgeführte Anwendung ist wegen Fehlverhaltens nicht erreichbar | Softwareausfälle, bspw. aufgrund von Speicher-, Hardware- oder Konnektivitätsfehlern. Dies führt z. B. zu erhöhten Latenzzeiten und nicht ansprechbaren Diensten. |
| Cyberangriffe auf Funktionen | ! | Die ausgeführte Anwendung kann kompromittiert werden und entweder nicht mehr verfügbar sein oder es können Datenschutz- oder Vertraulichkeitsprobleme auftreten. | Von Angreifern verursachte Ereignisse, die die Vertraulichkeit, Integrität und Verfügbarkeit des Systems verletzen. Zum Beispiel Denial of Service (DoS) oder Man-in-the-Middle Angriffe. |
| Update von Funktionen | ⟳ | Bei der Aktualisierung einer Funktion müssen diese und eventuell neu hinzukommende Funktionen verteilt werden. | Wenn eine neue Version für Funktionen verfügbar ist, z. B. um eine Sicherheitslücke zu schließen. |

Tabelle 4.2: On-Demand Ereignisse für Nodes
*Komponenten hinzugefügt +, ausgefallen ! oder geändert ⟳*

| **Ereignis** | **Node** | **Beschreibung** | **Beispiel** |
|---|---|---|---|
| Neue Nodes | + | Zusätzliche Nodes mit mehr Ressourcen werden integriert, so dass effizientere Zuweisungen gefunden werden können. | Wenn Tasks effizienter auf dedizierter Hardware wie KI-Beschleunigern (z. B. Tensor Processing Units (TPUs)) ausgeführt werden können und diese in einem neuen Hardware-Node verfügbar sind. |
| Ausfall des Nodes | ! | Ausgeführte Tasks sind nicht mehr verfügbar, weil der Node, auf dem sie ausgeführt wurden, ausgefallen ist. | Wenn ein Node ausfällt, bspw. aus elektrischen Gründen oder aufgrund von Umweltbedingungen. |
| Cyberangriffe auf Nodes | ! | Ausgeführte Tasks sind nicht mehr vertrauenswürdig und müssen in einem anderen Node ausgeführt werden. | Wenn ein Node aufgrund von Angriffen oder Sabotage nicht oder nur eingeschränkt genutzt werden kann. |
| Überlastete Nodes | ! | Einige Tasks können nicht bedarfsgerecht ausgeführt werden und müssen neu zugewiesen werden, um den Node zu entlasten. | Wenn zusätzliche Datenübertragung und -verarbeitung erforderlich ist. Infolgedessen können die Edge Nodes mit der Verarbeitung überlastet werden, was zu Verzögerungen, Ausfallzeiten oder Nichtverfügbarkeit führt. |
| Mobilität der Teilnehmer | ! | Durch die Bewegung eines Teilnehmers können die ausgeführten Tasks nicht erreicht werden und müssen in einem erreichbaren Node neu zugewiesen werden. | Wegen der permanenten Bewegung von Komponenten und der Heterogenität der Kommunikationstechnologie. Wenn das Gerät den Standort wechselt und eine Verbindung zu anderen Edge Nodes herstellen muss, um eine bessere Latenzzeit zu erreichen. |
| Austausch eines Nodes | ⟳ | Beim Austausch eines Nodes müssen die zuvor auf dem Node ausgeführten Tasks neu verteilt werden. | Wenn ein bisheriger Node ersetzt werden muss, z.B. zugunsten eines leistungsfähigeren Nodes oder aufgrund von Alterung. |

## 4.2 Modellgetriebener Entwicklungsprozess und -werkzeug für Edge Systeme

Die kontinuierliche Adaption erfordert unter anderem die Durchgängigkeit der Informationen über den gesamten Produktlebenszyklus, wofür die modellgetriebene Entwicklung eingesetzt wird. Um in der Betriebsphase automatisch über die Allokation entscheiden zu können, müssen die relevanten Informationen in der Entwurfsphase formal so beschrieben werden, dass sie weiterverwendet werden können.

Das modellbasierte Systems Engineering (MBSE) hat sich als Systems Engineering Ansatz des 21. Jahrhunderts positioniert. Seit der Standardisierung der Systems Modeling Language (SysML) durch die Object Management Group (OMG) im Jahr 2007 gewinnt sie in Industrie und Wissenschaft an Popularität [83].

Ein modellgetriebener Entwicklungsprozess ermöglicht die Definition von Informationen im Entwurf und auf dieser Basis können die notwendigen Informationen für die Entscheidung der Reallokation im Betrieb extrahiert werden.

Modelle werden in der Systementwicklung häufig eingesetzt, um die Systemkomplexität zu reduzieren. Das Model Driven Engineering ist eine ursprünglich aus der Softwareentwicklung stammende Methodik, die den Entwicklungsprozess für komplexe Softwaresysteme effizienter und die Komplexität des Systems durch Abstraktion beherrschbarer macht. Beim Model Driven Engineering werden Modelle nicht nur zur Dokumentation, sondern auch als erstklassige Artefakte in der Entwicklung eingesetzt. Modelle können zur Beschreibung des gesamten Systems, zur Generierung von ausführbarem Programmcode oder zur Systemanalyse verwendet werden [89]. Die Modelle bilden somit eine Grundlage für verschiedene Entwicklungsaktivitäten (Abbildung 4.4). Hierbei können die Modelle eine Unterstützung für den gesamten Produktlebenszyklus sein.

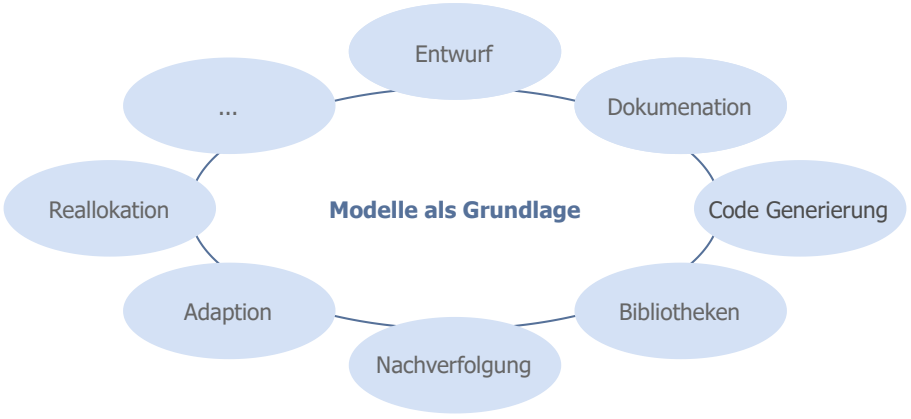



Abbildung 4.4: Modelle als Grundlage für Entwurf und Analyse während des gesamten Produktlebenszyklus

Daher wird in dieser Arbeit insbesondere die Nutzung von Modellinformationen für die Reallokation von Funktionen im Betrieb betrachtet. Damit die Informationen für die Allokation extrahiert werden können, müssen sie jedoch zunächst im Modell so beschrieben und annotiert werden, dass eine automatische Interpretation und Extraktion erfolgen kann.

Ein Ansatz zur Formalisierung der Modellierung ist die Erstellung von Metamodellen, die als Grundlage für die Modelle verwendet werden. Dies wird als Metamodellierung bezeichnet. In dieser Arbeit wurden Metamodelle entwickelt, um die notwendigen Informationen für die Allokation formal zusammen mit dem Systemmodell zu beschreiben [89].

1. Metamodell: Ein Metamodell ist ein Modell, das die Komponenten und Beziehungen zwischen Modellen mit Einschränkungen und Regeln definiert. Ein Metamodell besteht aus den folgenden Elementen: abstrakte Syntax, konkrete Syntax, statische Semantik und dynamische Semantik.

2. Metamodellierung: Metamodellierung ist der Prozess der Analyse, Konstruktion und Entwicklung der Regeln, Einschränkungen und Beziehungen zur Definition eines Metamodells für einen bestimmten Bereich. Die Metamodellierung ist ein wesentlicher Schritt bei der Modellbasierten Entwicklung.

### 4.2.1 Analyse von Ansätzen zur Beschreibung von Industrie 4.0 Systemen

Für die Entwicklung von Edge Systems wurden verschiedene Ansätze unter den Begriffen Industrie 4.0 (I4.0) und Cyber Physical Systems analysiert, um eine Grundlage für diese Arbeit zu wählen.

Es existieren mehrere Ansätze, die sich mit der Beschreibung und Konfiguration von Industrie 4.0 Systemen befassen; diese wurden im Rahmen des SysKit-HW Projekts analysiert. Die für diese Arbeit relevantesten Ansätze werden im Folgenden zusammengefasst.

Die verschiedenen Ansätze haben unterschiedliche Beschreibungsformen und unterschiedliche Schwerpunkte. So gibt es beispielsweise Ansätze, die SysML und das Paradigma der modellbasierten Sprache für die Entwicklung mechatronischer Systeme verwenden, um eine Möglichkeit zur Wiederverwendung von Komponenten, Modulen und Subsystemen in einer Anlage zu ermöglichen [90]. Ein anderer Ansatz verwendet eine formalisierte Prozesssprache, um modularisierte Produktionssysteme zu beschreiben. Diese Beschreibung erlaubt es, den Informationsaustausch von Prozessen und ihren Services zu formulieren. Das ermöglicht eine serviceorientierte Architektur auf Modulebene [91], [92]. Darüber hinaus gibt es modellbasierte Konzepte zur Abbildung von Softwarearchitekturen. Für den Entwurf von eingebetteten Systemen wird die Modellierungsmethodik Software Platform Embedded Systems (SPES) verwendet. Am Beispiel einer Meerwasserentsalzungsanlage

wird SPES eingesetzt, um eine durchgängige Modellierung der Anforderungen von der funktionalen, logischen bis hin zur technischen Architektur zu demonstrieren [93].

Die aufgezeigten Ansätze zur Beschreibung von I4.0 Systemen adressieren jeweils nur bestimmte Teilaspekte, die für eine konsistente, modellbasierte Beschreibung notwendig sind. Für die Analyse und Konfiguration einer Anlage mit verteilten Funktionen auf intelligenten Geräten werden weitere Ansätze benötigt. Für die in dieser Arbeit entwickelten Ansätze sind die Modellierung und Konsistenz der für die Allokation notwendigen Informationen von besonderer Bedeutung.

### 4.2.2 Modellgetriebenes Industrie 4.0 Entwicklungswerkzeug

Im Rahmen des Projektes SysKit-HW[1] [94] wurde ein Industrie 4.0 (I4.0) Entwicklungswerkzeug konzipiert und umgesetzt. Für diese Arbeit wurde es als Grundlage genommen und unter Berücksichtigung der zuvor genannten Analyse und Kriterien angepasst.

Das Industrie 4.0 Entwicklungswerkzeug wurde unter Verwendung verschiedener Abstraktionsebenen, Bibliotheken und den Konzepten der modellbasierten Entwicklung realisiert. Eine Übersicht über die Struktur des I4.0 Entwicklungswerkzeugs ist in Abbildung 4.5 zu sehen. Für die Beschreibung von I4.0 Systemen sind verschiedene Ebenen von der Anforderungs- bis zur System- und Topologieebene vorgesehen. Die Wiederverwendung von Modulen wird durch Bibliotheken ermöglicht.

Für die modellbasierte Beschreibung und Analyse des Systems werden unterschiedliche Modelle für die verschiedenen Systemebenen benötigt. Daher wurden entsprechende Metamodelle für jede Ebene definiert. In den Metamodellen werden auch die Beziehungen zwischen den Modellen definiert, so dass die Informationen für die Allokation durch Modellabfragen und programmatische Metriken unterstützt werden können. Eine detaillierte Beschreibung der Ebenen folgt in den nächsten Abschnitten. In diesem Kapitel werden die Ebenen zur Modellierung des Gesamtsystems, auch Systemmodell genannt, erläutert. Die detaillierte Beschreibung des Konzepts zur Modellierung der für die Allokation notwendigen Informationen wird im Kapitel 5.2 erläutert.

Die Definition der notwendigen Abstraktionsebenen für die Beschreibung von I4.0 Systemen [95] erfolgte im Projekt SysKit-HW in Anlehnung an die Sicherheitsstandards IEC 62443 und unter Anwendung der Methodik für die szenariobasierte Entwicklung von Industrie 4.0 Beschreibungssprachen [96]. Die Ebenen des I4.0 Entwicklungswerkzeugs werden dabei in verschiedene technische Domänen unterteilt. Mit den verschiedenen Ebenen und Verknüpfungen zwischen den Ebenen wird eine konsistente Spezifikation des Systems erreicht, wie sie in den Normen beschrieben ist.

[1] https://www.syskit-projekt.de/. Accessed 2. Aug. 2022

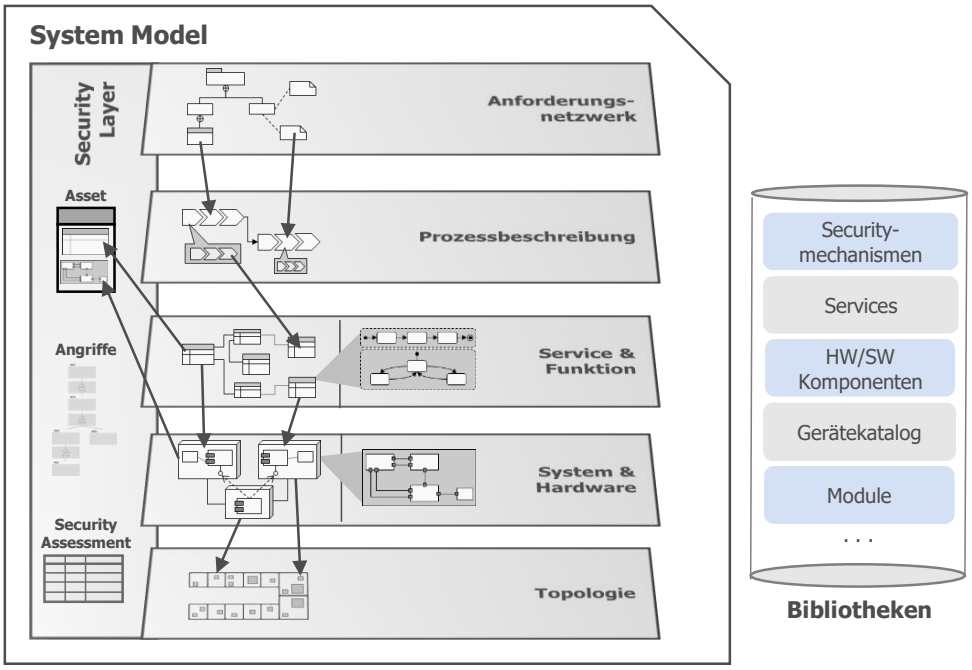



Abbildung 4.5: Modellierungsansatz des Industrie 4.0 Entwicklungswerkzeugs mit verknüpften Ebenen und Bibliotheken [95]

**Anforderungsnetzwerk:** Auf der Ebene werden Anforderungen an das System beschrieben, damit sie später mit anderen Anforderungen und Modellelementen aus den anderen Ebenen verknüpft werden können. Das unterstützt die Nachvollziehbarkeit und ermöglicht es, abzubilden, welche Systemelemente durch welche Anforderungen realisiert werden. Auf dieser Ebene können Rahmenbedingungen und Optimierungsziele definiert werden, die später für die Allokation relevant sind.

**Prozessbeschreibung:** In der Prozessbeschreibung wird die logische Abfolge der Aktivitäten im System beschrieben, so dass die gewünschte Funktionalität des Systems auf einer höheren Ebene definiert werden kann, ohne dass detaillierte Kenntnisse über die benötigten Hard- und Softwarekomponenten vorliegen.

**Serviceebene/Verhaltensbeschreibung:** Auf der Serviceebene werden die verschiedenen erforderlichen Services und das Verhalten des Systems beschrieben. Das erfolgt mit Hilfe von Diagrammen, die auf der UML basieren, wie z. B. Klassendiagrammen, Sequenzdiagrammen oder Zustandsdiagrammen. Es besteht eine direkte Verbindung zwischen den auf dieser Ebene beschriebenen Services und den auf der Prozessebene beschriebenen Aktivitäten.

**System/HW Ebene:** Auf der Systemebene wird die eigentliche physikalische Umsetzung des I4.0 Systems beschrieben. Hier werden u.a. die verwendeten Hardwarekomponenten und die physikalische Kommunikation zwischen den Komponenten definiert. Dazu

werden im I4.0 Entwicklungswerkzeug u.a. Komponenten- und Vernetzungsdiagramme bereitgestellt. Speziell für die Allokation ist das Zusammenspiel der Ebenen der Services und HW relevant, sowie ihre Beschreibung, so dass die Allokation im Betrieb berechnet werden kann.

**Topologie:** Bei Industrie 4.0 Anlagen kann der Standort von Anlagenkomponenten für Analysen wesentlich sein, z. B. um zu beurteilen, wie eine Allokation auf verschiedene Anlagenbereiche räumlich realisiert werden kann, oder um Randbedingungen zu definieren. Auf dieser Ebene können Gebäude- und Umgebungspläne genutzt werden, um eine Zuordnung der Lage der Komponenten vorzunehmen. Es kann zwischen statischen und beweglichen Komponenten unterschieden werden.

### 4.2.3 Konzept für die modellgetriebene Entwicklung mit kontinuierlicher ereignisbasierter Adaption

Um den Entwurf, die Konfiguration und den Einsatz von komplexen dynamischen Edge Computing Systemen zu systematisieren, präsentiert diese Arbeit ein Framework zur Modellierung und einen Entwicklungsprozess, der die Neuzuordnung von Services mit einbezieht. Die Modellierung basiert auf den in den vorangegangenen Kapiteln beschriebenen Konzepten zur expliziten Berücksichtigung von Adaptionen und zur Beschreibung von vernetzten, serviceorientierten Industrie 4.0 Systemen. Der SysKit Ansatz wurde erweitert, um ihn an das DevOps Paradigma anzupassen. Auf der Grundlage der Modellbeschreibungen werden dann Allokationen berechnet und Deployment Konfigurationen generiert. Mit der Modellunterstützung sind Reallokationsszenarien zur Laufzeit möglich.

Der vorgesehene Entwicklungsprozess ist in Aktivitäten zur Entwurfszeit und zur Laufzeit unterteilt, wie in Abbildung 4.6 dargestellt. Der Schwerpunkt liegt darauf, jeden Schritt mit modellbasierten Techniken zu unterstützen, um die Komplexität zu reduzieren und die automatische Reallokation zu erleichtern. Die Entwurfsaktivitäten sind wie folgt definiert:

In der Dev Phase wird der Plan-Schritt detaillierter betrachtet. Es werden die Systemebenen modelliert und die Allokationsinformationen werden verzahnt beschrieben. Modelliert werden u.a. die Umgebung, das Anwendungsszenario, die servicebasierten Funktionen sowie die Wirkungsstruktur und die HW/SW-Komponenten. Dazu werden insbesondere Services auf der Basis der in [97] vorgestellten modellbasierten Servicebeschreibungssprache beschrieben. Die Beschreibung der Dienste wird von den HW/SW-Komponenten entkoppelt, so dass die Allokation der servicebasierten Funktionen auf die HW-Komponenten nachgelagert erfolgen kann.

Zusammen mit dem Systemmodell werden die notwendigen Informationen für die spätere Allokationsberechnung verzahnt modelliert. Die Idee dabei ist, dass auf jeder Ebene und

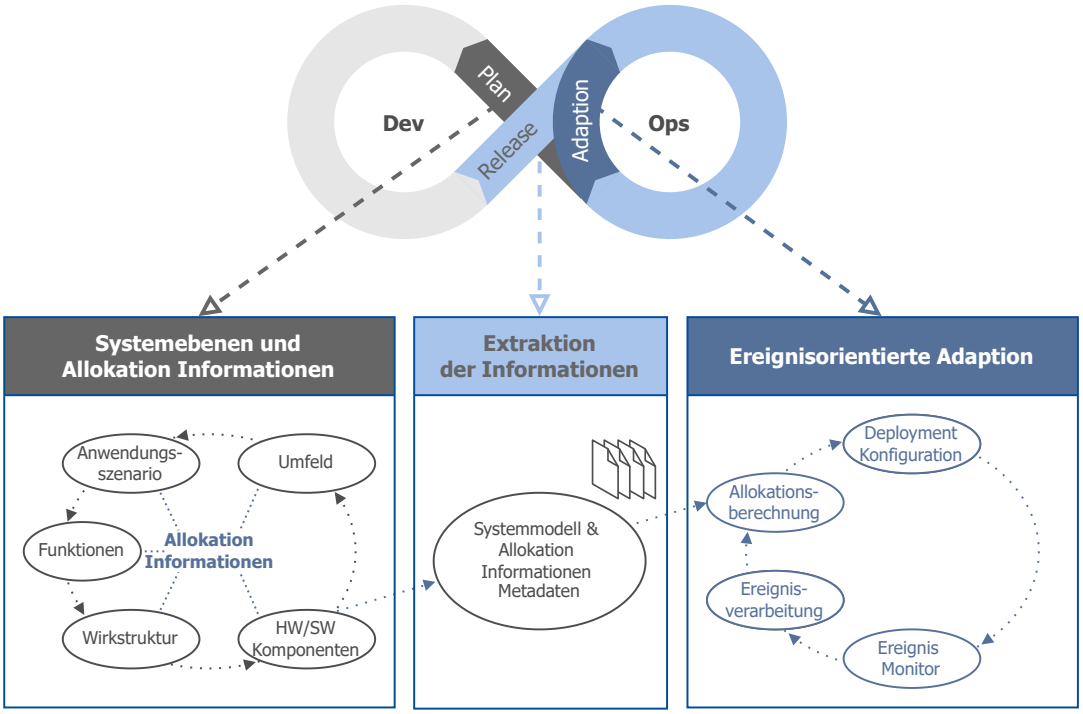



Abbildung 4.6: Konzept der modellgetriebenen Entwicklung mit kontinuierlicher ereignisbasierter Adaption

bei jeder Komponente Informationen wie Constraints oder Optimierungsziele definiert werden können, die später bei der Allokation berücksichtigt werden müssen. Das detaillierte Konzept und Metamodell hierfür wird im Kapitel 5.2 erläutert.

Beim Übergang von der Entwicklungs- zur Betriebsphase müssen die notwendigen Informationen aus der Planungsphase extrahiert werden, um die Metadateninformationen für die Allokation während der Betriebsphase auf schlanke Weise zu nutzen. Hintergrund ist, dass die Systemmodelle weitaus mehr Informationen und Details enthalten können als die für die Allokation benötigten Informationen. Benötigte Informationen sind bspw. Kennzahlen zu Ressourcen, Ausführungskosten oder Beschränkungen. Um eine schnelle Berechnung der Allokation im Betrieb durchführen zu können, ist es von Vorteil, die zugrunde liegenden Daten im Vorfeld auf das Notwendigste zu reduzieren.

Im Ops Zyklus wird der Adaptions-Schritt ausführlicher betrachtet. Für die ereignisbasierte Adaption werden die Ereignisinformationen verwendet, die während des Betriebs von Monitoren gesammelt werden. Die Ereignisinformationen werden zunächst aufbereitet, um bspw. zu ermitteln, welche Funktionsumfänge und HW-Komponenten betroffen sind. Mit diesem Wissen und den Systemmodell Metadaten wird die Allokation im laufenden Betrieb vorgenommen. Die Ergebnisse werden in Deployment Dateien und Konfigurationen übersetzt, so dass anschließend auf der Basis ein neues Deployment durchgeführt werden kann.

## 4.3 Umsetzung und Erweiterung eines Modellierungswerkzeugs für kontinuierliche Adaption und Reallokation

Für die Umsetzung der zuvor beschriebenen Ansätze wurde ein Entwicklungswerkzeug entwickelt und erweitert auf der Basis des Industrie 4.0 (I4.0) Entwicklungswerkzeuges vom Projekt SysKit-HW.

### 4.3.1 Eingesetzte Frameworks und Tools

Das Entwicklungsframework ist auf Basis des Eclipse Modeling Framework entwickelt. Die benutzten Frameworks sind open source und unter der Eclipse Public License verwendbar. Ein Diagramm der Softwarearchitektur des Werkzeuges ist in Abbildung 4.7 zu sehen. Im Folgenden werden die einzelnen Komponenten der Architektur beschrieben. EcoreTools dient der graphischen Erstellung von Ecore-Metamodellen. Mit dem Framework Sirius werden die graphischen Editoren für die Modelle implementiert. Für die Überprüfung der statischen Syntax der Metamodelle wird das Framework Object Constraint Language (OCL) benutzt. Mit Hilfe dieser Frameworks werden die technischen Metamodelle (Domain Specific Language – DSL) entwickelt. Für die Traversierung und Extraktion von Modellinformationen sowie das Generieren von Metadaten-Dateien wird das Framework Acceleo eingesetzt. Dazu existieren verschiedene Schnittstellen und weitere Plug-Ins, um andere Werkzeuge zu integrieren.

Darauf aufbauend werden drei Hauptteile entwickelt: Systemmodelle, Anwendungsszenarien und Abfragen. Diese werden durch einen Bibliotheksmechanismus unterstützt, womit Modelle, Szenarien und Abfragen wiederverwendet werden können. Durch das Zusammenspiel von den drei Teilen und die integrierte verzahnte Modellierung der Allokationsinformationen wird die spätere Extraktion der notwendigen Daten für eine kontinuierliche Adaption ermöglicht.

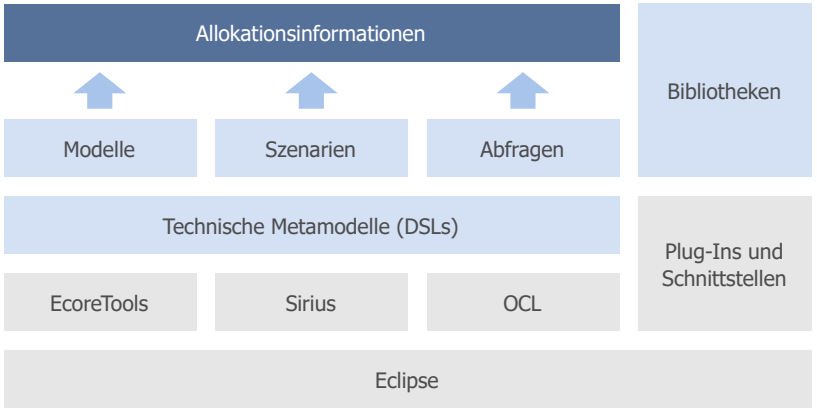



Abbildung 4.7: Architektur des Modellierungswerkzeuges

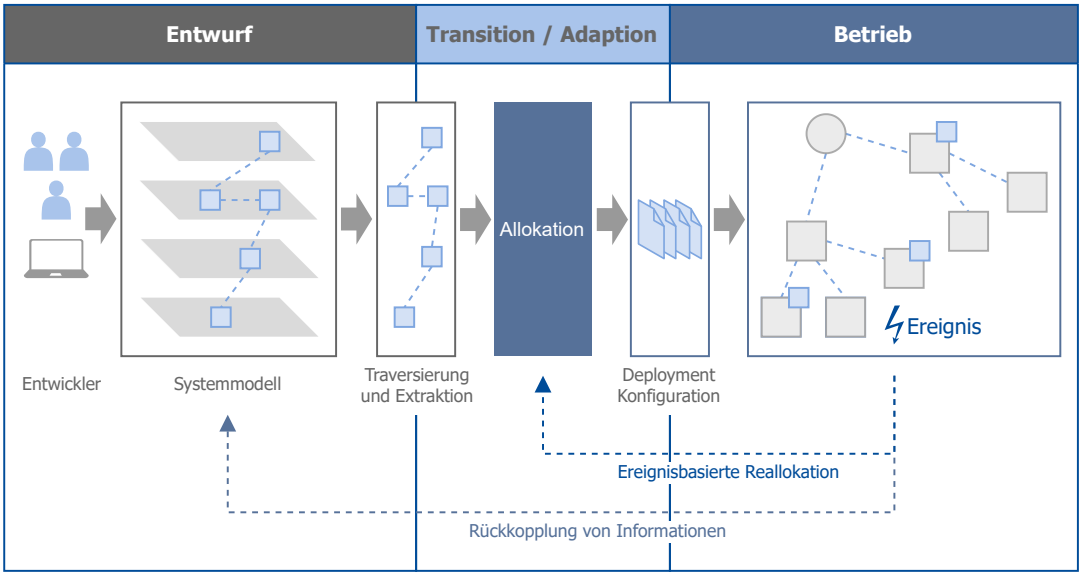



Abbildung 4.8: Modellierungswerkzeug im kontinuierlichen Entwicklungsprozess

Als Grundlage für die Modellierung wurde das Eclipse Modeling Framework gewählt, da es als opensource zugänglich ist, was die Entwicklung, aber auch den späteren Einsatz und eine Erweiterung erleichtert. Außerdem steht eine große Community im Hintergrund durch die Eclipse Organisation. Die Umsetzung der Metamodelle basiert auf eigenentwickelten Beschreibungssprachen (DSLs) statt bspw. SysML Modellen, weil dadurch vereinfachtere und gezieltere Beschreibungssprachen für die während der Arbeit analysierten Domänen möglich waren. Die Ansätze dieser Arbeit limitieren sich jedoch nicht auf die hier umgesetzten Metamodelle und können durchaus in einem SysML basierten Tool integriert werden. An der Stelle ist das EMF-basierte Modellierungswerkzeug als eine Demonstrationsplattform für die entwickelten Ansätzen zu verstehen.

### 4.3.2 Modellierungsprozess und Werkzeug

Abbildung 4.8 gibt einen Überblick über das implementierte Modellierungsframework und wie es in den entworfenen Adaptionsprozess integriert ist. In Abbildung 4.9 ist ein Screenshot des Modellierungswerkzeugs zu sehen.

In der Entwurfsphase definieren die Entwickler mit Hilfe des Modellierungswerkzeugs das Systemmodell mit den verschiedenen Ebenen. Zu dem Zeitpunkt werden die mit EMF und Sirius entwickelten grafischen Editoren verwendet. Mit Hilfe des Acceleo Frameworks werden die Modelle durchlaufen und die für die Allokation erforderlichen Informationen extrahiert. Die Ergebnisse dieses Schrittes werden in Metadaten-Dateien gespeichert. Zu dem Zweck wurden JSON- und XML-basierte Dateien verwendet. In der Transition oder direkt im Betrieb werden auf Basis der Metadaten und eventuell aufgetretener Ereignisse Allokationen berechnet. Darüber hinaus wurden Schnittstellen für die Rückführung von

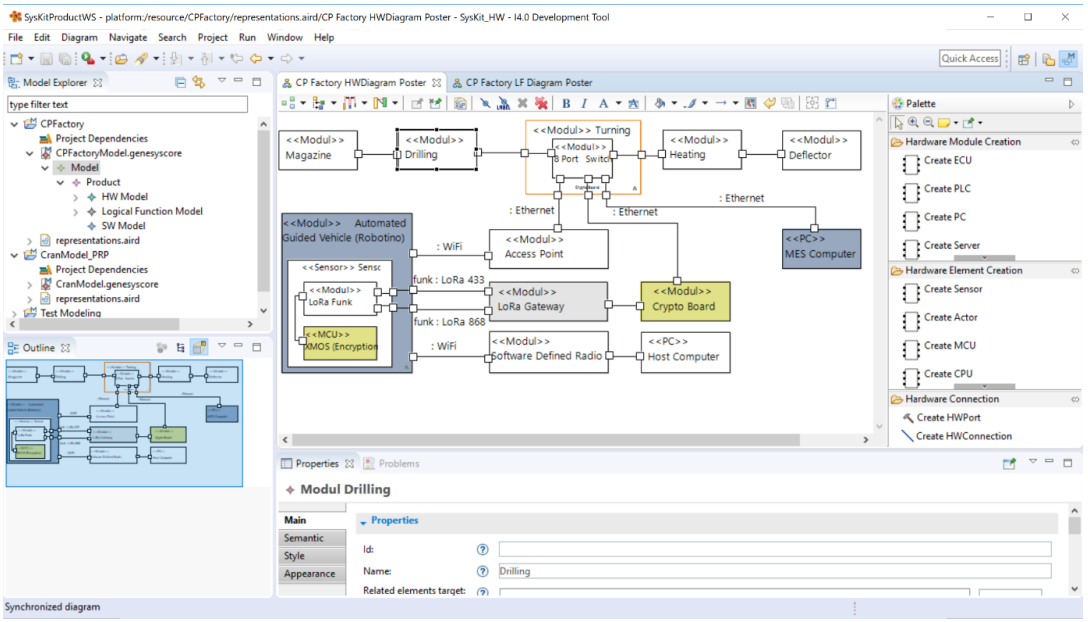



Abbildung 4.9: Modellierung einer Hardware Architektur in dem implementierten Modellierungswerkzeug [94]

Informationen in das Systemmodell implementiert. Die Werkzeuge zur Implementierung der Allokationsberechnung und des anschließenden Deployments werden in den Kapiteln 6.2 und 7.1 erläutert.

### 4.3.3 Schnittstellen

Für die Interaktion des Modellierungswerkzeugs mit dem Entwickler und anderen Werkzeugen wurden verschiedene Schnittstellen implementiert. Für die Modellierung wurde eine grafische Schnittstelle gewählt und mit dem Tool Sirius implementiert. In Abbildung 4.10 ist der Baumeditor und in Abbildung 4.11 ein grafischer Beispieleditor für die Modellierung der Hardwareschichten dargestellt.

Für das Traversieren der Systemmodelle wurde das Acceleo Framework verwendet. Acceleo führt eine Model2Text Transformation durch, was bedeutet, dass die Modelle durchlaufen und bestimmte Informationen oder Eigenschaften in Textform umgewandelt werden. Das wurde genutzt, um Allokationsinformationen aus dem Systemmodell zu extrahieren und sie in Metadaten oder direkt in Konfigurationsdateien zu transformieren. So kann der Übergang zwischen Dev und Ops realisiert werden. Dies wurde unter anderem vom Autor in [86] gezeigt.

Die Verknüpfung bzw. Rückführung von Informationen aus dem Betrieb in den Entwurf wurde am Beispiel der kontinuierlichen Kommunikation des Modellierungsframeworks mit einem Intrusion Detection System umgesetzt. Dies soll zeigen, wie Informationen aus dem

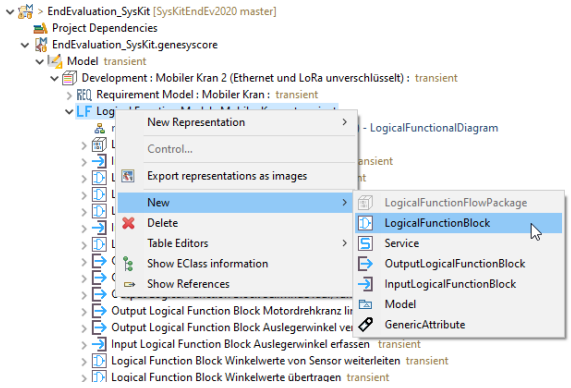



Abbildung 4.10: Modellierung einer logischen Funktionsarchitektur in einer hierarchischen Baumdarstellung [94]

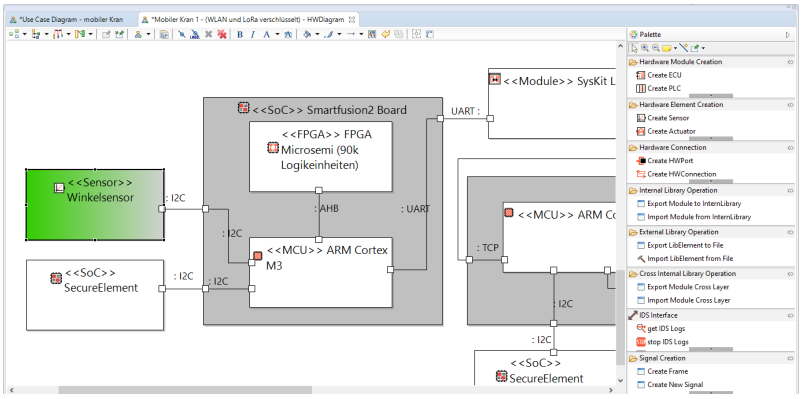



Abbildung 4.11: Modellierung einer Hardware Architektur in einem grafischen Diagramm [94]

Betrieb in die Systemmodelle integriert werden können und so eine Brücke von Ops zu Dev darstellen. Dieser Ansatz wurde vom Autor in [98] vorgestellt.

# Kapitel 5

# Komponentenspezifisches Policy Beschreibungskonzept für effiziente Allokation

In diesem Kapitel wird die Entwicklung des Policy Beschreibungskonzepts zur Modellierung der notwendigen Informationen für eine effiziente Allokation dargestellt. Es beinhaltet die Analyse und Formalisierung des Allokationsproblems, der Effizienz sowie den Entwurf eines Policy Metamodells.

Für die automatische Adaption der Edge Computing Infrastruktur ist die Entscheidung über die Task Allokation notwendig, wie im Kapitel 4.1 diskutiert. Edge Computing Systeme sind durch heterogene Recheneinheiten und begrenzte Rechenressourcen im Vergleich zur Cloud Umgebung gekennzeichnet. Im Edge Kontext sind Aspekte wie Energieeffizienz, Echtzeitfähigkeit, Zuverlässigkeit, Verfügbarkeit oder Ausfallsicherheit von zunehmender Bedeutung. Die Ausprägung dieser Aspekte hängt jedoch von der industriellen Anwendung und dem betrieblichen Kontext ab [2]. In diesem Zusammenhang wird die Bedeutung von Effizienz analysiert und für den Kontext der Task Allokation definiert.

Wie in Kapitel 4 erläutert, nimmt die Komplexität von Anforderungen und Systemen im Kontext des Edge Computing zu. Ein System kann aus zahlreichen Teilnehmern oder Recheneinheiten bestehen und jede Komponente im System stellt unterschiedliche Anforderungen und verfolgt unterschiedliche Effizienzziele. Beispielsweise steht bei der Objekterkennung in der Robotik die Minimierung der Latenzzeit im Vordergrund, während bei batteriebetriebenen fahrerlosen Transportsystemen ein niedriger Stromverbrauch gefordert ist. Das bedeutet, dass die Effizienz eines Edge Computing Systems anwendungsspezifisch und komponentenspezifisch definiert werden muss.

Die zunehmende Systemkomplexität und die unterschiedlichen Anforderungen, Randbedingungen und Effizienzziele der verschiedenen Teilnehmer führen dazu, dass eine hochqualitative und verlässliche Allokationsbestimmung unter Einhaltung der oben beschriebenen Informationen in einem Edge System nicht manuell handhabbar ist. Daher wurde in dieser Arbeit ein Beschreibungskonzept entwickelt, das die automatisierte Entscheidung über eine effiziente Allokation ermöglicht. Dadurch können die Qualität, Zuverlässigkeit und Dauer verbessert, sowie die Verfügbarkeit der Recheninfrastruktur erhöht werden. Besonderes Augenmerk wurde auf die Modularität und Skalierbarkeit des Konzeptes gelegt.

In diesem Kapitel wird zunächst die Definition des Allokationsproblems analysiert und die notwendigen Informationen werden identifiziert und in der Schichtenarchitektur verortet. Es folgt eine Analyse der Effizienz und der hierarchischen Anforderungen und Ziele der Systemkomponenten, die im Konzept dieser Arbeit als Policies beschrieben werden. Es wurde ein Policy Metamodell konzipiert, das die Beschreibung von Policies für Anwendungen und Recheneinheiten sowie ein Bibliothekskonzept für die Beschreibung und Wiederverwendung von Modulen beinhaltet. Basierend auf den modellierten Policies wurde die Extraktion von allokationsrelevanten Informationen in Form von Metadaten entworfen. Die Konzepte wurden als Erweiterung des Modellierungsframeworks aus dem Kapitel 4.3 implementiert und im Hinblick auf Machbarkeit, Modellierung von zwei industriellen Anwendungen und Skalierung evaluiert.

## 5.1 Analyse und Definition des Allokationsproblems im Zusammenhang mit dem Systemmodell

Für eine formalisierte Beschreibung ist es zunächst wichtig zu analysieren, was ein Allokationsproblem ist und welche Informationen zur Beschreibung und Lösung des Problems erforderlich sind. Im Allgemeinen wird das Allokationsproblem auch als Zuordnungs- oder Zuweisungsproblem bezeichnet (engl. assignment problem). In der Regel wird das Problem so beschrieben, dass eine Eins-zu-Eins Zuordnung zwischen $n$ Aufgaben und $m$ Agenten gefunden werden soll, mit dem Ziel, die Gesamtkosten der Zuordnungen zu minimieren. Klassische Beispiele sind die Zuordnung von Aufgaben zu Maschinen, von Aufgaben zu Arbeitern oder von Arbeitern zu Maschinen.

Im Kontext dieser Arbeit besteht das Problem, stark vereinfacht, darin, den Agenten Aufgaben zu bestimmten Kosten zuzuordnen. Die Anwendungen im Edge System stellen die Aufgaben mit unterschiedlichen Kosten oder Anforderungen an die Rechenressourcen dar, und die Recheneinheiten stellen die Agenten mit den verfügbaren Ressourcen dar. Das Konzept dieser Arbeit ist, dass die Allokationinformationen zusammen mit den Komponenten auf den verschiedenen Systemebenen einer Schichtenarchitektur beschrieben und in einen modellgetriebenen Prozess, wie im Kapitel 4.2 beschrieben, integriert werden können. Damit kann die Information dort definiert werden, wo sie relevant ist.

### 5.1.1 Definition des Allokationsproblem und allokationsrelevante Informationen

Das Allokationsproblem oder Zuordnungsproblem wurde bereits 1955 von Kuhn et al. [99] veröffentlicht. Pentico et al. [49] gibt eine Zusammenfassung der verschiedenen Definitionen, Varianten und Ansätze des Problems im letzten Jahrhundert.

Bei Zuordnungsproblemen geht es darum, die Elemente von zwei oder mehr Mengen optimal aufeinander abzustimmen, wobei sich die Dimension des Problems auf die Anzahl der abzustimmenden Mengen von Elementen bezieht. Wenn es zwei Mengen gibt, wie es in dieser Arbeit der Fall ist, können sie als *Aufgaben* und *Agenten* bezeichnet werden. In dieser Arbeit sind *Tasks* die zu erteilenden Aufgaben und *Recheneinheiten* sind die Agenten, welche die Aufgaben erledigen können.

Während das Zuordnungsproblem in seiner ursprünglichen Version darin bestand, jede Aufgabe einem anderen Agenten zuzuordnen, wobei jedem Agenten höchstens eine Aufgabe zugewiesen wurde (Eins-zu-Eins Zuordnung), werden in den Modellen dieser Arbeit mehrere Aufgaben demselben Agenten zugewiesen. Um das entwickelte Zuordnungsproblem näher zu erläutern, erklären wir zunächst das verallgemeinerte Zuordnungsproblem (engl. generalized assignment problem - GAP).

Das Modell geht davon aus, dass jede Aufgabe einem Agenten zugewiesen wird, lässt aber die Möglichkeit zu, dass einem Agenten mehr als eine Aufgabe zugewiesen wird, und berücksichtigt dabei, wie viel von der Kapazität eines Agenten für die Erledigung der Aufgaben jeweils in Anspruch genommen wird. Das mathematische Modell kann wie in (5.2) ausgedrückt werden, wobei $x_{ij} = 1$ ist, wenn der Agent $i$ der Aufgabe $j$ zugewiesen wird, 0, wenn das nicht der Fall ist. Dabei ist $c_{ij}$ die Kosten für die Zuweisung des Agenten $i$ zur Aufgabe $j$, $a_{ij}$ ist die verwendete Kapazität des Agenten $i$, wenn dieser Agent der Aufgabe $j$ zugewiesen wird, und $b_i$ ist die verfügbare Kapazität des Agenten $i$. Die erste Gruppe von Beschränkungen bewirkt, dass jede Aufgabe nur einem Agenten zugewiesen wird, und die zweite Gruppe von Beschränkungen bewirkt, dass die einem Agenten zugewiesene Aufgabengruppe seine Kapazität nicht überschreitet. Zu beachten ist, dass es in der zweiten Gruppe zwar $m$ Beschränkungen gibt, eine für jeden Agenten, jedoch nur eine *Ressource*, also die Kapazität des Agenten zur Erfüllung der Aufgaben, berücksichtigt wird.

$$\begin{aligned} \text{Minimieren} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{Unter Beachtung von:} \quad & \sum_{i=1}^{m} x_{ij} = 1 \quad j = 1,\ldots,n, \\ & \sum_{j=1}^{n} a_{ij}x_{ij} \leq b_i \quad i = 1,\ldots,m, \\ & x_{ij} = 0 \text{ oder } 1 \end{aligned} \tag{5.2}$$

Für die Lösung eines Allokationsproblems im Edge Computing Kontext sind jedoch einige **Anpassungen an das klassische Zuordnungsproblem** erforderlich, die im Folgenden aufgeführt werden.

**Mehrere Ressourcen:** Da Recheneinheiten über mehrere Ressourcen verfügen können, muss das verallgemeinerte Zuordnungsproblem nur erweitert werden, um zu erfassen, wie mehrere Ressourcen die Kapazitäten der Agenten begrenzen können. Dazu müssen zusätzliche Einschränkungen der zweiten Gruppe in (5.2) hinzugefügt werden, eine für jede Ressource.

**Beschränkungen:** Im Edge System können weitere Einschränkungen bestehen, die die Zuordnung von Agenten zu Tasks einschränken. Das erfolgt bspw. durch die Begrenzung der Anzahl der Tasks auf einen Agenten. Dies kann jedoch in der Regel als mehrere Ressourcen modelliert werden. Im Allgemeinen können weitere Beschränkungen hinzugefügt werden.

**Multikriterien:** Aufgrund der verschiedenen Effizienzziele, die in einem Edge System auftreten, muss das Allokationsproblem in der Lage sein, mehrere Kriterien zu berück-

sichtigen. Es gibt zwei grundlegende Methoden, mit denen mehrere Kriterien in Entscheidungsmodellen berücksichtigt werden können: (1) durch Kombination zu einem einzigen Kriterium und (2) durch getrennte und sequentielle Berücksichtigung der Kriterien. In dieser Arbeit werden die Kriterien getrennt betrachtet, obwohl es möglich ist, dass ein Kriterium mehrere Variablen umfasst, von denen wiederum angenommen werden kann, dass sie zu einem einzigen Kriterium kombiniert werden.

**Qualifikation der Agenten berücksichtigen:** Bei der Zuweisung von Aufgaben im Edge Kontext können Situationen entstehen, in denen Aufgaben explizit an Recheneinheiten zugewiesen oder ausgeschlossen werden können. Auch ist nicht jeder Agent für jede Aufgabe qualifiziert. Das kann durch das Einfügen einer Variable $q_{ij}$ ergänzt werden, wobei $q_{ij} = 1$ ist, wenn Agent $i$ für die Aufgabe $j$ qualifiziert ist, 0 wenn nicht.

**Kategorisierte Zuordnung:** In Edge Systemen gibt es Hierarchien und Gruppierungen sowohl auf der Aufgaben- als auch auf der Agentenseite, z.B. wenn die Aufgabe Teil einer übergeordneten Anwendung ist oder der Agent Teil eines Clusters oder Subsystems ist.

#### 5.1.1.1 Überblick über das Allokationsproblem im Edge Kontext

In dieser Arbeit wurde ein 2-dimensionales Zuordnungsproblem gewählt. Das bedeutet, dass das Problem darin besteht, die Elemente von zwei Gruppen - Aufgaben und Agenten - optimal zuzuordnen. Im Folgenden werden Recheneinheiten, Geräten und Agenten als Synonyme verwendet. Die Abbildung 5.1 zeigt die Struktur des Zuordnungsproblems, das in Aufgaben und Agenten unterteilt ist. Auf der linken Seite befinden sich die Aufgaben, die die Menge $T$ bilden, diese werden üblicherweise in Form von Anwendungen ($App$) gruppiert. Auf der rechten Seite befinden sich die Agenten bzw. Recheneinheiten, die die Menge $D$ bilden, diese können auch hierarchisch in Clustern ($CL$) gruppiert sein. Die Allokation hat die Aufgabe, eine Zuordnung zwischen den beiden Mengen zu finden.

Um dies als Problem zu beschreiben, wird die Entscheidungsmatrix $X$ aufgestellt, wie in der Mitte der Abbildung dargestellt. Die Matrix $X$ besteht aus den Entscheidungsvariablen $x_{ij}$, die die Zuordnung von Agent $i$ zu Aufgabe $j$ binär beschreibt. Nämlich 1, wenn zugewiesen, und 0, wenn nicht. Das Ergebnis der Entscheidungsmatrix bei der Lösung des Problems ist die Allokation.

Um über die Zuordnung entscheiden zu können, müssen die Kosten für die Zuordnungen definiert werden. Dazu wird die Matrix $C^K$ mit den Kosten $c_{ij}$ für die Zuteilung einer bestimmten Aufgabe $j$ an einen Agenten $i$ aufgestellt. In einem Edge System kann es verschiedene Arten von Kosten geben, die hier als Kostenart $K$ bezeichnet werden. Kostenarten können z.B. die Ausführungszeit oder der Energieverbrauch sein.

Eine besondere Art von Kosten sind die Ressourcen ${a_{ij}}^R$, die für die Ausführung einer Aufgabe $j$ in einem Agenten $i$ benötigt werden. Es kann auch verschiedene Arten von

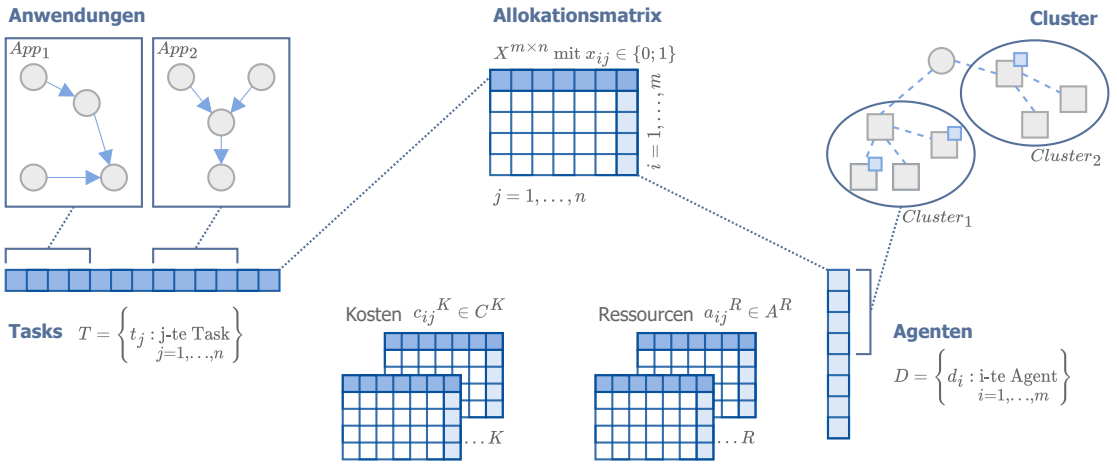



Abbildung 5.1: Überblick über das Allokationsproblem im Edge Kontext

Ressourcen $R$ geben, so dass es mehrere Ressourcenmatrizen $A^R$ geben kann. Auf der Seite der Recheneinheit ist die zugehörige Kapazität ${b_i}^R$ für die Ressourcenart $R$ definiert. Die Ressourcenbeschränkungen können also als die Summe über alle Aufgaben aus der Multiplikation der benötigten Ressource $a_{ij}$ mit der Allokationsvariablen $x_{ij}$ aufgestellt werden, und die Summe soll kleiner oder gleich der Kapazität oder Ressourcengrenze $b_i$ sein. Die Einschränkung kann für jeden Ressourcentyp bestimmt werden.

Eine weitere hilfreiche Information für die Zuordnung sind die Limitierungen, die verschiedene Komponenten im Edge System haben können. Hier werden die Task- und Gerätegrenzen für die Kostenarten $K$ berücksichtigt. Grenzen können auch auf Anwendungs- oder Clusterebene definiert werden. Dies ist z.B. sinnvoll, wenn die Gesamtlatenz einer Anwendung oder der Energieverbrauch einer Teilmenge von Recheneinheiten begrenzt werden soll. So können Constraints und Optimierungen definiert werden, die wie die Ressourcen Constraints aufgebaut sind, sich aber nicht auf Ressourcentypen, sondern auf Kostenarten beziehen.

#### 5.1.1.2 Formulierung des Allokationsproblems mit der allokationsrelevanten Information

In den Tabellen 5.1, 5.2 und 5.3 sind die zur Formulierung des Zuordnungsproblems verwendete Notation und die relevanten Zuordnungsinformationen aufgeführt. Im vorangegangenen Kapitel wurden die Eingangsinformationen für das Allokationsproblem erörtert. Das Allokationsproblem, das diese Eigenschaften des Systems nutzt, wird durch die Aufstellung von Constraints und Optimierungen formuliert. So können Lösungen für die Allokationsmatrix $X$ berechnet und verglichen werden.

Constraints für Kapazitäten, Aufgaben, Apps, Geräte und Cluster werden in dieser Arbeit berücksichtigt. Auch die Möglichkeit, eine spezifische Einschränkung zu beschreiben, wird

bei Bedarf einbezogen. Das geschieht durch die Aufstellung von Ungleichungen von Kosten zu Limits. In analoger Weise wurden Optimierungen für Tasks, Apps, Geräte und Cluster betrachtet, sowie die Möglichkeit, spezifische Optimierungen zu beschreiben. Das geschieht durch die Aufstellung von Minimierungs- oder Maximierungsfunktionen der Kosten einer Zuordnung. Insbesondere bei der Berücksichtigung verschiedener Kostenkriterien für die Optimierungen ist es sinnvoll, kombinierte Kostenausdrücke zu verwenden. Diese sind Linearkombinationen verschiedener Kostenarten und vereinfachen das Problem im Vergleich zu parallelen Optimierungsfunktionen.

Sowohl für Constraints als auch für Optimierungen gilt, dass sie nicht unbedingt für alle Komponenten des Edge Systems gelten müssen. Stattdessen ist eine komponentenspezifische Sammlung von Vorgaben, die als Constraints oder Optimierungen beschrieben werden, angedacht. Damit ist es möglich, die zunehmende Komplexität der unterschiedlichen Anforderungen von Aufgaben und Agenten im Edge Kontext so zu erfassen, dass eine automatische Zuordnung berechnet werden kann.

Tabelle 5.1: Notation von allokationsrelevanten Informationen für die Problemformulierung - Input Informationen

| **Input Informationen** | | |
|---|---|---|
| **Element** | **Notation** | **Erklärung** |
| Taskmenge | $T = \left\{ t_j : \text{j-te Task} \atop {}_{j=1,\dots,n} \right\}$ | Menge der $n$ zuzuordnenden Aufgaben oder Tasks $t_j$. |
| Anwendungsmenge | $APP = \{App_1, \dots : App_x \subseteq T\}$ | Teilmenge von $T$, die eine Gruppe von Aufgaben/Tasks darstellt. |
| Agentenmenge | $D = \left\{ d_i : \text{i-te Agent} \atop {}_{i=1,\dots,m} \right\}$ | Menge von $m$ Agenten oder Recheneinheiten $d_i$. |
| Clustermenge | $CL = \{Cluster_1, \dots : Cluster_y \subseteq D\}$ | Teilmenge von $D$, die eine Gruppe von Agenten darstellt. Dies kann zur Beschreibung von Subsystemen oder Clustern verwendet werden. |
| Entscheidungsmatrix | $X^{m \times n}$ mit $x_{ij} \in \{0; 1\}$ | Matrix der Größe $m \times n$, bestehend aus den Zuordnungsvariablen $x_{ij}$, die eine binäre Zuordnung von Agent $i$ $(d_i)$ zu Aufgabe $j$ $(t_j)$ definiert. Die Variablenmatrix wird verwendet, um Beschränkungen und Optimierungen aufzustellen, bei denen die Kosten nur im Falle einer Zuweisung $(x_{ij} = 1)$ summiert werden. Die Lösung entspricht der gesuchten Allokation. |
| Kosten | ${c_{ij}}^K \in C^K$ | Kosten der Art $K$ für die Zuordnung von Agent $i$ zu Aufgabe $j$. Diese bilden die Kostenmatrix $C^K$ mit einer beliebigen Anzahl von Kostenarten $K$. |
| Ressourcen | ${a_{ij}}^R \in A^R$ | Ressourcen der Art $R$, die zur Zuordnung von Agent $i$ zu Aufgabe $j$ verwendet werden. Diese bilden die Ressourcenmatrix $A^R$ mit einer beliebigen Anzahl von Arten $R$. |
| Kapazität | ${b_i}^R \in B^R$ | Kapazität der Art $R$, die die Recheneinheiten haben. Diese bilden den Kapazitätsvektor $B^R$. |
| Task / App Limit | ${l_j}^K$ / ${l_{App_x}}^K$ | Limit der Kostenart $K$ für eine Aufgabe $j$ oder für eine Teilmenge von Aufgaben, die in einer $App_x$ zusammengefasst sind. |
| Agenten / Cluster Limit | ${l_i}^K$ / ${l_{Cluster_y}}^K$ | Limit der Kostenart K für einen Agenten $i$ oder für eine Teilmenge von Agenten, die in einem $Cluster_x$ zusammengefasst sind. |

Tabelle 5.2: Notation von allokationsrelevanten Informationen für die Problemformulierung - Constraints

| **Constraints** | | |
|---|---|---|
| **Element** | **Notation** | **Erklärung** |
| Einmalige Aufgabenzuweisung | $\sum_{i=1}^{m} x_{ij} \quad j = 1, \ldots, n$ | Einschränkung, die sicherstellt, dass jede Aufgabe nur einem Bearbeiter zugewiesen wird. Das könnte explizit aufgabenspezifisch angepasst werden, z. B. wenn Aufgaben mehreren Agenten zugewiesen werden sollen. |
| Kapazität Constraint | $\sum_{j=1}^{n} {a_{ij}}^R \cdot x_{ij} \leq {b_i}^R$ | Die Summe der benötigten Ressourcen ${a_{ij}}^R$ für die Zuweisungen $x_{ij}$ sollte die Kapazität ${b_i}^R$ des Agenten $i$ über alle Aufgaben nicht überschreiten. |
| Task Constraint | ${c_{ij}}^K \cdot x_{ij} \leq {l_j}^K$ | Die Kosten des Typs $K$ für eine Zuteilung $x_{ij}$ sollen kleiner oder gleich dem festgelegten Limit $l_j$ der Aufgabe $j$ sein. |
| App Constraint | $\sum_{\forall t_j \in App_x} {c_{ij}}^K \cdot x_{ij} \leq {l_{app_x}}^K$ | Wie im Fall des Task Constraints, jedoch unter Hinzufügung der Summe aller Tasks $t_j$, die Elemente der Anwendung $x$ sind. Der Gesamtwert sollte kleiner oder gleich dem vorgesehenen Limit der Anwendung $l_{app_x}$ sein. |
| Gerät Constraint | $\sum_{j=1}^{n} {c_{ij}}^K \cdot x_{ij} \leq {l_i}^K$ | Die Summe der Kosten der Art $K$ für eine Zuweisung sollte kleiner oder gleich dem vorgesehenen Limit ${l_i}^K$ des Agenten $i$ sein |
| Cluster Constraint | $\sum_{\forall d_i \in Cluster_y} \sum_{j=1}^{n} {c_{ij}}^K \cdot x_{ij} \leq {l_{cluster_y}}^K$ | Wie im Fall des Gerät Constraints, jedoch unter Hinzufügung der Summe aller Agenten, die Elemente des Clusters $y$ sind. Der Gesamtwert sollte kleiner oder gleich dem vorgesehenen Limit des Cluster $l_{cluster_y}$ sein. |
| Allgemeines Constraint | $\sum \cdots \sum_{Typ} cost \cdot x_{ij} [\leq] limit$ | Im Allgemeinen kann eine Beschränkung als Multiplikation eines Kostenausdrucks mit der Allokationsvariablen $x_{ij}$ definiert werden. Die Summen über eine oder mehrere Mengen können dann als (Un-)Gleichungen zu einem Grenzwertausdruck mit verschiedenen Operatoren wie $(=, <, \leq, >, \geq)$ definiert werden. |

Tabelle 5.3: Notation von allokationsrelevanten Informationen für die Problemformulierung - Optimierungen

| **Optimierungen*** | | |
|---|---|---|
| **Element** | **Notation** | **Erklärung** |
| Task Optimierung | $\min_{\text{j-te Task}} \left( {c_{ij}}^{K} \cdot x_{ij} \right)$ | Die Kosten der Art $K$ für eine Zuordnung $x_{ij}$ sind für die ausgewählten Aufgaben $j$ zu minimieren. |
| App Optimierung | $\min_{\text{x-te App}} \left( \sum_{\forall t_j \in App_x} {c_{ij}}^{K} \cdot x_{ij} \right)$ | Wie bei der Task Optimierung, jedoch unter Hinzufügung der Summe aller Tasks $t_j$, die Elemente der ausgewählten Anwendung $x$ sind. |
| Gerät Optimierung | $\min_{\text{x-te App}} \left( \sum_{j=1}^{n} {c_{ij}}^{K} \cdot x_{ij} \right)$ | Die Summe der Kosten der Art $K$ für eine Zuordnung soll für den ausgewählten Agenten $i$ minimiert werden. |
| Cluster Optimierung | $\min_{\text{x-te App}} \left( \sum_{\substack{\forall d_i \\ \in Cluster_y}} \sum_{j=1}^{n} {c_{ij}}^{K} \cdot x_{ij} \right)$ | Wie bei der Geräteoptimierung, jedoch unter Hinzufügung der Summe aller Agenten, die Teil des Clusters $y$ sind. |
| Allgemeine Optimierung | $\min_{Teilmenge} (\sum \cdots \sum cost \cdot x_{ij})$ | Im Allgemeinen kann eine Optimierung als Multiplikation eines Kostenausdrucks mit der Allokationsvariablen $x_{ij}$ dargestellt werden, deren Summen über eine oder mehrere Mengen dann zu minimieren sind. |
| Kombinierte Kostenausdruck | $\alpha_1 \cdot {c_{ij}}^{K_1} + \alpha_2 \cdot {c_{ij}}^{K_2} + \cdots + \beta$ | Bei der Berücksichtigung mehrerer Kostenkriterien für Constraints und Optimierungen ist eine lineare Kombination verschiedener Kostenarten in einem Kostenausdruck möglich. |

**alle Optimierungen sind sowohl als Maximierungs- als auch als Minimierungsprobleme möglich*

### 5.1.2 Abbildung der allokationsrelevanten Informationen auf das Systemmodell

Um das Allokationsproblem auf der Basis der Systemmodelle aufzustellen und zu lösen, müssen die allokationsrelevanten Informationen aus der Tabelle 5.1 im oder zusammen mit dem Systemmodell beschrieben werden. Dazu muss zunächst die Schichtenarchitektur aus Kapitel 4.2.2 verfeinert werden. Abbildung 5.2 zeigt eine Schichtenarchitektur mit den relevanten Schichten und Modellen zur Beschreibung des Allokationsproblems im Edge Kontext.

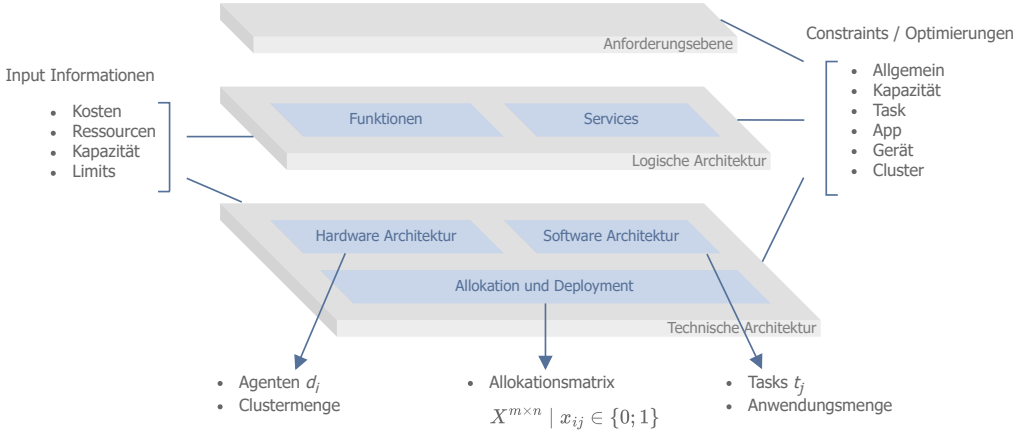



Abbildung 5.2: Allokationsrelevante Informationen im Systemmodell

Zur Detaillierung der Schichtenarchitektur für die Zuordnung im Edge Kontext wurden die Schichten auf der Grundlage des ARCADIA Modellierungskonzepts aus dem Eclipse Capella Tool[1] angeordnet. Dieser Modellierungsansatz wird von Unternehmen wie Thales aktiv genutzt. Er unterscheidet zwischen dem logischen und dem physischen Architekturentwurf. Bei der ersten geht es darum, *wie das System funktionieren soll, um die Anforderungen zu erfüllen*, während es bei der zweiten darum geht, *wie das System gebaut werden wird.*

In der vorliegenden Arbeit wird diese Unterscheidung verwendet, um die *logische Architektur* von der tatsächlichen *technischen Architektur* zu trennen. Limits, Constraints und Optimierungsziele können auf allen Ebenen definiert werden, da sie die Erwartungen an das System darstellen. Die *technische Architektur* ist hingegen von großer Bedeutung für die Zuordnung, da die erforderlichen Eingangsinformationen wie Kosten, Ressourcen und Kapazitäten von der tatsächlichen Umsetzung abhängen.

Die technische Architektur umfasst das Edge Computing Netzwerk, das eine spezielle Darstellung der Hardware Architektur ist, in der das Netzwerk von Recheneinheiten im Edge Computing System erfasst wird. Hier werden die notwendigen Informationen auf Seiten

[1] https://www.eclipse.org/capella/. Accessed 7. Mai. 2022

der Agenten $d_i$ und der Cluster beschrieben, wie zum Beispiel die Kapazität der Recheneinheiten. Eine Sonderform der SW Architektur stellen die Edge Computing Applications dar, in denen die Anwendungen (*App*) des Edge Computing Systems mit den zugehörigen Aufgaben $t_j$ beschrieben werden. Aus der Kombination von Recheneinheiten und Aufgaben lassen sich die Kosten und der Ressourcenbedarf beschreiben, da diese von beiden abhängig sind und daher in der technischen Architektur definiert werden müssen.

Zur expliziten Beschreibung des Allokationsproblems wurde die technische Architektur um die Ebene *Allokation und Deployment* erweitert. In dieser Ebene wird unter anderem die Zuordnungsmatrix $X$ definiert, die später die Zuordnung eines Agenten $d_i$ zu einer Aufgabe $t_j$ mit den Entscheidungsvariablen $x_{ij}$ löst, wie im Kapitel 5.1.1 erläutert.

Eine ausführliche Erklärung der notwendigen Informationen über welche Systemebene, welche Abhängigkeiten und welche Auswirkungen auf das Zuordnungsproblem findet sich in den Tabellen 5.4 und 5.5.

Tabelle 5.4: Abbildung der allokationsrelevanten Informationen auf die Systemebenen - Input Informationen

| **Input Informationen** | | | |
|---|---|---|---|
| **Element** | **Systemebene** | **Erklärung** | **Abhängigkeiten** |
| Task | Software Architektur | Teil der Edge Computing Anwendung | Gruppiert in *App* Verknüpft mit anderen Tasks |
| Anwendung | Software Architektur | Beschreibt einen gerichteten Graphen, der aus Tasks besteht | Gruppiert Tasks |
| Agent | Hardware Architektur | Beschreibt die Recheneinheiten als Teil des Edge Computing Netzes | Gruppiert in Clustern Über ein Netzwerk mit anderen Agenten verbunden |
| Cluster | Hardware Architektur | Beschreibt einen Graphen, der aus Recheneinheiten besteht | Gruppiert Agenten |
| Entscheidungsmatrix | Allokation und Deployment | Besteht aus Allokationsvariablen $x_{ij}$ | Verlinkung zu Agent $d_i$ und Task $t_j$. |
| Kosten | Allokation und Deployment | Die Kosten $c_{ij}$ für die Allokation, die bei der Recheneinheit oder bei der Aufgabe beschrieben werden können | Verlinkung zu Agent $d_i$ und Task $t_j$. |
| Ressourcen | Allokation und Deployment | Benötigte Ressourcen $a_{ij}$ für die Allokation, beschrieben in analoger Weise zu den Kosten. | Verlinkung zu Agent $d_i$ und Task $t_j$. |
| Kapazität | Hardware Architektur | Eigenschaft der Recheneinheiten | Verlinkung zu Agent $d_i$ |
| Task / App Limit | Requirements, Logische-, Software Architektur | Eigenschaft der Task / App | Verlinkung zu Task / App |
| Agenten / Cluster Limit | Requirements, Logische-, Hardware Architektur | Eigenschaft des Agenten / Clusters | Verlinkung zu Agent / Cluster |

Tabelle 5.5: Abbildung der allokationsrelevanten Informationen auf die Systemebenen - Constraints / Optimierungen

| Constraints / Optimierungen | | | |
|---|---|---|---|
| **Element** | **Systemebene** | **Erklärung** | **Abhängigkeiten** |
| Kapazität Constraint | Logische-, Hardware Architektur | Die implizite Einschränkung wird automatisch bei der Eingabe von Kapazitäts- und Ressourcenanforderungen erstellt. | Verlinkung zu Kapazitäten und Ressourcen |
| Task Constraint / Optimierung | Logische-, Software Architektur | Speziell für jede Aufgabe definiert | Verlinkung mit zugehörigen Kosten und Entscheidungsvariable |
| App Constraint / Optimierung | Logische-, Software Architektur | Speziell für jede Anwendung definiert | Verlinkung mit zugehörigen Kosten und Entscheidungsvariable |
| Gerät Constraint / Optimierung | Logische-, Hardware Architektur | Speziell für jedes Gerät definiert | Verlinkung mit zugehörigen Kosten und Entscheidungsvariable |
| Cluster Constraint / Optimierung | Logische-, Hardware Architektur | Speziell für jeden Cluster definiert | Verlinkung mit zugehörigen Kosten und Entscheidungsvariable |
| Allgemeines Constraint / Optimierung | Requirements, Logische-, Software-, Hardware Architektur | Kann auf jeder Ebene definiert werden | Verlinkung mit zugehörigen Kosten und Entscheidungsvariable |

## 5.2 Beschreibung der komponentenspezifischen Effizienz mittels Policies

Im vorangegangenen Kapitel wurde das Allokationsproblem im Edge Computing Kontext analysiert und formal beschrieben, um auf Basis der identifizierten Informationen automatisch eine Allokation zu berechnen. Die berechnete Allokation soll effizient sein, wobei sich die Frage stellt, was Effizienz in diesem Zusammenhang bedeutet und wie diese Information im Entwurf erfasst werden kann, um sie bei der Berechnung der Allokation zu berücksichtigen.

Effizienz hängt von verschiedenen Aspekten und Systemkomponenten ab. Aufgrund der Komplexität und Vielfalt der Anwendungen im Edge Computing Kontext ist eine pauschale Definition von Effizienz nicht zielführend. Daher werden in dieser Arbeit die Ziele und Bedürfnisse jeder Systemkomponente auf den verschiedenen Ebenen in Form von Policies betrachtet. Die Modellierung der allokationsrelevanten Policies der einzelnen Komponenten und die anschließende Einhaltung und Optimierung dieser Eigenschaften führt zu einer effizienten Allokation, die in dieser Arbeit als komponentenspezifische Effizienz bezeichnet wird.

In diesem Kapitel wird zunächst die Definition von Effizienz analysiert, um auf der Basis das entwickelte Konzept für eine hierarchische und komponentenspezifische Sicht auf Effizienz zu erklären. Anschließend wird das für die Policy Beschreibung entworfene Metamodell vorgestellt und die Aspekte Modularität, Verknüpfung mit dem Systemmodell, Anwendungsbereich, Extrahierbarkeit und Skalierbarkeit werden diskutiert.

### 5.2.1 Analyse von Effizienz im Kontext der Task Allokation

Effizienz ist ein Aspekt, der in verschiedenen Domänen und Fachgebieten auftritt. Bei der Entwicklung von Edge Systemen ist es keine Ausnahme, dass Effizienz eine Rolle spielt, jedoch hängt es vom System, Kontext und den Anforderungen ab, welche Art von Effizienz jeweils betrachtet wird. Im Falle der Aufgabenverteilung findet man im Laufe der Jahrzehnte etwa wissenschaftliche Arbeiten, die sich mit der Energie, Latenz, Kosten, etc. als Effizienz auseinandersetzen und diese optimieren wollen. Diese Ansätze betrachten meist ein oder zwei Effizienzaspekte und schlagen eine Optimierungslösung vor. Das gilt jedoch in der Regel nur für den Anwendungsbereich und die gewählten Aspekte. In dieser Arbeit wurde der Effizienzgedanke so abstrahiert, dass die Effizienzaspekte in Abhängigkeit von den Anwendungskomponenten modular beschrieben werden und nicht von einer festen Optimierung abhängen. So können spezifische Effizienzaspekte für verschiedene Edge Systeme so berücksichtigt werden, dass eine anwendungsspezifische effiziente Allokation für das System berechnet werden kann.

**Definition von Effizienz (engl. efficiency):** Im Allgemeinen kann Effizienz als die Fähigkeit beschrieben werden, gewünschte Ergebnisse zu erzielen, ohne Material, Zeit oder Energie zu verschwenden. Obwohl das Wort sowohl auf Menschen als auch auf Gegenstände angewendet werden kann, wird es weitaus häufiger auf Gegenstände wie Maschinen, Systeme, Prozesse und Organisationen bezogen[2]. Im Kontext der Aufgabenzuweisung in einem Edge Computing System kann Effizienz als die Verringerung unnötiger Ressourcen verstanden werden, die zur Erzeugung eines bestimmten Ergebnisses eingesetzt werden, einschließlich beispielsweise Zeit und Energie. In diesem Zusammenhang ist Effizienz kein Selbstzweck oder ein eigenständiges Ziel an sich. Vielmehr trägt Effizienz dazu bei, mehr der gewünschten Ergebnisse zu erzielen. Denn Effizienz sollte immer zusammen mit Effektivität betrachtet werden. Während Effizienz bedeutet, die gewünschten Ergebnisse optimal zu erzielen, bedeutet Effektivität, die gewünschten Ergebnisse zu erreichen.

**Definition von Effektivität (engl. efficacy):** Im Kontext der Zuweisung kann Effektivität als die Fähigkeit definiert werden, eine durchführbare Zuweisung zu finden und die Effizienz, dass diese Zuweisung die Verschwendung von Material, Energie, Aufwand, Geld und Zeit vermeidet. Bei der Analyse verschiedener industrieller Anwendungen wurde festgestellt, dass in den meisten Fällen eine durchführbare Zuweisung, die eine angemessene Ressourcenverwendung mit sich bringt, wichtiger ist als die ultimative effiziente Zuweisung, die die Ressourcenverschwendung auf ein globales Minimum reduziert. Es wird deutlich, wenn man nicht nur die Effizienz der Zuweisung, sondern auch die Effizienz beim Ermitteln der Zuweisung analysiert.

Aus dem Grund liegt in dieser Arbeit der Schwerpunkt bei der Berechnung einer effizienten Zuweisung zunächst auf der Suche nach realisierbaren Zuweisungen mit vertretbarem Ressourcenverbrauch und dann auf der Optimierung der Zuweisung mit vertretbarem Overhead. Das erfordert jedoch die Definition des vertretbaren Ressourcenverbrauchs für die Lösung des Allokationsproblems, und dies wird durch die Einschränkungen bei der Formulierung des Allokationsproblems realisiert. Wichtig ist, dass die Einschränkungen für jede Systemkomponente definiert werden können, so dass die vertretbare Ressourcenverwendung für jede Komponente immer bei der Allokation berücksichtigt werden kann. Das gilt analog für die Optimierungsziele, die ebenfalls wie Nebenbedingungen für jede Komponente definiert werden können.

**Definition der komponentenspezifischen Effizienz:** Ausgehend von der obigen Analyse können wir für den Kontext der Task Allokation in dieser Arbeit komponentenspezifische Effizienz als die Fähigkeit definieren, zuerst realisierbare Allokationen zu finden, die einen angemessenen Ressourcenverbrauch erfüllen, und diese dann zu optimieren. Bei Edge Systemen ist es in der Regel viel effizienter, schnell eine realisierbare Zuweisung zu finden,

[2] "Efficient." Merriam-Webster.com Dictionary, Merriam-Webster, https://www.merriam-webster.com/dictionary/efficient. Accessed 2 Apr. 2022.

als lange Zeit mit der Berechnung optimalerer Zuweisungen zu verbringen. So kann z.B. die Verfügbarkeit des Systems erhöht oder die Ausfallzeit verringert werden, wenn eine Zuweisung möglichst schnell gefunden wird.

Zentraler Bestandteil der komponentenspezifischen Effizienz ist die Möglichkeit, den vertretbaren Ressourcenverbrauch und ggf. die Optimierungsziele für jede Komponente zu definieren. Das wird hier als Policy bezeichnet und basiert auf den in der Formulierung des Allokationsproblems definierten Constraints und Optimierungen.

### 5.2.2 Hierarchische und komponentenspezifische Bedarfe als Policy

Im vorherigen Kapitel wurden Effizienz und Effektivität im Zusammenhang mit der Aufgabenzuweisung diskutiert. Um die ermittelte komponentenspezifische Effizienz bei der Formulierung und Lösung des Allokationsproblems berücksichtigen zu können, müssen die Anforderungen und Bedürfnisse der Tasks sowie die Kapazitäten der Agenten beschrieben werden. Wie diese Informationen formal mathematisch definiert werden können, wurde im Kapitel 5.1.1 gezeigt.

Um eine effiziente Allokation zu erreichen, können für jede Komponente Randbedingungen und Optimierungsziele definiert werden. Die Randbedingungen und Optimierungsziele können als Regeln für die Zuweisung verstanden werden. Die ersten sind Regeln, die befolgt werden müssen, damit eine Zuweisung gültig ist, die zweiten sind fakultative Regeln, die zur Verbesserung der Zuweisung verwendet werden können. Das bedeutet, dass Regeln für die Zuweisung von Aufgaben an Agenten für ein bestimmtes Edge System definiert werden können, und diese Regeln können auf allen Ebenen und Komponenten des Systems definiert werden.

Zur Formalisierung dieser Regeln bei der Modellierung des Systems wurden sie als *Policies* bezeichnet, da es ein System von Leitlinien zur Steuerung von Entscheidungen und zur Erzielung rationaler Ergebnisse impliziert. Im Zusammenhang mit der Aufgabenzuweisung stellen die Policies also ein System dar, das die Zuweisungsentscheidung auch während des Betriebs des Systems steuert und zu realisierbaren Ergebnissen führt.

**Definition von Policy:** Im Kontext der Aufgabenzuweisung ist eine Policy in dieser Arbeit eine Form zur Beschreibung der Anforderungen, Einschränkungen und Optimierungsziele einer bestimmten Anwendung. Sie leitet und verifiziert Entscheidungen wie die Ressourcenzuweisung, um ein realisierbares und designkonformes Edge Computing System zu erreichen. Dies wurde vom Autor in [87] vorgestellt.

**Modularität und Hierarchie der Policies:** Die Policies als ein System von Richtlinien zur Entscheidung über die Allokation bestehen aus Policy Elementen, nämlich den Constraint Policies und Optimization Policies, die modular für jede Komponente definiert

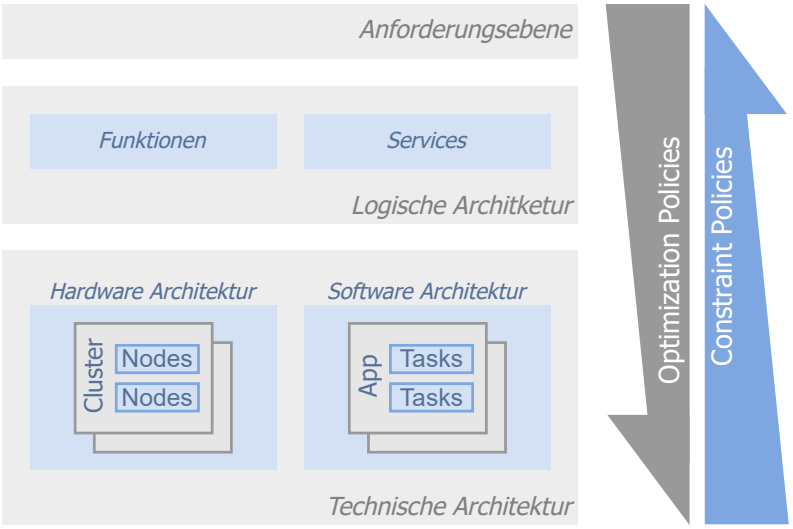



Abbildung 5.3: Vorgehen bei der Betrachtung der Policy Hierarchien - Bottom-Up (constraintPolicies) und Top-Down (optimizationPolicies)

werden können. Bei der Aggregation der Policies zur Lösung einer Allokation wird die Hierarchie der Komponenten im Systemmodell berücksichtigt. Constraint Policies werden in einem Bottom-Up Prozess und Optimization Policies in einem Top-Down Prozess geprüft, wie in Abbildung 5.3 dargestellt.

Die Unterscheidung dient einer effizienten und skalierbaren multikriteriellen Lösung des Allokationsproblems und stellt in erster Linie sicher, dass alle Constraint Policies auf der untersten Ebene erfüllt werden. Dies ist, wie bereits erläutert, der Schwerpunkt der komponentenspezifischen Effizienz in dieser Arbeit. Im weiteren Verlauf versuchen die Optimization Policies vom Gesamtsystem aus nach unten, die Allokation zu verbessern. Dabei wird im Zweifel in Kauf genommen, dass die Allokation auf der unteren Ebene einzelner Komponenten möglicherweise nicht optimal ist, jedoch steht die Betrachtung des Gesamtsystems im Vordergrund. Das Hierarchiekonzept bei der Betrachtung der Optimierungsmaßnahmen kann auch durch die Einführung einer Priorisierung der Maßnahmen ergänzt werden, z.B. Prioritäten (Niedrig, Normal, Hoch), wobei standardmäßig die Prioritätsstufe Normal verwendet wird.

Im Kapitel 6.2 wird die detaillierte Vorgehensweise bei verschiedenen Optimization Policies diskutiert, die ein multikriterielles Problem darstellen. Für die Lösung wird eine Kombination aus gewichteten Summen und lexikographischer Optimierung verwendet.

## 5.3 Entwurf des Metamodells für das Policy Beschreibungskonzept

Um die in den vorangegangenen Kapiteln entwickelten Konzepte in einem modellgetriebenen Entwicklungsprozess wie im Kapitel 4.2.3 nutzen zu können, wurde in dieser Arbeit ein Metamodell entwickelt. Mit Hilfe des Policy Metamodells kann die Definition und Extraktion der allokationsrelevanten Informationen so vorgenommen werden, dass eine automatisierte komponentenspezifische Allokation berechnet werden kann.

Das Metamodell muss verschiedene Aspekte der oben beschriebenen Konzepte, wie Modularität der Policies, Verknüpfung mit den Systemmodellkomponenten, Skalierbarkeit sowie die eindeutige Zuordnung und Identifizierbarkeit der allokationsrelevanten Informationen in Bezug auf das Allokationsproblem beinhalten, um die Informationen extrahieren zu können.

In diesem Kapitel wird zunächst ein Überblick über das Metamodell gegeben, um darauf aufbauend auf die verschiedenen Aspekte einzugehen. Verknüpft mit dem Systemmodell wird die detaillierte Modellierung der Anwendung und des Edge Computing Netzwerks als Graph gezeigt. Anschließend wird der Modulbibliotheksansatz erläutert, mit dem die Skalierung der Beschreibung adressiert wird, sowie das Konzept zur Identifikation und Extraktion der allokationsrelevanten Informationen in Form von Metadaten.

### 5.3.1 Überblick des Policy Metamodells

Eine vereinfachte Version des Metamodells mit den policy-relevanten Elementen ist in Abbildung 5.4 zu sehen, wie vom Autor in [87] präsentiert. In Anlehnung an die Definition des Allokationsproblems im Kapitel 5.1.1 werden zwei Arten von Policies betrachtet: *PolicyConstraint* und *PolicyOptimization*. Die Policies können für ein *PolicyBasedElement* beschrieben werden, das auch *PolicyLimit* haben kann. Durch die Erweiterung von *PolicyBasedElement* aus einem *Node*, *Task* und allgemein Systemkomponenten, werden die Policies mit dem Systemmodell verknüpft.

Ein Tupel aus *Node* und *Task* enthält eine Referenz auf die Zuweisungsvariablen, die festlegen, ob die Aufgabe für dieses Paar zugewiesen ist. Alle Zuweisungsvariablen werden in einer Zuweisungsmatrix gespeichert, die auch die Liste der Tasks und Nodes für die Zuweisung enthält, die den Umfang des Zuweisungsproblems definiert.

Darüber hinaus müssen die Kosten für die Zuweisungen definiert werden. Die Kosten werden den *PolicyCost* entnommen, die die Kosten für die Zuordnung eines Paares aus *Task* und *Node* definieren. Die Kosten können von einer bestimmten Kostenart sein und alle Kosten desselben Typs werden in einer *PolicyCostMatrix* für die entsprechende Kostenart gespeichert.

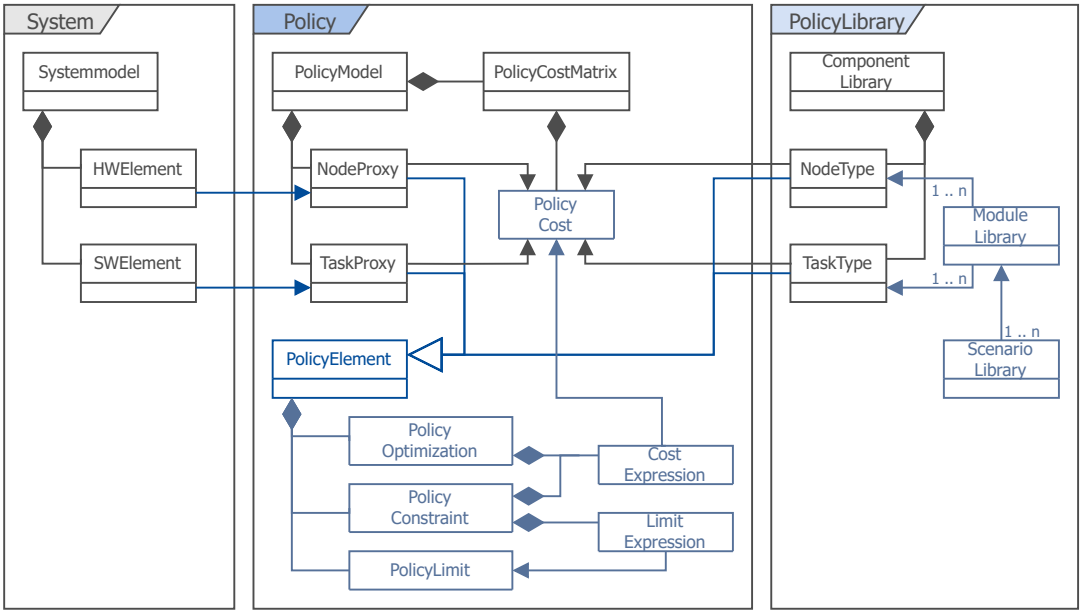



Abbildung 5.4: Überblick über das Metamodell für die Policy Beschreibung

Obwohl im Kapitel 5.1.1 die Definition von Kosten und Ressourcen bzw. Kapazität und Grenzen getrennt betrachtet wurde, gibt es keinen Unterschied in der Modellierung dieser Aspekte. Daher werden im Metamodell nur Kosten und Limits berücksichtigt. Mit *PolicyCost* können auch Ressourcen modelliert werden und die entsprechenden Kapazitäten können als *PolicyLimit* von Nodes modelliert werden.

Bei der Modellierung von *PolicyConstraints* kann ein *ConstraintType* gewählt werden, um die Art der Ungleichheit zu definieren, standardmäßig wird *LessOrEqualThan* gewählt. Das Constraint hat einen *costExpression*, der eine lineare Kombination von Kosten ist. Dazu verweist der *costExpression* auf die *PolicyCostMatrix* des entsprechenden Kostentyps. Für die andere Seite der Ungleichheit hat die Bedingung einen *limitExpression*, der eine Linearkombination von PolicyLimits ist. Bei der Modellierung der *PolicyOptimization* wird nur ein *costExpression* verwendet und die Art der Optimierung gewählt, standardmäßig wird die Minimierung verwendet.

Die hierarchische Modellierung von Policies ergibt sich intrinsisch aus der hierarchischen Struktur des Systemmodells. Deshalb werden Cluster oder Anwendungen nicht direkt, sondern indirekt über die Systemkomponente verknüpft.

Das Metamodell ermöglicht die Modellierung allokationsrelevanter Informationen und eine Verknüpfung mit dem Systemmodell. So kann beispielsweise eine Anwendung zur Personenkontrolle auf einer Baustelle aus Tasks wie Bilderfassung, Streaming, Personenerkennung und Alarmbenachrichtigung bestehen. Die Tasks können auf verschiedenen Agenten mit unterschiedlichen Kosten ausgeführt werden. Darüber hinaus können anwendungsbezogene Policies beschrieben werden, z.B. wenn die Gesamtkosten einer Anwendung begrenzt

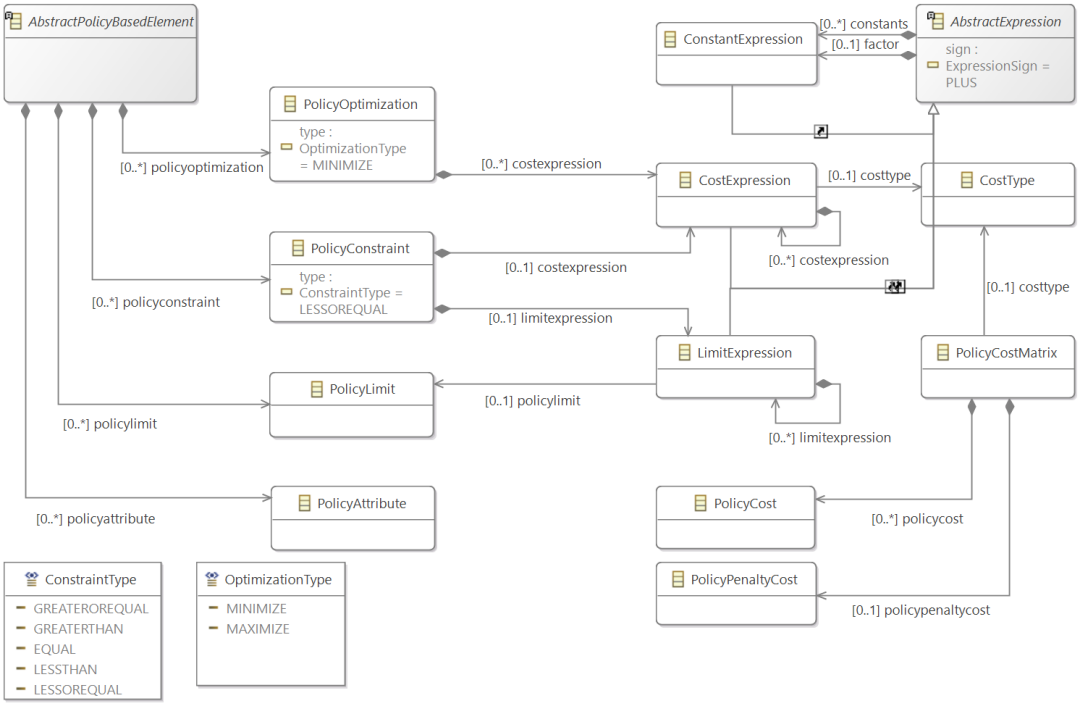



Abbildung 5.5: Auszug des Metamodell für die Policy Modellierung

oder optimiert werden sollen. Das kann durch die Beschreibung eines *PolicyConstraints* oder *PolicyOptimization* unter einer Applikation erfolgen. So werden die benötigten Informationen wie Kosten, Kapazitäten, Constraints und Ziele beschrieben, um eine automatische Berechnung des Zuordnungsproblems zu ermöglichen. Der Ansatz ist strukturiert und hierarchisch durch die integrierte Beschreibung von Policies auf den Ebenen Task, ECNode und Application. Dies lässt sich leicht auf Teilsysteme und Gesamtsysteme ausweiten.

### 5.3.2 Policy Modellierung und Verbindung mit dem Systemmodell

#### 5.3.2.1 Policy Modellierung

Die Modellierung von Policies dient der Beschreibung von Anforderungen oder Bedarfen. Um diese Informationen für eine automatische Berechnung der Allokation nutzen zu können, müssen sie formalisiert beschrieben werden. Die Beschreibung der Policies basiert auf der Definition von mathematischen Ausdrücken für die Kosten und Grenzwerte, daher wurde in dieser Arbeit ein System zur Modellierung dieser Ausdrücke im Metamodell entwickelt. Eine detaillierte Darstellung der Artefakte für die Policy Beschreibung und der mathematischen Ausdrücke ist in Abbildung 5.5 zu sehen.

**PolicyConstraint:** Ein Constraint besteht aus einem *CostExpression*, einem *LimitExpression* und dem Typ des Constraints, der aus der Enumeration *ConstraintType*

ausgewählt werden kann. Auf diese Weise kann eine Bedingung definiert und später in eine Ungleichung wie in (5.3) umgewandelt werden.

$$\sum_{iteration} \cdots \sum costExpression \cdot x_{ij} \underset{constraintType}{\left[\leq\right]} limitExpression \tag{5.3}$$

**PolicyOptimierung:** Eine PolicyOptimierung besteht aus einem *CostExpression* und der Art der Optimierung, die aus der Enumeration *OptimizationType* ausgewählt werden kann. Auf diese Weise kann eine Optimierung definiert und später in die mathematische Form wie in (5.4) umgewandelt werden.

$$\underset{optimizationType}{\left[\min\right]} \left\{ \sum_{iteration} \cdots \sum costExpression \cdot x_{ij} \right\} \tag{5.4}$$

**AbtractExpression:** Für einen allgemeinen mathematischen Ausdruck einer Linearkombination wurden die folgenden Komponenten modelliert (siehe 5.5). Ein Vorzeichen (*sign*), ein konstanter Vorfaktor ($factor$), der Inhalt oder die Variable(n) und der/die konstante(n) Offsetfaktor(en) (*constants*). Durch ein rekursives Containment kann ein Ausdruck dann selbst einen Ausdruck enthalten, wodurch komplexe Ausdrücke modelliert werden können. Bei der Kombination der Komponenten des Ausdrucks wird immer eine Addition verwendet, da jede Komponente selbst ein Vorzeichen hat.

$$\left[ \underset{sign}{\pm} \underset{factor}{\alpha} \cdot (x + \ldots) + \underset{constant}{\beta} \right] + \left[ \pm \alpha \cdot (\ldots) + \beta \right] + \ \ldots \tag{5.5}$$

**ConstantExpression:** Für die Beschreibung eines konstanten Wertes wird die Hilfsklasse *DoubleUnitValue* eingeführt, die für die Modellierung von Zahlenwerten verwendet wird.

**DoubleUnitValue:** Mit dieser Hilfsklasse kann ein Wert als Double modelliert werden. Zusätzlich kann eine Textbeschreibung und die 10-er Ordnung des Wertes hinzugefügt werden. So können z.B. Werte als milli, micro, nano, etc. modelliert werden. Alle Klassen, die einen Wert beschreiben, erben von dieser Hilfsklasse.

**CostExpression:** Der *CostExpression* erbt von *AbstractExpression* und hat ein rekursives Containment, mit dem ein *CostExpression* weitere CostExpressions enthalten kann. Wichtiger Bestandteil sind die Kosten. Hierfür hat eine *CostExpression* einen Verweis auf die *PolicyCostMatrix*. So kann beschrieben werden, dass die Latenzkosten Teil dieser *CostExpression* sind.

**LimitExpression:** Die *LimitExpression* funktioniert analog zur *CostExpression*, wobei sie ein rekursives Containment zur *LimitExpression* hat und als Inhalt bzw. Variable einen Verweis auf das *PolicyLimit* statt auf die *PolicyCostMatrix* hat.

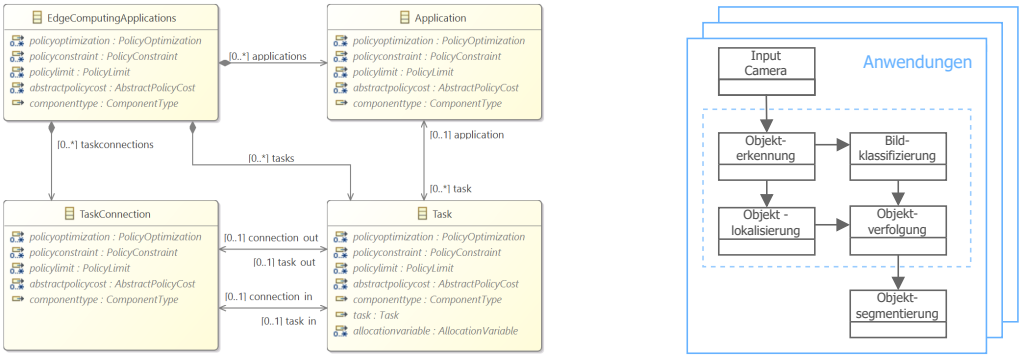



Abbildung 5.6: Auszug des Software Metamodells (links) und beispielhafter Anwendungsgraph (rechts)

#### 5.3.2.2 Systemmodellierung - Edge Computing Anwendung

Das Allokationsproblem befasst sich mit der Zuordnung einer Menge von Agenten oder Recheneinheiten zu einer Menge von Aufgaben. Die Aufgabenmenge wird im Systemmodell als Teil des Softwaremodells modelliert. Die Anwendungen werden jeweils als gerichteter Graph $G_{app} = (Task, TaskConnection)$ dargestellt, wobei die Aufgaben die Knoten und die TaskConnections die Verbindungen zwischen den Aufgaben sind. Auf diese Weise lässt sich die Kommunikation von Tasks sowie die Gruppierung von Tasks zu Anwendungen modellieren. Im Edge System kann es verschiedene Anwendungen geben. Abbildung 5.6 zeigt einen Auszug aus dem Metamodell und einen beispielhaften Anwendungsgraph.

#### 5.3.2.3 Systemmodellierung - Edge Computing Netzwerk

Die Menge der Recheneinheiten wird im Systemmodell als Teil des Hardwaremodells modelliert. Das Edge Computing Netzwerk wird als Graph $G_{device} = (EdgeNode, NetworkConnection)$ modelliert, wobei die Recheneinheiten (EdgeNode) die Knoten und die NetworkConnections die Kanten des Netzwerks sind. Letztere können verwendet werden, um die Kommunikationseigenschaften zwischen Nodes zu beschreiben. Zusätzlich können EdgeNodes zu EdgeClustern gruppiert werden. Abbildung 5.7 zeigt einen Auszug aus dem Metamodell sowie ein beispielhaftes Edge Computing Netzwerk.

#### 5.3.2.4 Verbindung von Policy Modellierung und Systemmodellierung

Für die Verbindung der Metamodelle zur Systemmodellierung und Policy Modellierung bieten sich zwei gegensätzliche Varianten an, wie dies realisiert werden kann.

Einerseits kann das Policy Metamodell direkt in das System Metamodell integriert werden. Ein Vorteil ist, dass die Informationen wie Kosten, Limits, Constraints und Optimierungen direkt in der Systemkomponente definiert werden können. Ein Nachteil ist, dass das

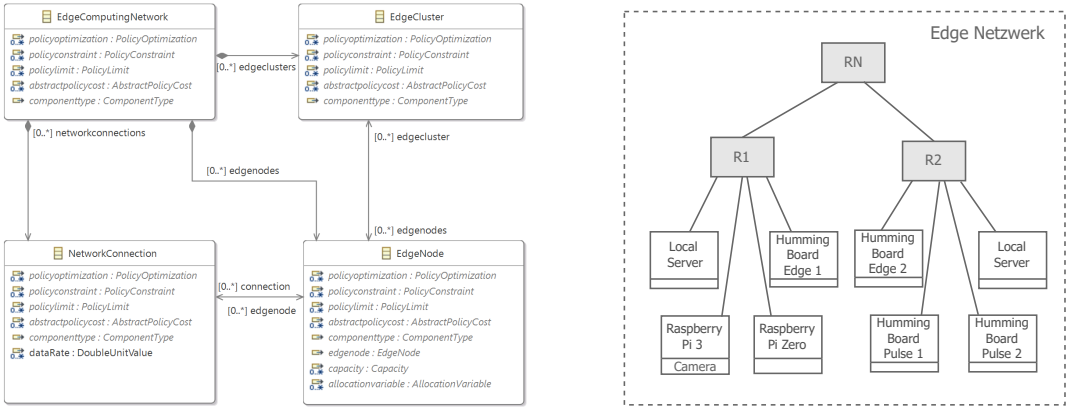



Abbildung 5.7: Auszug des Hardware Metamodells (links) und beispielhaftes Edge Netzwerk (rechts)

Systemmetamodell von der Policy Modellierung abhängig wäre, was die Übertragung der Policy Modellierung auf ein anderes Systemmodell erschwert und die anschließende Extraktion allokationsrelevanter Informationen komplex macht.

Andererseits kann das Policy Metamodell lose mit dem System Metamodell gekoppelt werden. Dazu wird das Adapter Design Pattern angewendet, um beide Metamodelle zu verbinden. Ein Vorteil ist, dass die Informationen von Systemmodell und Policy Modell in getrennten Modellen gespeichert werden können, was die spätere Extraktion der allokationsrelevanten Informationen vereinfacht. Außerdem ist eine Zuordnung zwischen Policies und Systemkomponenten stets nachvollziehbar. Der Nachteil ist, dass die Informationen aus Sicht des Metamodells nicht direkt bei der Systemkomponente definiert sind, jedoch kann es bei der späteren Umsetzung durch ModelEditoren umgangen werden, so dass dies aus Sicht der Entwickler dennoch möglich ist.

Aus den Gründen wurde in dieser Arbeit die Verbindung der Metamodelle durch Repräsentanten oder Wrapper der Systemkomponenten im Policy Modell gelöst. Im Policy Metamodell gibt es die Klassen *NodeProxy*, *EdgeProxy* und *SystemElementProxy*. Diese sind die Repräsentanten der jeweiligen Komponenten im Systemmodell. Damit können die Proxy Klassen alle notwendigen Verweise auf die anderen Klassen im Policy Metamodell bilden. Die Proxy Klassen erben auch von der abstrakten Klasse *PolicyBasedElement*, was bedeutet, dass *PolicyConstraint* und *PolicyOptimization* für sie definiert werden können.

Die eigentliche Verbindung wird dadurch hergestellt, dass die Komponenten im System Metamodell *EdgeNode*, *Task* und *SystemElement* jeweils durch die zugehörigen Proxy Klassen im Policy Metamodell referenziert werden. Dadurch können die Tasks und EdgeNodes von der Systemkomponente in der Allokationsmatrix ausgewählt werden, die für den Umfang des Allokationsproblems in Frage kommen.

### 5.3.3 Modulare Policy Modellierung und Wiederverwendungskonzepte

#### 5.3.3.1 Modulare Policy Modellierung

Durch die Klasse *PolicyBasedElement* werden die Fähigkeiten definiert, die jede Systemkomponente zur Beschreibung von Policies benötigt. Durch die Verknüpfung des Policy Modells mit dem Systemmodell verfügt jede Systemkomponente über einen Proxy als *PolicyBasedElement*, so dass die Bedarfe und die Effizienz modular und komponentenspezifisch modelliert werden können, entsprechend dem im Kapitel 5.2 beschriebenen Konzept.

Ein Vorteil davon ist, dass jede einzelne Policy entsprechend den Anforderungen der Komponenten definiert werden kann. Dies spielt vor allem in Edge Systemen, in denen sich Teilsysteme nicht oft wiederholen und spezifische Anforderungen haben, eine große Rolle. So kann die komponentenspezifische Einhaltung von z.B. Constraints später bei der Allokation sichergestellt werden.

Ein Nachteil der expliziten Modellierung der Policy Informationen für jede einzelne Komponente ist, dass die Anzahl der Policies abhängig von der Anzahl der Komponenten wächst. Insbesondere in Edge Systemen, in denen sich Teilsysteme mehrfach wiederholen, würde sich auch die Modellierung der Policies wiederholen, was einen erhöhten Modellierungsaufwand ohne zusätzlichen Informationsgewinn bedeuten würde.

Um eine effiziente und skalierbare Modellierung und anschließende Allokationsberechnung zu ermöglichen, wurden beide Fälle betrachtet. Zu diesem Zweck wurden für jeden Fall Wiederverwendungskonzepte untersucht und umgesetzt.

#### 5.3.3.2 Wiederverwendungskonzepte für die Policy Modellierung

##### 5.3.3.2.1 *Systemmodul Bibliotheksansatz*

Ziel der Wiederverwendung ist es, den Modellierungsaufwand für das System und die Policies zu reduzieren, indem die Modellierung von wiederkehrenden Informationen vermieden wird. Beim Bibliotheksansatz geht es um die Speicherung und Wiederverwendung wiederkehrender Bausteine als Modul.

Der zentrale Baustein des Bibliotheksansatzes ist die Einführung von *SystemModulen*. Unter einem Modul können Subsysteme im Edge Kontext kombiniert und wiederverwendet werden. Wichtig ist dabei die Möglichkeit einer ebenenübergreifenden Definition von Modulen, d.h. ein Modul kann z.B. Elemente aus dem Hardware Modell und dem Software Modell enthalten.

Systemmodule werden in der Systembibliothek gespeichert und es können bereits Policies für die Komponenten im Systemmodul definiert werden. Wenn Systemmodule instanziiert werden, werden sie dem Systemmodell hinzugefügt, einschließlich der zuvor definierten Policies. Somit ist die Wiederverwendung der Informationen vorgesehen.

#### 5.3.3.2.2 *Komponententyp Bibliotheksansatz*

Die Wiederverwendung von Policy Kosten kann nicht auf dieselbe Weise erfolgen, da sie von der Kombination aus Task und Recheneinheit abhängt, d.h. die Kosten eines Systemmoduls, das nur Tasks enthält, können nicht modelliert und wiederverwendet werden.

Um eine wiederverwendbare Modellierung der Zuweisungskosten von Task- und Recheneinheiten zu ermöglichen, wird der *componentType* eingeführt. Damit kann ein $EdgeNode$ einen $EdgeNodeType$ haben. Die Kosten einer Zuordnung können dann zwischen einer $Task$ und einem $EdgeNodeType$ definiert werden. So kann es z.B. drei EdgeNodes desselben Typs geben und die Kosten müssen nur ein- statt dreimal definiert werden.

Analog dazu kann es auf der Seite der Tasks $TaskTypes$ geben, wobei die Kosten generell zwischen $EdgeNodeTypes$ und $TaskTypes$ definiert werden. Im Allokationsmodell wird geprüft, ob es eine explizite Definition der Kosten (PolicyCost) gibt, wenn diese nicht festgelegt ist, werden die Allokationskosten über die *componentTypes* gesucht.

Dadurch entsteht eine Bibliothek von ComponentTypes. Die Kombination aus der ComponentType Bibliothek und der Systemmodulbibliothek ermöglicht die gleichzeitige Wiederverwendung von modellierten Policies und Kosten.

#### 5.3.3.2.3 *Skalieren durch Szenarien Bibliotheksansatz*

Bei der Modellierung von Edge Systemen mit vielen ähnlichen Systemmodulen erfordert die explizite Instanziierung jeder Systemkomponente im Systemmodell einen erhöhten Modellierungsaufwand und Speicherbedarf im Systemmodell. In dem Fall kann die Szenariomodellierung unter Verwendung der Systemmodul Bibliothek und der ComponentType Bibliothek eingesetzt werden.

Bei der Modellierung von Szenarien wird ein *Systemmodul* nicht direkt instanziiert, sondern es wird ein Repräsentant davon instanziiert. Dieser Repräsentant bekommt eine Multiplizitätszahl, mit der beschrieben werden kann, wie oft das Systemmodul im Edge Netzwerk vorkommt. Szenarien können auch rekursiv beschrieben werden und enthalten ebenfalls eine Multiplizitätszahl. Damit lässt sich der Policy Modellierungsansatz skalieren, um Systeme mit mehreren hundert Komponenten zu beschreiben, ohne jede Policy einzeln modellieren zu müssen.

```
//policy_metadaten
{
  "component_id": "...",
  "component_name": "...",
  "policy_constraints": [
    {
      "constraint_name": "...",
      "cost_expression": "...",
      "limit_expression": "value",
      "type": "less_than"
    },
    { ... }
  ],
  "policy_optimizations": [
    {
      "optimization_name": "...",
      "cost_expression": "value",
      "type": "min"
    },
    { ... }
  ],
  "policy_costs": [
    {
      "cost_type": "...",
      "assoc_component": "...",
      "policy_cost": "value",
      "type": "less_than"
    },
    { ... }
  ]
}
```

Abbildung 5.8: Beispiel einer Austauschdatei für Policy-Informationen im JSON-Format

### 5.3.4 Traversierung und Extraktion der allokationsrelevanten Informationen

Ziel der Policy Modellierung ist es, die allokationsrelevanten Informationen in einem modellbasierten Entwicklungsprozess formal zu beschreiben, um die automatische Allokation auf Basis der modellierten Informationen berechnen zu können. Zu dem Zweck wurde ein Konzept zur Traversierung und Extraktion des Modells entwickelt, bei dem die relevanten Daten in einem maschinenlesbaren Austauschformat gespeichert werden. In dieser Arbeit wird das als Metadaten des Systems bezeichnet. Für jede PolicyBasedElement Komponente aus dem Policy Modell werden die Policy Informationen extrahiert und gespeichert. D.h. für jeden *NodeProxy* und *TaskProxy* wird eine Datei erzeugt, die aus vier Teilen besteht: den Identifikationsdaten, den Policy Constraints, der Policy Optimierung und den Policy Kosten. Ein Beispiel für die Austauschdatei im JSON-Format ist in Abbildung 5.8 zu sehen.

In analoger Weise werden Metadaten für die anderen Komponenten im System extrahiert, die Policy Informationen enthalten. Die Metadaten werden zur Berechnung der Allokation verwendet, wie im Kapitel 6.2 beschrieben. Die modulare Extraktion und Darstellung der Metadaten hat unter anderem den Vorteil, dass bei Änderungen im System zusätzliche Metadaten-Dateien an den Solver gegeben werden können, ohne dass eine erneute vollständige Extraktion aller Policy Informationen notwendig ist.

## 5.4 Umsetzung im Entwicklungswerkzeug

Für die Umsetzung der oben beschriebenen Ansätze wurde das in Kapitel 4.2.2 beschriebene Entwicklungswerkzeug erweitert. Es wurde ein Policy Metamodell zur Beschreibung von allokationsrelevanten Informationen implementiert. Das Policy Metamodell wurde mit den relevanten Schichten des Systemmodells verknüpft. Darüber hinaus wurde eine Traversierungskomponente für die Extraktion der Policy Metadaten implementiert.

### 5.4.1 Eingesetzte Frameworks und Tools

Unter Verwendung der gleichen Werkzeuge wie im Kapitel 4.3 beschrieben, wurde das Policy Metamodell mit dem Eclipse Modeling Framework (EMF) als eigenständige Beschreibungssprache (DSL) implementiert. Die Konzepte zur Definition und Beschreibung des Allokationsproblems können auch in anderen Modellierungssprachen, wie z.B. SysML implementiert werden; die EMF-Implementierung dient der Verdeutlichung und Evaluation der Konzepte.

Für das Traversieren und Extrahieren der Policy Informationen aus dem Policy Modell wurde das Model2Text Framework Acceleo verwendet und als Austauschformat beispielhaft JSON gewählt. Die Verwendung einer Austauschdatei hat den Vorteil, dass die Modellierungsaktivitäten zeitlich von der Ausführung im Betrieb entkoppelt werden können, wie im kontinuierlichen Modellierungsprozess in Kapitel 4.2.3 beschrieben. JSON-Dateien wurden aufgrund ihrer weiten Verbreitung, der leicht lesbaren Textform und der großen Anzahl von Parsern und Generatoren ausgewählt, was die Integration in den Entwicklungsprozess und andere Werkzeuge vereinfacht.

### 5.4.2 Modellierung und Extraktion von Policies

Aus den Modellinformationen werden die Policy Metadaten extrahiert und in einem Austauschformat gespeichert, das als Input für die anschließende Allokationsberechnung verwendet wird.

Für die Verknüpfung von Systemmodell und Policy Modell wurde das Adapter Entwurfsmuster [100] aus dem Bereich der Softwareentwicklung verwendet und die beiden Alternativen mit Delegation und mit Vererbung betrachtet.

Die Verknüpfung über Vererbung, z.B. dass ein Hardware Element im Systemmodell vom *NodeProxy* des Policy Metamodells erbt, hat den Vorteil, dass die Policy Modellierung direkt erfolgen kann und das Hardware Element und bestehende Diagramme leichter erweitert werden können. Die Nachteile sind, dass das Systemmodell komplett neu generiert

werden müsste und das Systemmodell eine Abhängigkeit vom Policy Metamodell hat. Außerdem könnten dadurch auch einige Diagramme und Tools in ihrer Funktionalität beeinträchtigt werden. Dadurch wurde die Verknüpfung über die Delegation realisiert, dass z.B. ein *NodeProxy* eine Referenz auf das Hardwareelement im Systemmodell hat und damit die Informationen aus dem Systemmodell abrufen kann. Die Vorteile sind, dass die Modellierungsdiagramme ebenfalls erweitert werden können und gleichzeitig zwei separate Metamodelle und spätere Modelldateien gepflegt werden können. Somit kann die Policy Beschreibung als eine Art Add-on zum Systemmodell betrachtet werden, bei dem eine zusätzliche Beschreibungsmöglichkeit integriert wird, ohne die allein stehenden Funktionalitäten der bestehenden Systemmodellierung zu beeinträchtigen.

Auszüge aus dem Metamodell für Richtlinien sind im Kapitel 5.3.2 zu finden.

Zur leichteren Modellierung von Policies wurden verschiedene Wiederverwendungskonzepte berücksichtigt, wie im Kapitel 5.3.3 beschrieben. Diese wurden beispielhaft in EMF und den Sirius Diagrammen durch den Import und Export von Bibliotheksinstanzen und die Verwendung von Referenzen auf Instanztypen umgesetzt. Zum Beispiel könnten die wiederkehrenden Policy Informationen von Nodes in einem NodeType beschrieben werden, auf den die Nodes zeigen. Das ist in dieser Arbeit als eine Möglichkeit gedacht, die Policy Modellierung mit Wiederverwendungskonzepten zu kombinieren, die bereits in anderen Modellierungswerkzeugen existieren.

# Kapitel 6

# Leichtgewichtige Allokationsmethode für selbstadaptive Edge Systeme

In diesem Kapitel wird die Entwicklung der leichtgewichtigen Allokationsmethode für die automatische Entscheidung über die Zuordnung von Funktionen zu Recheneinheiten im laufenden Betrieb beschrieben. Es beinhaltet die Analyse des Tradeoffs zwischen Optimierung und Rechenaufwand sowie die Bedeutung von Leichtgewichtigkeit und Ansätze zur Modularisierung und Skalierung der Allokationsberechnung.

Für die automatische Adaption der Edge Computing Infrastruktur im Betrieb muss die Entscheidung über die Task Allokation ebenfalls im Betrieb und in einer überschaubaren Zeit, etwa im ms-Bereich, getroffen werden. Daraus ergibt sich die Notwendigkeit einer leichtgewichtigen Allokationsberechnungsmethode, die für die Integration in das Edge System im Betrieb ausgelegt ist.

Wie im Kapitel 5.2.1 beschrieben, ist eine effiziente Allokation dadurch gekennzeichnet, dass ein realisierbares Ergebnis bei angemessenem Ressourcenverbrauch erreicht wird. In dem Ansatz dieser Arbeit wird das Allokationsproblem durch komponentenspezifische Policies beschrieben. Die Hauptelemente der Policy Beschreibung sind die Constraints und ggf. die Optimierung bestimmter Aspekte. Erstere definieren die Gültigkeit bzw. Realisierbarkeit einer Allokation, letztere geben eine Vorgabe für die Optimierung der Allokation. Daraus ergibt sich ein multikriterielles Problem, bei dem der Trade-off zwischen Optimierung und vertretbarem Rechenaufwand und -zeit berücksichtigt werden muss, insbesondere wenn eine leichtgewichtige Berechnungsmethode angestrebt wird.

Daher wurden in dieser Arbeit Lösungsansätze für das multikriterielle Allokationsproblem untersucht und eine Allokationsmethode entwickelt, die auf Constraint Programming, lexikographischer Optimierung und mehrstufiger Ressourcenverwaltung basiert, so dass die Allokationsberechnung im laufenden Betrieb auf leichtgewichtige Weise durchführbar ist.

***Constraint Programming (CP)*** ist ein Paradigma zur Lösung von kombinatorischen Rechenproblemen. Es umfasst die Definition von Entscheidungsvariablen und Beschränkungen (Constraints). Dabei basiert die Constraint Programmierung auf der Idee, dass Berechnungsprobleme in Form von Beschränkungen erklärt werden können, die einer Gruppe möglicher Lösungen vorgegeben werden. Bei der Constraint Programmierung liegt der Schwerpunkt auf der Einhaltung von Beschränkungen, wodurch die Realisierbarkeit gewährleistet wird.

Der ***lexikografische Optimierungsansatz*** wird verwendet, um mehrere Optimierungsprobleme zu lösen, die in der Reihenfolge der hohen bis niedrigen Priorität der Ziele nacheinander gelöst werden. Das mehrstufige Ressourcenmanagement wird verwendet, um den Umfang des Problems einzugrenzen. Die lexikografische Optimierung und die mehrstufige Verwaltung kombinieren einen Ansatz, der nicht auf ein optimales Optimum abzielt, sondern die Policy Optimierungen in das Problem integriert und die Allokation in einer angemessenen Berechnungszeit ermöglicht.

Dadurch kann die Zuverlässigkeit und Verfügbarkeit der Recheninfrastruktur erhöht werden, indem Entscheidungen über die Allokation im Betrieb berechnet werden.

## 6.1 Analyse des Entscheidungsproblems und von multikriteriellen Lösungsansätzen

Die in dieser Arbeit entwickelte Policy Modellierung basiert auf der Beschreibung komponentenspezifischer Constraints und Optimierungen auf allen Ebenen des Systems. Definitionsgemäß handelt es sich bei mehreren Policy Optimierungen um ein multikriterielles Problem, das unterschiedliche Optimierungsziele haben kann. Diese Art von Problem verursacht in der Regel einen hohen Rechenaufwand und ist schlecht skalierbar. Aus dem Grund werden die Anforderungen an die Berechnungsmethoden und Ansätze zur Lösung von multikriteriellen Problemen analysiert, um eine leichtgewichtige Allokationsmethode trotz multipler Optimierungsziele zu erreichen.

### 6.1.1 Anforderungen an die Allokationsmethode

Um die Allokation basierend auf dem Policy Modell aus Kapitel 5.3 zu berechnen und im Betrieb zu integrieren, muss der Lösungsalgorithmus, in diesem Kapitel auch Allokationsmethode oder Berechnungsmethode genannt, folgende Anforderungen erfüllen.

**Funktionale Anforderungen:**
Festlegung von Beschränkungen: Es müssen Policy Constraints als Beschränkungen der möglichen Lösungen in den Lösungsalgorithmus integriert werden, um die Realisierbarkeit der Lösung zu überprüfen.
Umgang mit mehreren Optimierungszielen: Es sollen mehrere Policy Optimierungen berücksichtigt werden, um die Allokationslösungen zu verbessern.
Modularer Aufbau: Es soll möglich sein, Entscheidungsvariablen, Beschränkungen und Optimierungsziele auf modulare Weise zu definieren, so dass sie leicht hinzugefügt, entfernt oder angepasst werden können, wenn Änderungen im System vorgenommen werden.

**Nicht-funktionale Anforderungen:**
Zeitverhalten: Der Rechenaufwand und die Rechenzeit für die Lösung müssen begrenzbar sein und etwa im Bereich von Millisekunden (<1 sec) liegen.
Skalierbarkeit: Der Rechenaufwand sollte auch bei großen Problemen mit Teilnehmern im dreistelligen Bereich in einem vertretbaren Zeitbereich liegen (siehe oben).

Um die Allokationsmethode in die Adaptionsschleife des kontinuierlichen Entwicklungsprozesses zu integrieren, wie in Kapitel 4.2.3 beschrieben, muss die Entscheidung über eine Allokation oder Reallokation mit einer angemessenen Rechenzeit getroffen werden und auch in einem Edge System mit mehreren Teilnehmern funktionieren.

### 6.1.2 Analyse von multikriteriellen Lösungsansätzen

Die Einbeziehung mehrerer Optimierungskriterien in ein Entscheidungsproblem ist nicht trivial, und mehrere Arbeiten befassen sich mit unterschiedlichen Ansätzen. In dieser Arbeit werden drei Ansätze analysiert, wie mehrere Kriterien berücksichtigt werden können und die Vor- und Nachteile bei der Berechnung des Allokationsproblems diskutiert. Eine umfassende Analyse kann bei Marler und Arora [101] nachgelesen werden.

Ein multikriterielles Optimierungsproblem kann allgemein wie folgt beschrieben werden. Dabei ist $k$ die Anzahl der Optimierungsziele. $X$ ist der Vektor der Entscheidungsvariablen und somit ist $F(x)$ der Vektor der Zielfunktionen, auch Objektiv-, Kriterien- oder Kostenfunktion genannt.

$$\underset{x}{\text{Minimieren}}\ F(x) = [F_1(x), F_2(x), \ldots, F_k(x)]^T \tag{6.6}$$

Als *a priori Methoden* betrachten wir die gewichtete Summe, sowie die lexikographische Optimierung. A priori Methoden erlauben es dem Nutzer, Präferenzen anzugeben, die in Form von Zielen oder relativen Prioritäten verschiedener Ziele formuliert werden können.

***Gewichtete Summe (engl. weighted sum) [101]:*** Eine der gängigsten allgemeinen Skalierungsmethoden für die multikriterielle Optimierung ist die Methode der gewichteten Summe, bei der alle Zielfunktionen zu einer einzigen Funktion zusammengefasst werden (6.7). Dabei ist w ein Vektor von Gewichten, die vom Benutzer selbst festgelegt werden. So kann bei der Kombination der Zielfunktionen eine Präferenz vergeben werden. Allerdings kann der Benutzer hier keine umfangreichen Eingaben machen. Das ist nicht unbedingt ein Nachteil, da es Fälle geben kann, in denen die Präferenzangaben begrenzt sind oder nicht existieren. Beispielsweise, wenn mehrere Aufgaben ihre Latenzzeit minimieren wollen, aber nicht eindeutig priorisiert werden können. Dieser Ansatz ist einer der rechnerisch effizientesten, einfach zu verwendenden und weit verbreiteten Ansätze.

$$U = \sum_{i=1}^{k} w_i F_i(x) \tag{6.7}$$

***Lexikografische Optimierung (engl. lexicographic optimization) [101]:*** Bei der lexikographischen Methode werden die Zielfunktionen in der Reihenfolge ihrer Priorität angeordnet. Dann werden nacheinander die Optimierungsprobleme gelöst. $i$ steht für die Reihenfolge der Prioritäten. $F_j(x_j^*)$ steht für das gefundene Optimum bzw. Minimum bei der j-ten Iteration. Nach der Iteration wird eine neue Bedingung für die nächste Iteration

hinzugefügt, so dass nur Werte $F_j \leq$ des zuvor gefundenen $F_j(x_j^*)$ akzeptiert werden. Zur Relativierung der zusätzlichen Bedingungen und damit der Suchraum nicht zu sehr eingeengt wird, wodurch niedrig priorisierte Ziele effektiv kaum eine Optimierung erreichen können, werden relative Toleranzen $d_j$ hinzugefügt. Dadurch wird die Empfindlichkeit der endgültigen Lösung gegenüber der anfänglichen Priorität der Zielfunktion verringert. Die lexikografische Optimierung bietet also die Möglichkeit, verschiedene Ziele zu berücksichtigen, hat aber den Nachteil, dass die Optimierungen mehrfach berechnet werden müssen, was bei zu vielen Optimierungszielen ein Nachteil sein kann.

$$\begin{aligned} \underset{x \in X}{\text{Minimieren}} \quad & F_i(x) \\ \text{Unter Beachtung von:} \quad & F_j(x) \leq F_j(x_j^*), && j = 1, 2, \ldots, i-1 \\ & F_j(x) \leq (1 + d_j) F_j(x_j^*) && j = 1, 2, \ldots, i \end{aligned} \tag{6.8}$$

Im Vergleich dazu gibt es *a posteriori Methoden*, auch *generate-first-choose-later* Ansätze genannt. Die zuvor beschriebenen Methoden verwenden Standard Optimierungsverfahren (single objective). Andere Ansätze, wie z. B. genetische Algorithmen, können angepasst werden, um multikriterielle Probleme direkt zu lösen.

***Genetische Algorithmen:*** Genetische Algorithmen können unabhängig von der Art der Zielfunktionen und Beschränkungen eingesetzt werden. Sie kombinieren die Verwendung von Zufallszahlen und Informationen aus früheren Iterationen, um eine Population von Ergebnissen (eine Gruppe potenzieller Lösungen) zu bewerten und zu verbessern, anstatt jeweils ein einzelnes Ziel zu verfolgen. Obwohl genetische Algorithmen mehrere Ziele gleichzeitig berücksichtigen können und damit die Lösung von Sequenzen von Problemen mit nur einem Ziel vermeiden, müssen mehrere Iterationen durchgeführt werden, bis die Population potenzieller Lösungspunkte machbare und optimale Lösungen findet. Genetische Algorithmen berücksichtigen nämlich nicht nur die Optimierungsziele, sondern auch die Nebenbedingungen a posteriori. Daher sind sie nicht geeignet, um ein Problem zu lösen, das hauptsächlich auf Beschränkungen beruht. Außerdem ist der Rechenaufwand im Vergleich zu den anderen vorgestellten Ansätzen größer.

Aufgrund der Rechenkomplexität und der Tatsache, dass die Policy Modellierung zunächst auf der Beschreibung von Constraints basiert, wendet die Allokationsmethode dieser Arbeit die a priori Methoden gewichtete Summe und lexikographische Optimierung an. Um deren Nachteile zu kompensieren, wurde eine für das Allokationsproblem im Edge Computing Kontext geeignete Kombination entwickelt, die in Kapitel 6.2.1 beschrieben wird.

## 6.2 Konzept für eine leichtgewichtige Allokationsmethode im laufenden Betrieb

Für die automatische Berechnung der Allokationsentscheidung im laufenden Betrieb wird eine leichtgewichtige Methode benötigt. In diesem Zusammenhang wird beschrieben, was *leichtgewichtig* bedeutet, und es wird ein Ansatz für die Transformation von der Policy Modellierung zu einem Constraint Programming Problem vorgestellt. Darüber hinaus wird ein Ansatz für den Umgang mit der multikriteriellen Optimierung unter Verwendung der gewichteten Summe und der lexikographischen Methode sowie ein Multi-Level Ressourcenmanagement Ansatz vorgestellt.

### 6.2.1 Leichtgewichtige Methode basierend auf komponentenspezifischen Policies

#### 6.2.1.1 Definition der leichtgewichtigen Methode im Kontext der Task Allokation

Im Kontext der Informationstechnologie wird *leichtgewichtig* als Adjektiv für Programme verwendet, die relativ einfache und schnelle Berechnungen durchführen. Meistens bezieht sich dies auf einen geringen Speicherbedarf, eine geringe CPU Nutzung oder eine insgesamt geringe Nutzung von Systemressourcen.

*Leichtgewichtig* bedeutet im Kontext der hier betrachteten Task Allokation die relativ schnelle Suche nach einer hinreichend guten Allokation, die wenig Systemressourcen benötigt und damit für die Integration im laufenden Betrieb geeignet ist.

Das Gegenteil davon wäre eine rechenintensive Lösung einer Zuweisung, die viele Systemressourcen und eine längere Rechenzeit erfordert, was sich eher für eine Offline Berechnung eignen würde.

#### 6.2.1.2 Trade-off zwischen Optimierung und Rechenaufwand

Ein Trade-off beschreibt einen Zielkonflikt, bei dem eine Eigenschaft, Größe oder Qualität einer Gruppe oder eines Entwurfs im Gegenzug für einen Gewinn bei anderen Aspekten verringert oder eingebüßt wird. Vereinfacht ausgedrückt bedeutet ein Trade-off, dass ein Aspekt zunimmt und ein anderer hingegen abnehmen muss. Im Zusammenhang mit der Allokation ist der Kompromiss die Optimierung einer Allokationslösung im Verhältnis zu dem dafür erforderlichen Rechenaufwand in Form von Berechnungszeit.

Das Allokationsproblem ist aufgrund der vorhandenen Policy Optimierungen auch ein Optimierungsproblem, so dass neben anderen Aspekten auch eine optimale Lösung des Problems eine Rolle spielen wird. Für die Berechnung der Allokation im Betrieb ist eine

optimale Allokation anzustreben, aber im Edge Computing Kontext und wenn andere Aspekte wie Verfügbarkeit und Ressourcenbegrenzung in den Vordergrund drängen, spielt die Optimierung eine sekundäre Rolle. An erster Stelle steht die Suche nach realisierbaren Lösungen, die die Policy Constraints erfüllen. Der erforderliche Rechenaufwand bestimmt die Auswahl der Lösungsmethoden. Somit ist für die Integration in den laufenden Betrieb die Sicherstellung der Realisierbarkeit einer Allokation zunächst wichtiger als die Suche nach einem globalen Optimum.

**Optimierung:**
Auf der Optimierungsseite handelt es sich eher um einen Ansatz zur Lösungsfindung über Randbedingungen. Zu diesem Zweck wird das Paradigma der Constraint Programming (CP) verwendet. Dabei beschreibt die Constraint Programming ein Programm in Form von Entscheidungsvariablen, denen Werte zugewiesen werden müssen, und expliziten Beschränkungen für die Variablen, die gleichzeitig vergeben werden. Dann wird eine Menge von Belegungen für alle Variablen gefunden, die den Einschränkungen genügen [102]. Mit dem Ansatz der Constraint Programmierung können Grenzen definiert und formalisiert werden, um die Gültigkeit oder, in diesem Fall, die Realisierbarkeit einer Allokationslösung zu überprüfen und sicherzustellen. Daher stützt sich die Allokationsberechnung in dieser Arbeit insbesondere auf die Definition von Policy Constraints. Das Ziel ist also die Realisierbarkeit und nicht die Optimierung. Nichtsdestotrotz werden die Lösungen im beschränkten Raum mit Hilfe der gewichteten Summe und der lexikographischen Optimierung optimiert, um den Rechenaufwand zu reduzieren.

**Rechenaufwand:**
Auf der Seite des Rechenaufwands müssen die Rahmenbedingungen für die Integration in ein Edge Computing System und zwar im laufenden Betrieb berücksichtigt werden. Obwohl in dem Bereich, in dem diese Allokationsmethode angewendet wird, keine harte Echtzeitfähigkeit für die Reallokation von Diensten erwartet wird, spielt das Timing eine besondere Rolle und kann als weiche Echtzeit betrachtet werden. Abhängig von der Edge Computing Anwendung und dem Kontext können diese weichen Echtzeitanforderungen quantifiziert werden. In dieser Arbeit wird das in Größenordnungen berücksichtigt. Die Reallokation von Diensten wird im Bereich von Sekunden erwartet. Da die Entscheidung über die Zuteilung ein Teilschritt davon ist, wurde als Grenze definiert, dass die Zuteilung im Millisekundenbereich zu berechnen ist. Daher setzt die Arbeit auf Constraints und die schrittweise Optimierung einzelner Aspekte, solange die Berechnungszeit im Millisekundenbereich bleibt.

Bei der Trade-off Betrachtung von Optimierung und Rechenaufwand wird deutlich, warum Ansätze gewählt wurden, mit denen eine Lösung im Millisekundenbereich möglich ist, und zunächst auf die Realisierbarkeit der Lösung gesetzt wurde. Das bedeutet, dass für die Einhaltung eines vertretbaren Rechenaufwandes Abstriche bei der Optimierung in Kauf genommen werden.

### 6.2.2 Transformation der Policy Modellierung in Constraint Programming zur Allokationsberechnung

Wie im vorherigen Kapitel erläutert, liegt der Schwerpunkt der Allokationsberechnung auf der Durchführbarkeit basierend auf Beschränkungen. Das ergänzt sich mit der Policy Beschreibung von Kapitel 5.2. In diesem Kapitel wird erklärt, wie die Informationen aus dem Policy Modell mit Hilfe der Constraint Programmierung (CP) in ein ausführbares Lösungsproblem umgesetzt werden. Für die Lösung werden verfügbare CP-Solver verwendet. Die Details der Implementierung können dem Kapitel 6.3 entnommen werden.

Die Struktur des ausführbaren Allokationsproblems besteht aus drei Schritten, wie in Abbildung 6.1 dargestellt.

1. Zunächst werden die Daten vorbereitet, um das Problem für den Solver aufzustellen. Das beinhaltet die Interpretation und Verarbeitung der Metadaten der Policy aus Kapitel 5.3.4.

2. Anschließend werden auf der Grundlage der Policy Constraints die Beschränkungen aufgestellt. Darauf aufbauend kann bereits nach realisierbaren Allokationslösungen gesucht werden. Der Solver sucht nach Kombinationen, die die gegebenen Beschränkungen erfüllen, und hält an, wenn eine machbare Lösung gefunden wurde.

3. Im letzten Schritt werden die Policy Optimierungen nach dem hierarchischen Ansatz verarbeitet, wie im Kapitel 5.2.2 beschrieben. CP-Solver können nur mit einem Optimierungsziel arbeiten. Der genaue Umgang mit mehreren Optimierungsausdrücken ist in Kapitel 6.2.2.3 beschrieben.

#### 6.2.2.1 Aufbereitung der Problemdaten aus dem Policy Modell

Um eine Lösung mit einem Solver zu finden, muss er mit Daten befüllt werden. Zu dem Zweck werden die aus der Policy- und Systemmodellierung des Kapitels 5.3.2 extrahierten Metadaten in das Programm importiert. Die Aufbereitung der Daten gliedert sich in zwei Schritte, die Initialisierung und die Vorverarbeitung.

Bei der ***Initialisierung*** werden die Policy Kosten und Elementkennungen aus den Metadaten importiert und in Listen oder Matrizen instanziiert. Zunächst werden die Node Metadaten verarbeitet und eine Liste aller Nodes erstellt. Auf die gleiche Weise werden die Informationen zu den Task Metadaten verarbeitet. Mit den Listen der Nodes und Tasks können die Policy Kostenmatrizen instanziiert werden. Außerdem werden Listen mit den Policy Limits sowohl von Node und Tasks als auch von weiteren Systemkomponenten erstellt. Schließlich werden die Listen der Policy Constraints und Policy Optimierungen für

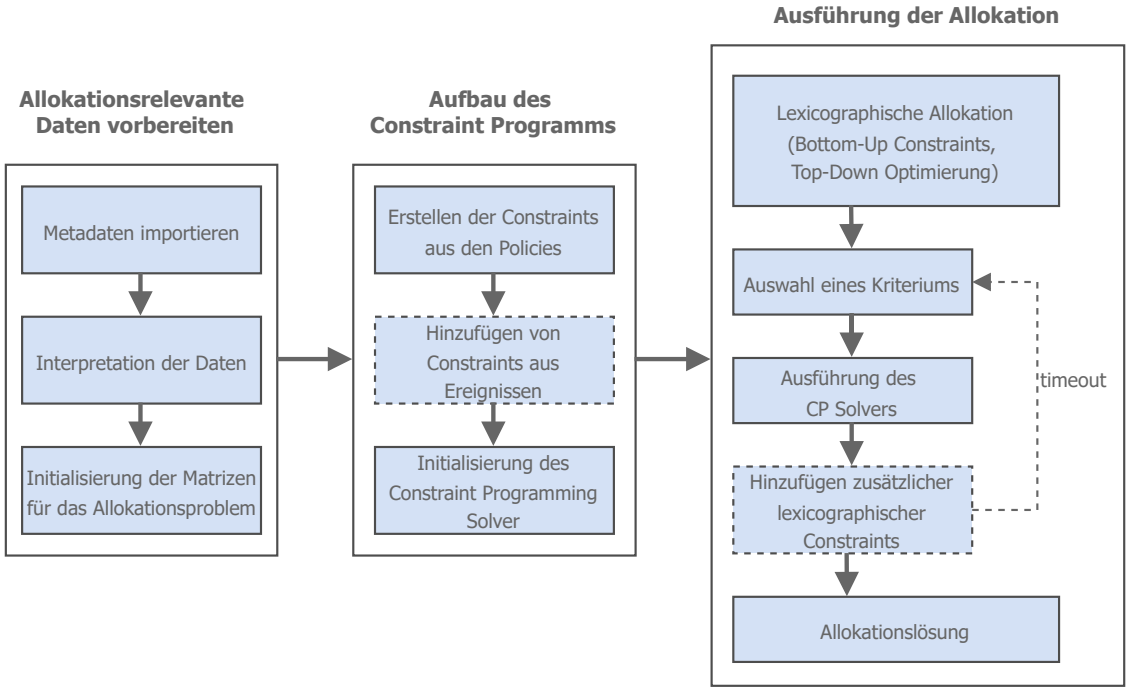



Abbildung 6.1: Struktur des ausführbaren Allokationsproblems

die spätere Integration in das Solver Modell erstellt. So werden die notwendigen Daten zur Lösung des Allokationsproblems im Solver geladen.

Die ***Vorverarbeitung*** umfasst die Erstellung zusätzlicher Kostenausdrücke oder Kostenmatrizen, die in den Policy Constraints oder der Optimierung definiert sind. So wird beispielsweise für die Gesamtlatenzkosten eine zusätzliche Kostenmatrix erstellt, die aus der Kombination der für Nodes oder Tasks beschriebenen Policy Kosten und den vom Systemmodell abhängigen Übertragungskosten besteht.

#### 6.2.2.2 Aufbau des Constraint Programms aus dem Policy Modell

Nach der Vorverarbeitung wird die Zuweisungsmatrix instanziiert, wobei die Spalten die Tasks und die Zeilen die Nodes darstellen. Die Zuweisungsmatrix besteht aus binären Variablen, die den Wert 0 oder 1 annehmen können. Der Solver sucht nach Kombinationen, indem er die Werte der Variablen variiert und die Beschränkungen überprüft. Mit den instanziierten Daten wird das Solver Modell wie im Kapitel 5.1.1.

Bei der Constraint Programmierung stehen die Beschränkungen (Constraints) im Vordergrund, mit denen der Suchraum der Lösungen eingeschränkt wird. Für die Allokation wird zunächst ein Basis Constraint Set erstellt, das sicherstellt, dass jeder Task genau einmal zugewiesen wird. Das wird dadurch realisiert, dass die Summe jeder Spalte (d.h. Task) der Zuweisungsmatrix genau eins sein muss.

Anschließend werden die extrahierten Policy Constraints aus der instanziierten Liste verarbeitet und die entsprechenden Constraint Sets erstellt. So werden beispielsweise Ressourcenkapazitäten von Nodes als Constraints erstellt, bei denen die Multiplikation der Zuweisungsmatrix mit der Policy Kostenmatrix oder Kostenausdruckmatrix pro Zeile (d.h. Node) kleiner oder gleich einem bestimmten Policy Limit sein soll.

Auf die gleiche Weise werden die Policy Constraints für die Tasks erstellt, wie z.B. die Latenzgrenzen für eine Task.

Dabei werden nicht nur die direkten Policy Kosten, z. B. für die Latenzzeit, berücksichtigt, sondern die gesamten Latenzkosten, zu denen die Latenzzeit für die Datenübertragung hinzukommt, je nachdem, wo die Task ausgeführt wird. Schließlich werden einzelne Policy Constraints erstellt, die sich auf Systemkomponenten beziehen.

#### 6.2.2.3 Multikriterielle Optimierung aus dem Policy Modell

Zum Schluss wird die Liste der Policy Optimierungen verarbeitet. Hierfür gibt es zwei Ansätze, die kombiniert werden, um verschiedene Optimierungsziele auf vereinfachte Weise zu behandeln. Aufgrund der Funktionsweise der Constraint Programmierung und ihres Solvers kann maximal ein Optimierungsziel im Solver definiert werden. Dabei werden die realisierbaren Lösungen in Bezug auf das Optimierungsziel verglichen.

Um mehrere Optimierungsziele für die Allokation bearbeiten zu können, werden sie nach dem ***lexikographischen Optimierungsansatz*** nacheinander gelöst. Bei jeder Ausführung wird der gefundene optimierte Wert für das Optimierungsziel als neue Randbedingung zum Solver Modell hinzugefügt. Um bei der nächsten Optimierung flexibel zu sein, enthält die neue Einschränkung eine relative Toleranz. Wenn zum Beispiel die Latenz einer Aufgabe optimiert wurde, das optimale Ergebnis $100ms$ ist und die relative Toleranz 10% beträgt. Es wird eine neue Bedingung hinzugefügt, bei der die Zuweisung nur einen Latenzwert von $100 \pm 10\% ms$ für diese Aufgabe akzeptiert.

Damit können die Optimierungsziele nacheinander betrachtet werden. Die Reihenfolge wird durch den Ansatz der hierarchischen Bedarfe aus dem Kapitel 5.2.2 bestimmt, bei dem die Optimierungsziele in Top-Down Reihenfolge berücksichtigt werden. Da mehrere Optimierungen durchgeführt werden, wird eine maximale Ausführungszeit für den Solver festgelegt. Diese kann beispielsweise eine Sekunde betragen und in der Zeit werden so viele Optimierungen wie möglich ausgeführt.

Da es Systemkomponenten gibt, die ähnliche Policy Optimierungen haben, z. B. wenn Tasks eine Policy Optimierung für denselben PolicyCost Typ haben. Das würde eine lexikografische Optimierung für jeden Task bedeuten, was schlecht skalierbar wäre.

Zu dem Zweck wird der Ansatz der Kombination von Optimierungszielen gleichen Typs zu einem Optimierungsziel mittels ***gewichteter Summe*** verwendet. Policy Optimierungen für gleiche Komponententypen und gleiche PolicyCost Typen werden zusammengefasst. So werden beispielsweise alle Tasks, die ihre Latenzkosten optimieren bzw. minimieren wollen, in einem Optimierungsausdruck zusammengefasst und dieser dann minimiert. Das bedeutet, dass für die lexikografische Optimierung nur ein Optimierungsschritt gelöst werden muss, anstatt einer Optimierung pro Task. Das hilft bei der Skalierung des Ansatzes.

Insgesamt werden also verschiedene Optimierungsausdrücke zunächst in lexikografischer Reihenfolge mit relativen Toleranzen in der Reihenfolge Top-Down betrachtet, und auf jeder Ebene werden die Optimierungen der gleichen Komponententypen und Kostenarten als gewichtete Summe zusammengefasst. Auf diese Weise wird auf den Trade-off zwischen Optimierung und Rechenaufwand, wie im Kapitel 6.2.1, eingegangen. Optimierungen werden ausgehend vom Gesamtsystem bis hin zu den einzelnen Komponenten durchgeführt. Mit diesem Ansatz wurden auch für große Probleme im Bereich von Millisekunden realisierbare und teilweise optimierte Allokationslösungen berechnet.

### 6.2.3 Mehrstufige Ressourcenverwaltung

Da Edge Computing Systeme aus einer großen Anzahl von Recheneinheiten und Anwendungen bestehen, ist die Skalierung des Lösungsansatzes von großer Bedeutung. Dafür wurde untersucht, wo die Allokation berechnet wird und welchen Umfang an Komponenten sie abdecken sollte. Es wurden sowohl ein zentrales, ein dezentrales, als auch ein verteiltes Ressourcenmanagement untersucht. In dieser Arbeit wurde ein Managementansatz ausgearbeitet, der diese Ansätze stufenweise kombiniert, um je nach Situation die Vorteile jedes Ansatzes zu nutzen, wie in Abbildung 6.3 und 6.2 dargestellt.

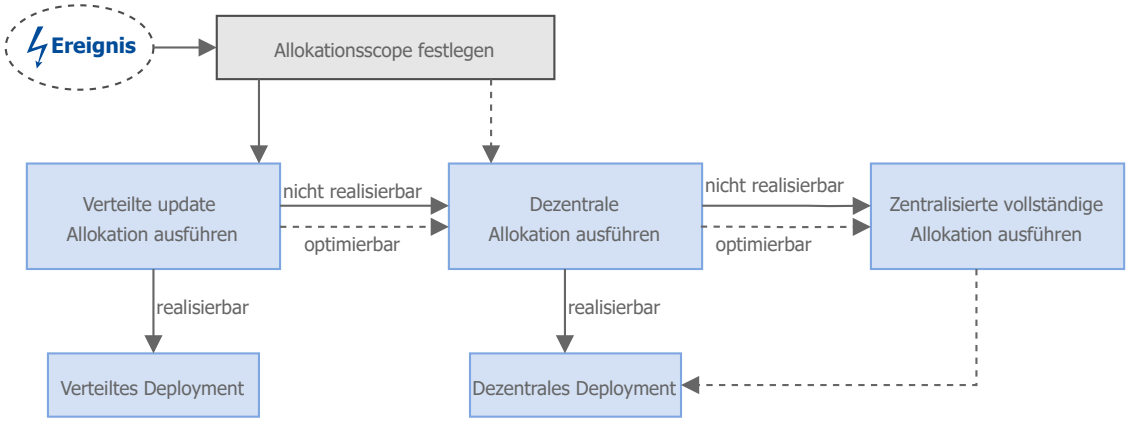



Abbildung 6.2: Ablauf der mehrstufigen Ressourcenverwaltung

**Verteilt:** Eine verteilte Zuweisungsberechnung wird durchgeführt, indem nur die geänderten Anwendungen und die betroffenen und in der Nähe verfügbaren Recheneinheiten berücksichtigt werden. Dadurch wird die Größe des Zuordnungsproblems (Anzahl der Tasks

und Recheneinheiten) auf die notwendigen Teilnehmer reduziert. Insbesondere bei der Durchführung von Allokationen nach On-Demand Ereignissen führt dies zu einer schnellen Entscheidung.

**Dezentral:** Wird keine mögliche Zuordnung gefunden oder soll der Umfang erweitert werden, wird in der zweiten Stufe eine dezentrale Berechnung der Zuordnung durchgeführt. In diesem Fall wird zunächst in Clustern nach einer Zuordnung für die geänderten Anwendungen gesucht.

**Zentral:** Wird keine mögliche Zuweisung gefunden, wird die Zuweisungsberechnung für das gesamte System durchgeführt. Dies kann mit einer Meldung an den Betreiber und einer umfangreichen Offline Zuteilungssuche kombiniert werden. Der Vorteil einer zentralen Zuteilungsentscheidung besteht darin, dass alle Teilnehmer berücksichtigt werden. Allerdings erhöht sich dadurch die Menge der zu berücksichtigenden Informationen, Beschränkungen und Optimierungen, was sich negativ auf die Berechnungszeit auswirken kann.

Das Konzept dieser Arbeit berücksichtigt die Vor- und Nachteile der verschiedenen Managementstufen und setzt auf die Kombination in einem mehrstufigen Managementsystem. So kann beispielsweise für eine erste Allokation eine zentrale Zuweisung offline und vor dem ersten Einsatz oder vor größeren geplanten Änderungen im Edge System berechnet werden. Um auf Änderungen im System reagieren zu können, wird eine verteilte Zuweisung bevorzugt, die die Verfügbarkeit durch schnelle Entscheidungen über eine Neuzuweisung erhöhen kann. Als Zwischenstufe kann die verteilte Zuweisung je nach Umfang und Situation eingesetzt werden. Das Konzept bleibt bei der Berechnung der Zuweisungsmethoden bewusst flexibel, um für das konkrete Edge System jeweils Akzente setzen zu können.

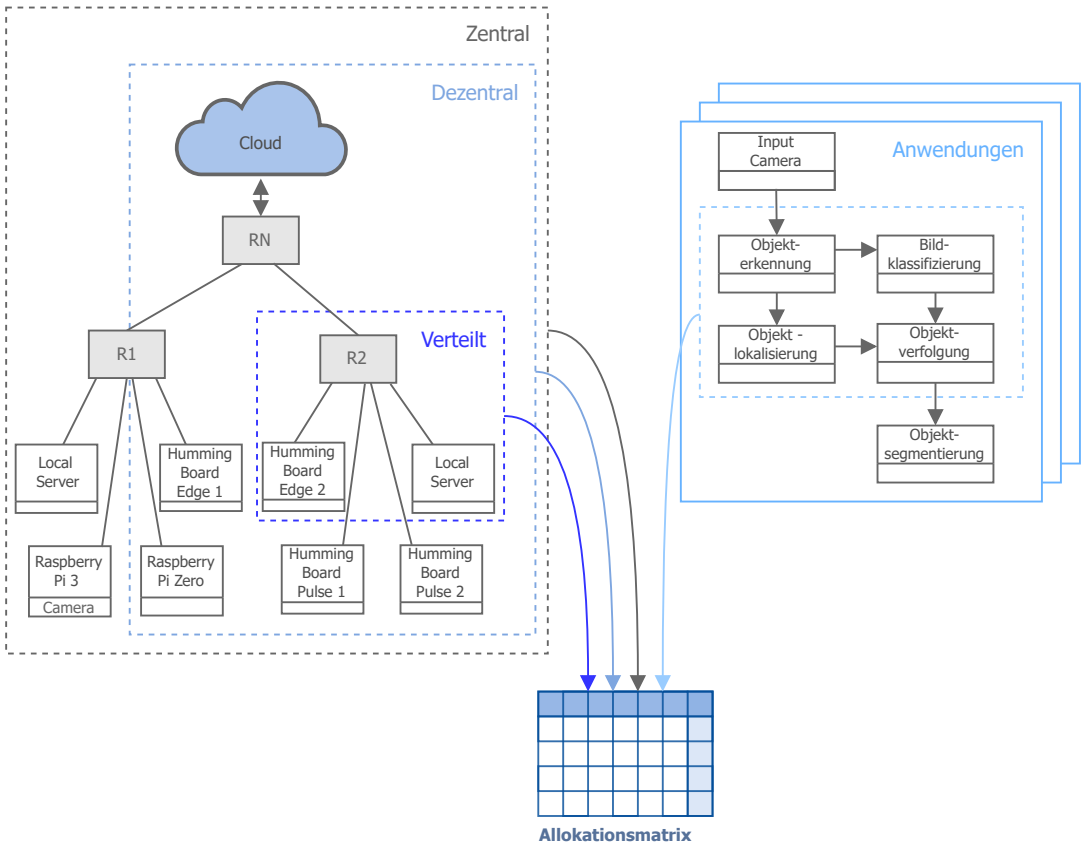



Abbildung 6.3: Schematische Darstellung der mehrstufigen Ressourcenverwaltung

## 6.3 Umsetzung

Für die Umsetzung der zuvor beschriebenen Allokationsmethode wurden verschiedene Solver evaluiert. Es wurden ein Open Source Solver und ein kommerzieller Solver verwendet. Darüber hinaus wurden die extrahierten Metadaten aus Kapitel 5.3.4 als Schnittstelle bzw. Austauschformat für die Integration der Policy Modellierung verwendet.

### 6.3.1 Auswahl des Solver

Das Allokationsproblem wurde als multikriterielles Problem definiert und verschiedene Lösungsansätze wurden untersucht. Es wurden evolutionäre Algorithmen (SPAM, SPEA2, NSGA2), lineare Optimierungen (ILP, MILP) und ein Constraint Satisfaction Problem Solver (CP-SAT) evaluiert und verglichen. Für den Solver wurden die folgenden Anforderungen definiert. In erster Linie muss der Solver in der Lage sein, sich leicht an den Policy Beschreibungsansatz anzupassen. Außerdem muss der Solver leichtgewichtig sein, d.h. der Rechenaufwand und die Dauer der Berechnung müssen möglichst gering sein, damit diese im laufenden Betrieb durchgeführt werden kann. Darüber hinaus wird keine optimale Lösung angestrebt, sondern eine Lösung, die schnellstmöglich alle Randbedingungen erfüllt und teilweise Optimierungen einbezieht.

Daher wurde der Einsatz von evolutionäre Algorithmen Solver ausgeklammert. Constraint Programming bzw. Constraint Satisfaction Solver wurden in erster Linie betrachtet, da sie darauf abzielen, machbare Lösungen zu finden, was die Anforderungen erfüllt. Zudem lässt sich das Konzept der Policy Beschreibung gut mit dem Constraint Satisfaction Solver verbinden.

Ein Open Source Solver (CP-SAT Solver) und ein kommerzieller Solver (CPLEX) wurden für die Implementierung der Allokationsmethode ausgewählt. Als Programmiersprache wurde Python gewählt, da sie weit verbreitet ist und verschiedene Solver APIs zur Verfügung stehen. Die Implementierung sollte die Möglichkeiten und Grenzen des Ansatzes aufzeigen.

Der ***CP-SAT Solver*** ist Teil der Google Optimization Tools Suite OR-Tools [103], einer Open Source, schnellen und portablen Software Suite zum Lösen kombinatorischer Optimierungsprobleme von Google. Der CP-SAT Solver ist ein Constraint Programming Solver, der auf Machbarkeit (finden einer machbaren Lösung) und nicht auf Optimierung (finden einer optimalen Lösung) basiert. Dabei stehen die Beschränkungen und Variablen im Vordergrund und nicht die Zielfunktion. Die Leistungsfähigkeit des CP-SAT Solvers wurde in den vergangenen Jahren durch den Gewinn mehrerer Preise im internationalen Constraint Programmierwettbewerb MiniZinc Challenge [1] demonstriert. Der CP-SAT

[1] https://www.minizinc.org/challenge.html. Accessed 30. Apr. 2022

Solver bietet nur eine Ein-Ziel Optimierung. Aus dem Grund wurde eine lexikographische Optimierung für das Allokationsverfahren dieser Arbeit erweitert und soll als Möglichkeit zur Validierung der Konzepte betrachtet werden. Für einen produktiven Einsatz kann die Implementierung der lexikographischen Optimierung entsprechend optimiert werden.

Das ***CPLEX Optimizer Solver Framework***[2] ist ein kommerzielles Programmsystem von IBM zur Modellierung und Lösung von Optimierungsproblemen mittels mathematischer Optimierung sowie Constraint Programming [104]. Für die Anwendung in dieser Arbeit wurde die öffentliche Variante verwendet, die mit bis zu 1000 Beschränkungen und Variablen arbeitet.

Zusätzlich wurde die Open Source Python API ***docplex***[3] verwendet, um auf den CPLEX-Solver in einem Python Programm zuzugreifen. Im Vergleich zu CP-SAT bietet docplex eine Implementierung und API für die direkte Nutzung der lexikographischen Optimierung, die auf eine optimierte Implementierung in CPLEX Optimizer zugreift und so die erforderliche Rechenzeit reduziert.

### 6.3.2 Implementierung der Allokationsmethode

Repräsentativ werden in diesem Kapitel Codeausschnitte aus der Implementierung der Allokationsmethode gezeigt. Die APIs des *CP-SAT* Solvers und des *docplex* Solvers sind ähnlich aufgebaut. Der Übersichtlichkeit halber wird die Implementierung unter Verwendung der *docplex* API gezeigt.

**Import der Daten:** Da die Metadaten im JSON-Format extrahiert wurden, wurde das Python Modul *json* verwendet. Dieses importiert die verschiedenen Metadaten-Dateien und interpretiert sie, um die Daten für den Solver zu instanziieren.

**Aufbereitung der Daten:** Die Daten werden in Form von Listen und Matrizen aufbereitet. Für die verschiedenen Multiplikationen zwischen Matrizen wurde u.a. das Python Modul *numpy* verwendet.

**Aufbau des Constraint Programms:** Für die Verwendung von *CPLEX Optimizer* wurde der Solver installiert und das python docplex Modul *docplex.mp.model* verwendet. In ähnlicher Weise wurde für den *CP-SAT Solver* das Modul *ortools.sat.python* verwendet. Die Instanziierung des Solver Modells und der Zuweisungsvariablen sieht bspw. wie folgt aus:

Quellcode 6.1: Solver Modell und Zuweisungsvariablen

```
mdl = Model(name='cp-problem', **kwargs)
# --- decision variables ---
mdl.assign_vars = mdl.binary_var_matrix(iter_nodes, iter_tasks)
```

[2]https://www.ibm.com/analytics/cplex-optimizer. Accessed 01. Aug 2022
[3]http://ibmdecisionoptimization.github.io/docplex-doc/. Accessed 01. Aug 2022

Dabei werden binäre Variablen verwendet. *iter_nodes* und *iter_tasks* sind Iteratoren für die instanziierten Listen von Nodes und Tasks.

Die grundlegende Bedingung, die sicherstellt, dass jede Aufgabe genau einmal zugewiesen wird, sieht wie folgt aus:

Quellcode 6.2: Grundbedingung für das Allokationsproblem

```
# assign each func exactly once
for t in iter_tasks:
    pass
    mdl.add_constraint( (sum(assign[n,t] for n in iter_nodes) == 1))
```

Für jede Task (Spalte der Zuweisungsmatrix) wird die Summe aller Nodes auf genau eins als Bedingung zum Solver Modell hinzugefügt.

Im Folgenden werden zwei Beispiele für das Hinzufügen von Beschränkungen für Nodes und Tasks gezeigt.

Quellcode 6.3: Beschränkungen für Nodes und Tasks

```
# node limit func
for n in iter_nodes:
    pass
    mdl.add_constraint( (sum(assign[n,t]*cost_func_matrix[n][t] for t
                                        in iter_tasks) <=
                                        nodelimit_yolo_func[n]) , '
                                        node-func-limit')
# task limit time
for n in iter_nodes:
    for t in iter_tasks:
        pass
        mdl.add_constraint( cost_time_proc_matrix[n][t]*assign[n,t] <=
                                            tasklimit_yolo_time[t], '
                                            task-time-limit')
```

Der erste Abschnitt fügt Beschränkungen hinzu, die die Anzahl der Funktionen für die Nodes bestimmen. Der zweite Abschnitt fügt Beschränkungen hinzu, die die Zeit bzw. Latenz der Funktionen für die Tasks bestimmen.

**Lexikographische Optimierung:** Für die lexikographische multikriterielle Optimierung wird die *minimize_static_lex* Methode der *docplex* API verwendet, für den *CP-SAT* Solver wurde diese eigenständig erweitert. In *docplex* sieht das bspw. wie folgt aus:

Quellcode 6.4: Lexikographische Optimierung mit docplex

```
mdl.minimize_static_lex(
    [mdl.total_cost_time, mdl.total_cost_energy]
    , reltols=[0.25,0.25]
    )
```

Dabei sind *total_cost_time* und *total_cost_energy* zwei Kostenausdrücke, die minimiert werden und jeweils die relative Toleranz *reltols* haben.

Im Falle des *CP-SAT* Solvers von OR Tools wurde die lexikographische Optimierung als wiederholte Ausführung des Solvers und Hinzufügen des optimierten Wertes als neue Randbedingung für die nächste Optimierung implementiert. Die Methode und der Aufruf dafür sehen wie folgt aus:

Quellcode 6.5: Lexikographische Optimierung mit CP-SAT Solver

```
def minimize_lex(mdl, solver, expr, reltol):
    mdl.Minimize(expr)

    status = solver.Solve(mdl)

    if status:
        obj = solver.ObjectiveValue()
        print(f'Total cost = {obj}\n')
        mdl.Add(expr <= trunc((1 + reltol) * obj))
...
...
  # Anwendung im Solver Model
    minimize_lex(model, solver, model.total_cost_time, 0.25)
    minimize_lex(model, solver, model.total_cost_energy, 0.25)
```

# Kapitel 7

# Edge Framework und Integration für die automatische Reallokation im laufenden Betrieb

In diesem Kapitel wird der Entwurf eines Edge Frameworks für die Verlagerung von Funktionen als Reaktion auf Veränderungen im System vorgestellt. Hierbei wird die Allokationsmethode unter Verwendung einer servicebasierten Architektur, Container Virtualisierung und Orchestrierungslösungen integriert.

Zur Realisierung einer automatischen selbstadaptiven Reallokation der Edge Computing Infrastruktur, wie in Kapitel 4.1 beschrieben, ist die Integration der Allokationsmethode in den Betrieb notwendig. Um eine Reallokation zu realisieren, ist jedoch nicht nur die Berechnung über eine Allokationsentscheidung notwendig, sondern auch eine Hardware/-Software Umgebung, die das realisieren kann, in dieser Arbeit Edge Framework genannt.

Daher wurde die Softwarearchitektur für Edge Computing Anwendungen analysiert, um eine Kapselung und lose Kopplung von Funktionen (Tasks) zu ermöglichen, so dass sie unabhängig voneinander angepasst, betrieben und aktualisiert werden können. Zu dem Zweck wurde eine serviceorientierte bzw. servicebasierte Architektur gewählt. Darüber hinaus ist eine Softwareumgebung erforderlich, damit die gekapselten Funktionen im laufenden Betrieb geändert, integriert und aktualisiert werden können. Dafür wurden Virtualisierungskonzepte analysiert und der Einsatz von Containern gewählt. Außerdem wird eine Instanz im Framework benötigt, die das Deployment der gekapselten Funktionen koordiniert und durchführt, sowie in der die Entscheidungen über eine Allokation integriert werden können. Zu diesem Zweck wurde ein Orchestrierung Framework ausgewählt und Mechanismen zur Steuerung des Deployments eingesetzt bzw. erweitert.

Diese Elemente werden in diesem Kapitel erläutert und diskutiert. Mit ihnen kann eine flexible Hardware/Software Umgebung bzw. ein Edge Framework entworfen werden, mit dem eine effiziente Reallokation implementiert werden kann und in das die entwickelte Allokationsmethode integriert werden kann.

## 7.1 Entwurf eines Edge Frameworks für flexible Task Allokation

Damit die Reallokation im laufenden Betrieb realisiert werden kann, ist eine Hardware/-Software Umgebung, in dieser Arbeit auch als Edge Framework bezeichnet, erforderlich, die das ermöglicht. Dieses Framework enthält die Softwarebausteine, die auf den Edge Knoten zur Ausführung von Edge Anwendungen eingesetzt werden. Dabei sind Mechanismen für die verteilte und effiziente Ausführung von Anwendungen enthalten. Dieses Kapitel erläutert die eingesetzten und erweiterten Konzepte für ein solches Framework, um die Allokationsmethode zu integrieren.

### 7.1.1 Service basierte Architektur für Edge Systeme

Für flexible und intelligente Edge Systeme, z. B. im Kontext von Industrie 4.0, sind dezentrale Aufgaben- und Funktionsverteilung zwei wesentliche Anforderungen [105]. Die zentralisierte Steuerungsarchitektur entwickelt sich zu einer dezentralen Architektur mit entsprechender Definition von Services [106].

Eine der möglichen Lösungen, um diese Anforderungen zu erfüllen, ist die Verwendung einer serviceorientierten Architektur (SOA). Sie gilt auch als eines der Gestaltungsprinzipien für I4.0 [107]. Eine serviceorientierte Architektur (SOA) ist ein Architekturmuster für die Software- und Systementwicklung. Tasks und Funktionen werden auf dezentralisierte intelligente Komponenten verteilt, die unabhängig voneinander kommunizieren können, um bestimmte Aufgaben zu erfüllen. Das heißt, die Kommunikation zwischen den Komponenten ist nicht statisch konfiguriert. Dienstnutzer (engl. Service Consumer) suchen zur Laufzeit nach Dienstanbietern (engl. Service Provider), da die Dienste lose gekoppelt sind, was das Plug-and-Produce Muster ermöglicht. Diese dynamische Architektur bietet die nötige Flexibilität, um schnell wechselnde Anforderungen der Anwender zu erfüllen und gleichzeitig die Interoperabilität und Wiederverwendbarkeit zu verbessern [108].

Die Flexibilität, die eine serviceorientierte Architektur mit sich bringt, eignet sich somit für die Realisierung von selbstadaptiven Edge Computing Systemen. Daher wird diese als Basis für die Softwarearchitektur des Frameworks verwendet. Das setzt voraus, dass die Softwarekomponenten durch die Definition von Service Consumer und Service Provider miteinander kommunizieren. Für die Verwendung des Ansatzes in dieser Arbeit werden die Tasks in Services gekappselt. Somit kann das Deployment von Tasks auch als das Deployment von Services bezeichnet werden.

Dieses Software Paradigma wird in verschiedenen Bereichen wie Web, Automotive, etc. eingesetzt. Der Autor hat zur Entwicklung einer Beschreibungssprache für die Modellierung serviceorientierter Architekturen für Industrie 4.0 Systeme beigetragen [97]. Diese Modellierungssprache wird in der vorliegenden Arbeit verwendet, um die Service Ebene zu modellieren, die eine Verbindung zu den zuzuordnenden Tasks hat.

### 7.1.2 Virtualisierungsansätze für Edge Systeme

Nachdem ein Software Paradigma für eine flexible Software Architektur auf logischer Ebene durch die Kapselung von Tasks in Services ausgewählt wurde, muss die Art und Weise bestimmt werden, wie die Tasks bzw. Services unabhängig von der zugrundeliegenden Hardware in modularer, flexibler Weise auf verschiedenen Edge Nodes deployed und verschoben werden können. Das ist notwendig, um eine selbstadaptive Reallokation zu ermöglichen.

Virtualisierung ist ein Ansatz in der Informatik, bei dem eine Abstraktionsschicht zwischen der Anwendung und der Ressource eingeführt wird. Dadurch können etwa Anwendungen flexibel und transparent von den zugrunde liegenden Ressourcen ausgeführt werden.

Für die Integration der Allokation im Betrieb wurden verschiedene Ansätze für das flexible Deployment von Anwendungen im Edge Computing Kontext untersucht. Die beiden wichtigsten Alternativen zur Softwarevirtualisierung sind die Virtualisierung mittels Containern oder mittels Hypervisor.

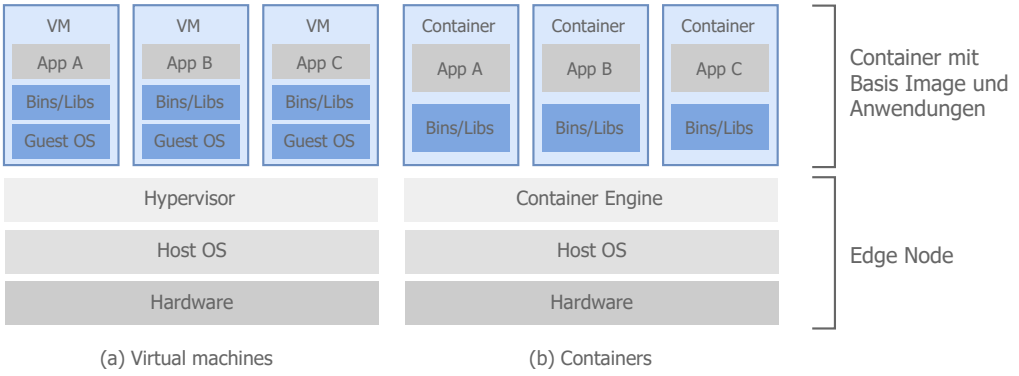



Abbildung 7.1: Schematischer Aufbau von Hypervisor und Container Virtualisierung

Bei der Virtualisierung mit Hypervisor werden virtuelle Maschinen verwendet, um mehrere Betriebssysteme gleichzeitig auszuführen oder um den Zugriff auf Ressourcen und Anwendungen zu isolieren. Allerdings ist damit ein relativ hoher Ressourcenaufwand verbunden, zum einen bei der Ausführung, zum anderen auch beim Deployment, da in der Regel jede virtuelle Maschine über ein Gastbetriebssystem verfügt.

Bei der Virtualisierung mittels Containerisierung werden mehrere Instanzen einer Laufzeitumgebung isoliert auf einem Hostsystem ausgeführt. Dabei wird in der Regel kein weiteres Betriebssystem gestartet, was als besonders ressourcenschonend gilt. Die Laufzeitumgebung lädt die notwendigen Abhängigkeiten und Bibliotheken und erleichtert so die Verlagerung auf andere Hostsysteme. Ein schematischer Aufbau der Container Virtualisierung im Vergleich zum Hypervisor ist in der Abbildung 7.1 dargestellt. Der Container kann in Base-Image und Application-Image aufgeteilt werden.

Für diese Arbeit wurde aus den folgenden Gründen ein containerbasierter Ansatz gewählt.

Erstens ermöglicht es ein flexibles, schnelles und kontinuierliches Deployment von Anwendungen. Daher ist der Ansatz im Cloud Bereich weit verbreitet. Die Nutzung für Edge Computing hat in den letzten Jahren zugenommen und optimierte Software Stacks für den Einsatz in eingebetteten Systemen sind heute verfügbar.

Darüber hinaus gibt es Arbeiten zur Containerisierung von Microservices mit einer sehr effizienten Boot-up Zeit, die für ein flexibles Deployment von Tasks interessant ist [109], [69]. Wenn das Basis-Image bereits deployed ist, werden nur die Microservices geladen. Das ist ein Faktor, der bei der Entscheidung über die Containerplatzierung berücksichtigt werden kann.

Es existieren Lösungen für die Container Orchestrierung, die bereits grundlegende Mechanismen für die Ressourcenverwaltung enthalten. Das bedeutet, dass in dieser Arbeit nicht erst die grundlegenden Funktionalitäten entwickelt werden müssen, sondern dass der Schwerpunkt direkt auf der Integration der Allokationsmethode liegen kann.

Außerdem basieren die Lösungen auf Open Source Software und entsprechen zunehmend Industriestandards, wodurch die Erweiterungen aus dieser Arbeit wiederverwendbar und für weitere industrielle Anwendungen relevant werden.

Der Autor hat in [110] gezeigt, wie Containerisierung und Microservice Architektur für eine flexible und redundante On-Demand Ausführung von Anwendungen genutzt werden können. Basierend auf diesen Erkenntnissen wurde die Virtualisierungslösung als Teil des Edge Frameworks entworfen.

### 7.1.3 Orchestrierung und Deployment Mechanismen

Gekapselte Tasks als Microservices, die in Containern laufen, müssen flexibel deployed werden, um ein selbstadaptives System zu erreichen. Insbesondere, wenn dies im laufenden Betrieb geschehen soll, ist eine Instanz in der Softwareumgebung bzw. im Edge Framework erforderlich, die das Deployment koordiniert und steuert.

Orchestrierung wird häufig im Zusammenhang mit serviceorientierten Architekturen und Virtualisierung eingesetzt, um die Ausführung dynamischer Dienste zu koordinieren. Orchestrierungslösungen haben sich in der Cloud bewährt und werden zunehmend auch im Edge Kontext eingesetzt. Sie müssen jedoch an die Ressourcenbeschränkungen im Edge Bereich angepasst werden. Obwohl sich Kubernetes (k8s) in den letzten Jahren zur De-facto Standardplattform für die Container Orchestrierung entwickelt hat, wächst eine Lösung, die geringere Ressourcenanforderungen hat (k3s[1]) und auf den Einsatz im Edge Kontext ausgerichtet ist.

Die Orchestrierungslösungen bringen Managementmechanismen mit sich, die das Deployment von containerisierten Diensten durch deklarative Konfiguration oder Automatisierungen erleichtern. Insbesondere aufgrund dieser Mechanismen werden die Lösungen als Teil des Edge Frameworks eingesetzt. So kann die entwickelte Allokationsmethode verwendet werden, um Allokationsentscheidungen während der Adaption zu treffen, und die Deployment Mechanismen der Orchestrierungslösungen werden gesteuert oder angepasst und müssen nicht neu entwickelt werden.

Zu dem Zweck werden die Schnittstellen zur Integration der Zuweisungsmethode betrachtet, wobei es zwei Alternativen gibt.

Der deklarative und indirekte Weg ist durch die Verwendung von Konfigurationsdateien. Hierzu werden aus der Allokationsmethode nach der Berechnung die zugehörigen Konfigurationsdateien mit den neuen Allokationen generiert. Die Konfigurationsdateien werden an den Orchestrator übergeben, um das Deployment von Anwendungen zu veranlassen.

[1] https://k3s.io/. Accessed 01. Aug. 2022

Der automatisierte und direkte Weg ist durch die Erweiterung der Automatisierungsinstanzen wie dem Scheduler von Kubernetes. Dieser bestimmt z.B. die Zuordnung von Services und durch die Erweiterung kann die in dieser Arbeit entwickelte Allokationsmethode direkt dafür aufgerufen werden. Somit wird die Lösung von der Allokationsmethode direkt an den Scheduler übergeben.

Beide Alternativen erfüllen die erforderliche Funktionalität. Die zweite Alternative erfordert mehr Aufwand, kann aber je nach Kontext effizienter sein. Obwohl beide Alternativen evaluiert wurden, wird im Folgenden die erste Alternative verwendet, da sie zur Verdeutlichung des Konzepts ausreichend ist.

### 7.1.4 Integration der Allokationsmethode in den Edge Framework für den Reallokationsprozess

In den vorherigen unterkapiteln wurden die Service Architektur, Containisierung und Orchestrierung erläutert, die für den Aufbau des Edge Frameworks in dieser Arbeit zum Einsatz kommen. Damit wird eine Umgebung aufgebaut, um die Allokationsmethode zu integrieren. So kann die Adapationsschleife der kontinuierlichen Entwicklungsprozesses aus Kapitel 4.2.3 realisiert werden.

Die Struktur des Edge Frameworks ist in der schematischen Darstellung in Abbildung 7.2 zu sehen. Als Beispiel sind drei EdgeNodes dargestellt. Jeder hat ein Host Betriebssystem, eine Agenteninstanz für die Orchestrierung und eine Container Laufzeit. Auf diesen werden Container ausgeführt, die die Implementierung der Services oder Tasks mit den notwendigen Bibliotheken kapseln. Außerdem gibt es einen Masternode, auf dem die zentrale Instanz des Orchestrators ausgeführt wird. Dieser besitzt die Komponenten zur Verwaltung der Agenten und damit der containierisierten Services. Hierbei lassen sich der Controller Manager und der Scheduler hervorheben.

Änderungen werden in der Monitoring Phase überwacht. Im Falle einer Änderung aktiviert der Controller Manager die Allokationsmethode, um eine neue Allokation zu finden. Die Allokationsmethode berechnet die Allokation auf der Grundlage der Metadaten und der aufgetretenen Änderungen. Für die neue Zuweisung werden die entsprechenden Konfigurationsdateien erstellt und an den Orchestrator übergeben. Anschließend führt der Orchestrator die neue Bereitstellung auf der Grundlage der berechneten Zuweisung aus.

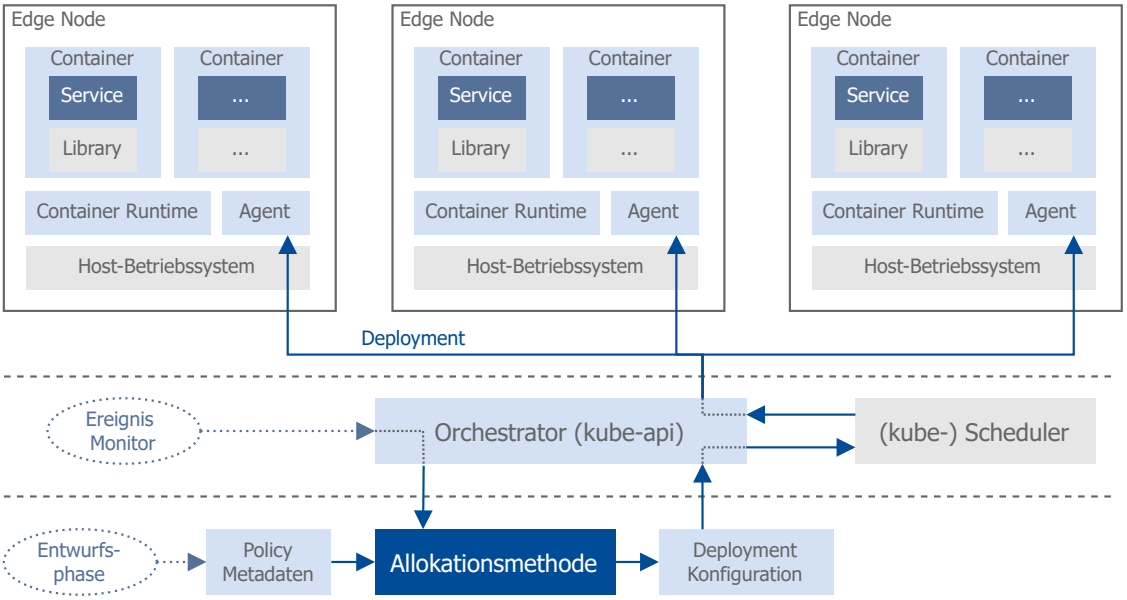



Abbildung 7.2: Schematischer Aufbau des Edge Frameworks

## 7.2 Umsetzung des Edge Frameworks und Integration der Allokationsmethode

Für die Umsetzung der oben beschriebenen Ansätze und des Reallokationsprozesses aus Kapitel 4.2.3 wurden verschiedene Plattformen analysiert. Anschließend wurde die Allokationsmethode mit Hilfe der Orchestrierungsmechanismen für das Deployment von Containern integriert.

### 7.2.1 Eingesetzte Frameworks und Tools

Bei der Auswahl einer Plattform für die dynamische Zuweisung von Diensten auf Edge Geräten wurden die folgenden funktionalen und nicht-funktionalen Anforderungen berücksichtigt.

Funktionale Anforderungen: Das Framework muss die Möglichkeit haben, Dienste auf verschiedenen Nodes zu starten und zu stoppen. Das Framework muss in der Lage sein, Dienste zu verlagern.

Nicht-funktionale Anforderungen: Das Framework muss die Integration in einen modellbasierten Entwicklungsprozess unterstützen. Das Framework muss verschiedene Container Engines unterstützen. Das Framework muss auf dynamische Änderungen, wie z. B. Ausfälle von Nodes, reagieren.

Darüber hinaus wurden die folgenden Bewertungskriterien auf Basis der Anforderungen für die Bewertung der verschiedenen Plattformen definiert, siehe Tabelle 7.1.

- K-1: Die Plattform soll Services unter Verwendung von container basierter Virtualisierung ausführen und verteilen.
- K-2: Die Plattform soll die Möglichkeit haben, Dienste zwischen ausführenden Knoten zu verschieben.
- K-3: Die Plattform soll auf Änderungen reagieren können.
- K-4: Die Plattform soll mit Hilfe von textuellen Konfigurationsdateien steuerbar sein.
- K-5: Die Plattform soll für den Edge Kontext optimiert sein.
- K-6: Die Plattform soll aus Open Source Software bestehen.

K3S[2] ist eine Kubernetes Distribution, die für Internet der Dinge (IoT) und Edge Computing entwickelt wurde. Es handelt sich um eine leichtgewichtige Variante von Kubernetes[3],

[2] https://k3s.io/. Accessed 01. Aug. 2022
[3] https://kubernetes.io/docs/home/. Accessed 01. Aug. 2022

Tabelle 7.1: Analyse von Edge Computing Plattformen

| Kriterien / Plattformen | AWS IoT Greengrass | Azure IoT Edge | Eclipse ioFog | openIoTFog | K3S (Kubernetes) |
|---|---|---|---|---|---|
| K-1 | - | x | x | x | x |
| K-2 | - | - | - | x | x |
| K-3 | x | x | - | x | x |
| K-4 | x | x | x | x | x |
| K-5 | - | - | x | x | x |
| K-6 | - | - | x | - | x |

einem Open Source System zur Automatisierung des Deployments, der Skalierung und der Verwaltung von containerisierten Anwendungen. Diese speziell an Edge Geräte angepasste Kubernetes Version ist für ARM-Prozessoren optimiert. Darüber hinaus wurde Docker[4] für die Container Virtualisierung verwendet.

### 7.2.2 Aufbau des Edge Frameworks

Der Aufbau des Edge Frameworks mit k3s und Docker ist in Abbildung 7.2 schematisch dargestellt. Die Struktur ist vergleichbar mit der in Kapitel 7.1.4.

In der Terminologie von k3s ist jeder EdgeNode ein k3s Worker Node oder k3s Agent. Dieser enthält u.a. ein Kublet und einen Kube-Proxy. Das Kublet dient dazu, den operativen Zustand jedes Knotens zu überprüfen und sicherzustellen, dass alle Container auf dem Knoten funktionieren. Es kümmert sich um das Starten, Stoppen und Warten von Containern, die in Pods organisiert sind. Der Kube-Proxy wird zur Verwaltung der Kommunikation und des Datenverkehrs verwendet. Pods sind die eigentlichen Einheiten, die in Kubernetes orchestriert werden. Diese wiederum enthalten die containerisierten Services bzw. Tasks.

Außerdem ist der Master Knoten in k3s die Steuerungsebene. Der Kubernetes Master ist die eigentliche Steuereinheit des Clusters, die dessen Arbeitslast verwaltet und die Kommunikation innerhalb des Systems steuert. Er besteht aus mehreren Komponenten, welche die Hochverfügbarkeit von Clustern ermöglichen. Der Controller Manager und der Scheduler werden näher betrachtet. Der Controller Manager verwaltet mehrere Controller. Controller sind Komponenten, die den tatsächlichen Zustand des Clusters steuern und mit dem API-Server kommunizieren, um die von ihnen verwalteten Ressourcen (z. B. Pods oder Dienste) zu erstellen, zu aktualisieren und zu löschen. Beispielsweise können mit den Controllern Replikationsmechanismen verwendet werden. Der Scheduler ist die erweiterbare Komponente, die auf der Grundlage der Ressourcenverfügbarkeit auswählt,

[4] https://docs.docker.com/. Accessed 01. Aug. 2022

auf welchem Node ein ungeplanter Pod ausgeführt werden soll. Controller und Scheduler sind für diese Arbeit wichtig, weil sie zur Integration der Allokationsmethode verwendet werden, um einen Reallokationsprozess zu realisieren.

### 7.2.3 Integration der Allokationsmethode

Für den (Re-)Allokationsprozess im Betrieb wurde die automatische Policy basierte Allokationsmethode integriert. Die dafür verwendeten Bausteine und Schnittstellen sind in Abbildung 7.2 dargestellt.

Aus der Entwurfsphase werden die Metadaten mit den allokationsrelevanten Informationen importiert. Die verteilten Kubelets sowie der Kube Controller Manager überprüfen den Betriebszustand der Pods und Nodes im Cluster. Gibt es Änderungen, wird die Allokationsmethode aktiviert. Dann wird der Allokation Solver aus Kapitel 6.2 ausgeführt und die berechnete Allokation entweder in eine Konfigurationsdatei geschrieben und an die Kube-API für das neue Deployment übergeben oder direkt in den Kube-Scheduler integriert. Der Orchestrator führt dann die notwendige Reallokation der Pods mit den containerisierten Services auf die Worker Nodes durch.

# Teil III

# Evaluation und Schlussfolgerung

# Kapitel 8

# Anwendungsorientierte Evaluation der Allokationsmethode

Dieses Kapitel zeigt die untersuchten industriellen Anwendungsfälle, die zur Evaluierung der entwickelten Allokationsmethode herangezogen wurden. Die erste Anwendung beschäftigt sich mit dem Offloading der Perzeption von fahrerlosen Transportfahrzeugen in einer Fabrikumgebung. Die zweite Anwendung beschäftigt sich mit den Veränderungen im System bei der Personenerkennung im Umfeld von kollaborativen Arbeitsmaschinen.

Für die Evaluation der Ansätze dieser Arbeit wurden Anwendungen im Edge-Computing-Kontext analysiert. Wie in Kapitel 1.1 erläutert, spielen Vernetzung und Ressourcenbeschränkungen sowie die Heterogenität der Recheneinheiten eine besondere Rolle. Zudem handelt es sich nicht um geschlossene Anwendungen und Systeme, sondern sie entwickeln sich im Laufe der Zeit weiter und sind Veränderungen ausgesetzt. Daher muss die Edge-Computing-Infrastruktur in der Lage sein, auf die Veränderungen zu reagieren und sich ihnen anzupassen, um Ausfallzeiten zu verringern.

Aus der Perspektive eines Sense-Plan-Act-Paradigmas ergeben sich verschiedene Anwendungen, die unterschiedliche Anforderungen an Echtzeitfähigkeit und Kritikalität stellen. Im Allgemeinen kann man zwischen Echtzeit- oder sicherheitskritischen Anwendungen und Best-Effort-Anwendungen unterscheiden. Die erste Gruppe von Anwendungen wird in einem echtzeit- und sicherheitskritischen Regelkreis benötigt, z. B. zur Kollisionsvermeidung. Die zweite Gruppe von Anwendungen wird nach einer Best-Effort-Strategie mit den verfügbaren Ressourcen ausgeführt. Dies bedeutet gleichzeitig, dass sie schwankender Verfügbarkeit unterliegen können, z. B. aufgrund von Überlastung oder Security Problemen. Der Ansatz dieser Arbeit konzentriert sich auf diese Art von Anwendungen, um bei Ereignissen oder Änderungen eine on-demand Reallokation berechnen zu können und die Ausfallzeit von Services durch Selbstadaption zu reduzieren.

Diese Eigenschaften von Edge-Computing-Systemen und einem sich verändernden System oder Umfeld finden sich in verschiedenen Bereichen wie Industrie 4.0, Industrial Internet of Things, Smart Cities, Transportsysteme. Für die Evaluation in dieser Arbeit wurden zwei industrielle Anwendungen aus dem Industrie 4.0-Kontext analysiert und die Allokationsmethode integriert. (I) Zum einen die Funktionsverlagerung im Kontext einer intelligenten Baustelle mit kollaborativen Arbeitsmaschinen in Kapitel 8.1 und (II) zum anderen das Offloading im Kontext eines intelligenten Werksgeländes mit vernetzten Mikromobilen in Kapitel 8.2.

Bei beiden Anwendungsfällen konnte die entwickelte Allokationsmethode unter den Aspekten Machbarkeit, komponentenspezifische Policies, multikriterielle Optimierungsvarianten, On-Demand Ereignisse und Skalierbarkeit evaluiert werden. Insgesamt konnte die zentrale These bestätigt werden, dass auf Basis von Constraint Policies in kurzer Zeit eine realisierbare Allokation gefunden werden kann. Integriert in den modellgetriebenen kontinuierlichen Entwicklungsprozess konnte sichergestellt werden, dass die Allokation entwurfskonform ist, d.h. dass die Anforderungen, Constraints und Ziele aus der Entwurfsphase stets erfüllt werden. Die Auswirkung von Änderungen bzw. Bedarfsereignissen und das Zeitverhalten des Solvers für die Integration im laufenden Betrieb wurden mit verschiedenen Szenarien analysiert. Sowohl in überschaubaren Szenarien mit 9 Teilnehmern und bis zu 108 Allokationsvariablen als auch in großen Szenarien mit bis zu 120 Teilnehmern und 3000 Allokationsvariablen wurde ebenfalls die Skalierung des Ansatzes untersucht. Es

wurde ein lineares Skalierungsverhalten der Berechnung bestätigt und anhand der mehrstufigen Ressourcenmanagement-Ansätze gezeigt, wie das Zeitverhalten reduziert werden kann, um die Berechnung im Betrieb zu ermöglichen. Mit den bestätigten Eigenschaften der Allokationsmethode und mit dem erarbeiten Edge Framework konnte somit eine automatische und effiziente Zuweisung von Anwendungen auf Recheneinheiten im Edge Computing Kontext im Betrieb realisiert werden.

Der Rest des Kapitels ist wie folgt aufgebaut. Zunächst wird die Evaluation der beiden Anwendungen getrennt aufgezeigt. Für jede Anwendung werden der Systemkontext und der Anwendungsfall sowie der Aufbau und die Vorgehensweise der Evaluation erläutert. Anschließend werden die Umsetzung und die Ergebnisse der verschiedenen Evaluationsszenarien gezeigt. Am Ende jeder Anwendung steht eine Bewertung und Diskussion der Ergebnisse. Die Evaluation der ersten Anwendung in Kapitel 8.1 konzentriert sich auf die Machbarkeit und den Einsatz von hierarchischen Policies. Die Bewertung der zweiten Anwendung in Kapitel 8.2 konzentriert sich auf den Umgang mit multikriteriellen Problemen und die Skalierbarkeit der Allokationsmethode. Abschließend wird in Kapitel 8.3 eine anwendungsübergreifende Diskussion der Evaluation sowie das Transferpotential in andere Anwendungsbereiche behandelt.

## 8.1 Anwendung 1 - Funktionsverlagerung im Kontext einer intelligenten Baustelle mit kollaborativen Arbeitsmaschinen

### 8.1.1 Kontext und Anwendungsfall

Kollaborative Arbeitsmaschinen gewinnen zunehmend an Bedeutung. Ein vernetztes Rechensystem ist dafür unabdingbar. In diesem Zusammenhang werden immer mehr Funktionen benötigt, die in unmittelbarer Nähe der Maschinen ausgeführt werden. Die Komplexität dieses Systems steigt, da mehrere Teilnehmer beteiligt sind und sich der Funktionsumfang im Laufe der Zeit ändern kann. Diese Situation ist repräsentativ für Edge Computing Systeme. Es entsteht ein Netzwerk von Teilnehmern, die Rechenressourcen haben und benötigen, um ihre Anwendungen auszuführen.

Es gibt verschiedene Arten von Funktionen in einem solchen Netzwerk, neben Kontrollfunktionen z.B. für die Steuerung der Maschinen gibt es auch systemunterstützende Funktionen wie z.B. die Umgebungsüberwachung, die den Manövrierbereich und den Arbeitsprozess überwacht und im Falle einer Warnung Alarme auslöst. Ein derartiges Warnsystem wurde als Anwendungsfall im Kontext von kollaborativ arbeitenden Maschinen betrachtet.

**Systemkontext**

Der Systemkontext ist eine intelligente Baustelle, auf der es mehrere Teilnehmer gibt: verschiedene Arten von Arbeitskränen, intelligente Kamerasysteme, menschliche Operatoren, usw. Ein wesentliches Merkmal aller Kranarten ist die Verwendung eines Seils zum Heben einer Traglast. Dies ist ein wichtiges Merkmal, vor allem unter dem Gesichtspunkt der Sicherheit, denn eine Last am Seil kann wie ein Pendel wirken und den Kran zum Umkippen bringen, wenn er nicht richtig gesteuert wird. Dies ist die Hauptursache für die meisten Unfälle mit Kränen. Ein Drittel aller tödlichen Unfälle oder Verletzungen mit bleibender Behinderung im Baugewerbe stehen im Zusammenhang mit Kränen [111].

Auch die Tragfähigkeit eines Krans ist begrenzt. Sollte es notwendig sein, ein Gewicht zu heben, das die Tragfähigkeit übersteigt, gibt es zwei Möglichkeiten: die Verwendung eines Krans mit einer geeigneten Tragfähigkeit oder die Erhöhung der Tragfähigkeit durch die Zusammenarbeit mehreren Kränen. Deshalb werden mobile Arbeitskräne betrachtet, die eine gemeinsame Aufgabe durchführen, z. B. einen Hebevorgang, und dafür untereinander und mit anderen Teilnehmern der Baustelle sowie mit einem Operator kommunizieren.In [108] hat der Autor eine servicebasierte Middleware für teilautomatisierte kollaborative Arbeitskräne präsentiert.

Darüber hinaus gibt es ein kamerabasiertes System, das die Arbeitsumgebung der Kräne überwacht, damit keine Unfälle passieren. Hierfür werden Personen und andere relevante


Abbildung 8.1: Funktionspipeline des Umgebungsüberwachungssystems

Objekte mit einer KI-basierten Objekterkennungsfunktion im Kamerabild detektiert, und wenn sie eine gefährliche Situation darstellen, wird eine Warnung an den Operator gesendet. Darüber hinaus kann im Notfall die Bewegung der Kräne gestoppt werden. Dieses Unterstützungssystem ist ein Beispiel für verschiedene Funktionen, die im Rahmen einer intelligenten Baustelle möglich sind.

**Anwendungsfall**

Für die Evaluation der Allokationsmethode dieser Arbeit wird die Funktionsverschiebung der Aufgaben des Umgebungsüberwachungssystems betrachtet. Das Umgebungsüberwachungssystem besteht aus verschiedenen Funktionen, wie in der Funktionspipeline aus Abbildung 8.1 zu sehen ist. Dabei ist die KI-basierte Objekterkennungsfunktion ein ideales Beispiel für eine rechenintensive Anwendung, die auf verschiedenen Recheneinheiten eine unterschiedliche Performance aufweist.

Für jede Kamera im System muss eine Funktionspipeline ausgeführt werden, und wenn Änderungen im System auftreten, wie das Hinzufügen neuer Kameras, aber auch der Ausfall von Edge Recheneinheiten, müssen die zugehörigen Funktionen verschoben oder neu instanziiert werden. Dies ist ein Beispiel für on-demand Ereignisse im Edge System, für die eine Reallokation notwendig ist.

Für die Ausführung der Objekterkennungsfunktion stehen verschiedene Edge Geräte zur Verfügung, z. B. Edge Recheneinheiten in den Mobilkränen, ein spezieller ML-Beschleuniger in einem Sensormast oder ein lokaler Edge Server.

### 8.1.2 Aufbau und Vorgehen

Als Evaluationsgegenstand wird die Reallokation von Objekterkennungsfunktionen untersucht. Zu diesem Zweck wurden die folgenden Dimensionen untersucht:

- Die Machbarkeit der Modellierung und Berechnung der Allokation
- Die Vorteile der gestuften hierarchischen Policies
- Die Abdeckung verschiedener On-Demand Ereignisse
- Die Robustheit und Skalierbarkeit des Ansatzes

Folgende Evaluationskriterien wurden zu diesem Zweck analysiert:

- Die Lösbarkeit (engl. Satisfiabilty) des Problems für die Analyse der Machbarkeit
- Das Zeitverhalten der Berechnung für die Analyse der Skalierbarkeit
- Die Gesamtkosten für die Analyse der Anforderungen und Ereignisse

Die folgenden Informationen wurden für die Evaluation verwendet: Für die Ausführung der Objekterkennungsfunktionen auf verschiedenen Hardwareplattformen wurden Messungen durchgeführt, um die Policy Werte zu ermitteln. Für die drahtlose Kommunikation wurde WLAN (Wi-Fi 6) verwendet, um die Kommunikationskosten in die Allokationsberechnung zu integrieren.

Für die Evaluierungsszenarien wurde außerdem eine Zuordnung aller Funktionen im lokalen Edge-Server bzw. in den Kran-Edge-Devices als Baseline verwendet, die eine manuelle statische Allokation darstellen soll.

**Vorgehen**

Zuerst wurde ein ***Basisszenario*** definiert, anhand dessen die Machbarkeit des Ansatzes in einem kleinen Rahmen validiert wurde. Das Basisszenario besteht aus 3 Objekterkennungsfunktionen und 4 Edge Geräten, was ein überschaubares Szenario darstellt. Aufbauend auf dem Basisszenario wurden die verschiedenen Dimensionen der Evaluation in weiteren Szenarien untersucht. Danach wurde untersucht, wie sich die Berücksichtigung von hierarchischen Policies auswirkt. Dies wird als ***hierarchische Policies Szenarien*** bezeichnet. Anschließend wurden verschiedene Änderungen im System vorgenommen und in 5 ***On-Demand Ereignis Szenarien*** analysiert. Schließlich wurde die Anzahl der Teilnehmer und Funktionen variiert, um die Robustheit und Skalierbarkeit des Ansatzes in 27 verschiedenen ***Variationsszenarien*** zu evaluieren.

### 8.1.3 Modellierung und Policy Beschreibung

Zur Verdeutlichung werden die Systemmodellierung und die Policy Beschreibungen des Basisszenarios im Folgenden gezeigt.

**Systemmodellierung**

Das System in den verschiedenen Ebenen. Die logische Funktionsarchitektur in Abbildung 8.2. Die Softwarearchitektur in Abbildung 8.3 und die Hardwarearchitektur in 8.4.

**Policy Modellierung**

Zur Veranschaulichung der Modellierung von Policy Werten ist in Abbildung 8.5 eine Auswahl von Task-Typen, in Abbildung 8.6 Constraint und Optimierung Policies für Tasks und in Abbildung 8.7 für Edge Nodes dargestellt.

Die verwendeten Policy Werte wurden in Messungen an verschiedenen Recheneinheiten ermittelt und sind in Tabelle 8.9 zusammengefasst.

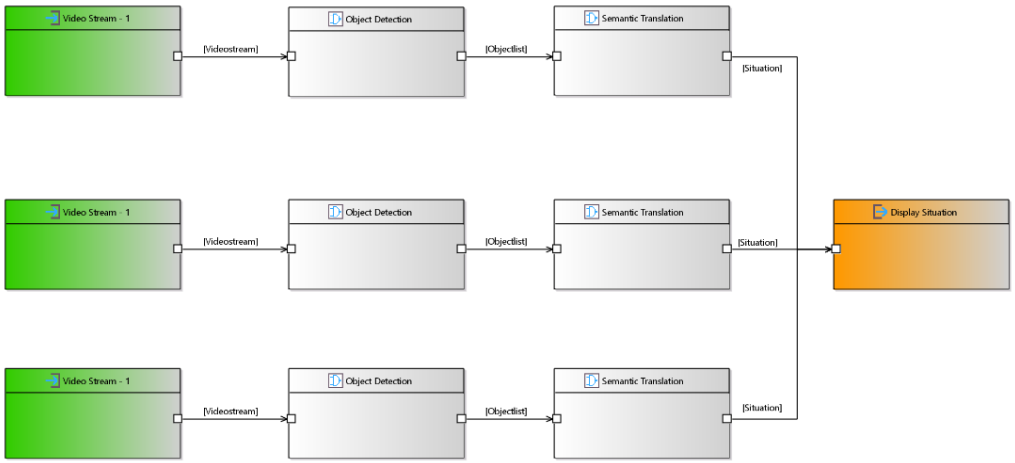



Abbildung 8.2: Anwendung 1 - Logische Funktionsebene

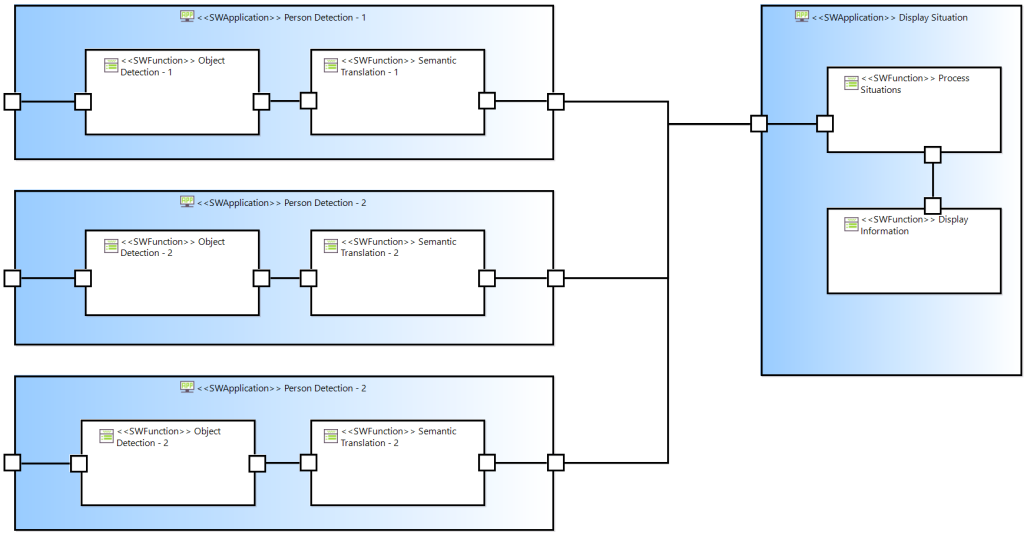



Abbildung 8.3: Anwendung 1 - Software Ebene

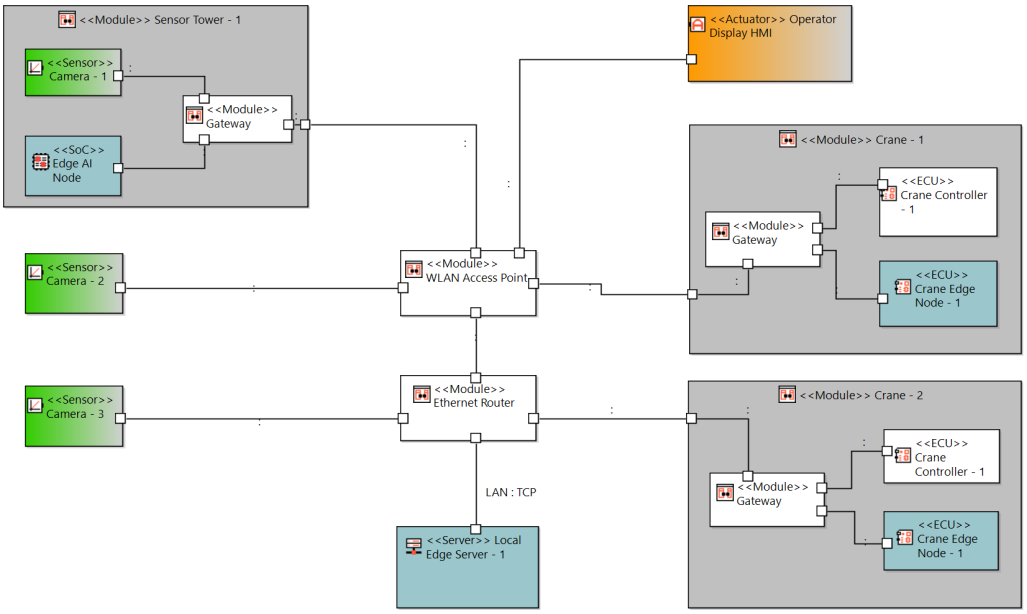



Abbildung 8.4: Anwendung 1 - Hardware Ebene

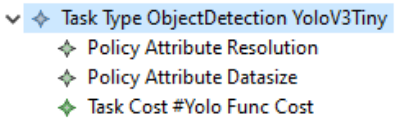



Abbildung 8.5: Task Type Modellierung

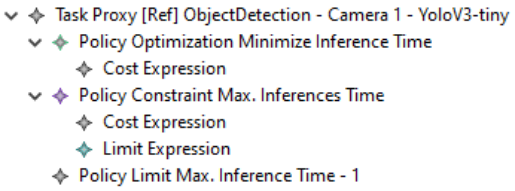



Abbildung 8.6: Task Constraint und Optimierung Policies

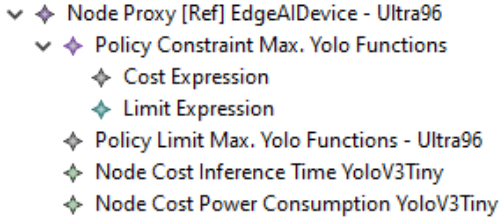



Abbildung 8.7: Node Constraint und Optimierung Policies

Tabelle 8.1: Input Information Auszug

| **Komponente** | **Policy** | **Art** | **Wert** | **Einheit** |
|---|---|---|---|---|
| EdgeAIDevice (Ultra96) | Cost | Yolo-Latenz | 0.02 | s |
| EdgeAIDevice (Ultra96) | Cost | Yolo-Power | 7.9 | W |
| EdgeAIDevice (Ultra96) | Limit | Instanzen | 1 | - |
| LocalEdgeServer(AGX) | Cost | Yolo-Latenz | 0.031 | S |
| LocalEdgeServer(AGX) | Cost | Yolo-Power | 30 | W |
| LocalEdgeServer(AGX) | Limit | Instanzen | 4 | - |
| CraneEdgeDevice1 (NX) | Cost | Yolo-Latenz | 0.02 | S |
| CraneEdgeDevice1 (NX) | Cost | Yolo-Power | 9.8 | W |
| CraneEdgeDevice1 (NX) | Limit | Instanzen | 2 | - |
| CraneEdgeDevice2 (TX2) | Cost | Yolo-Latenz | 0.022 | S |
| CraneEdgeDevice2 (TX2) | Cost | Yolo-Power | 6.9 | W |
| CraneEdgeDevice2 (TX2) | Limit | Instanzen | 1 | - |
| ObjectDetection1 (Cam1) | Limit | Latenz | 0.033 | s |
| ObjectDetection2 (Cam2) | Limit | Latenz | 0.033 | s |
| ObjectDetection3 (Cam3) | Limit | Latenz | 0.061 | s |

Tabelle 8.2: Policies Auszug

| **Komponente** | **Policy** | **Art** | **Cost Expression** | **Limit Expression** |
|---|---|---|---|---|
| EdgeAIDevice (Ultra96) | Constraint | $\leq$ | Yolo-Instanzen | Instanzen |
| LocalEdgeServer(AGX) | Constraint | $\leq$ | Yolo-Instanzen | Instanzen |
| LocalEdgeServer(AGX) | Optimierung | min | Yolo-Instanzen | - |
| CraneEdgeDevice1 (NX) | Constraint | $\leq$ | Yolo-Instanzen | Instanzen |
| CraneEdgeDevice1 (NX) | Optimierung | min | Yolo-Instanzen | - |
| CraneEdgeDevice2 (TX2) | Constraint | $\leq$ | Yolo-Instanzen | Instanzen |
| ObjectDetection1 (Cam1) | Constraint | $\leq$ | Yolo-Latenz | Latenz |
| ObjectDetection1 (Cam1) | Optimierung | min | Yolo-Latenz | - |
| ObjectDetection2 (Cam2) | Constraint | $\leq$ | Yolo-Latenz | Latenz |
| ObjectDetection2 (Cam2) | Optimierung | min | Yolo-Latenz | - |
| ObjectDetection3 (Cam3) | Constraint | $\leq$ | Yolo-Latenz | Latenz |
| ObjectDetection3 (Cam3) | Optimierung | min | Yolo-Latenz | - |

Tabelle 8.3: Basisszenario Messungen

| **Szenario** | **Realisier-barkeit** | **Solver Zeit [ms]** | **Gesamte Latenz [s]** |
|---|---|---|---|
| Nur Policy Constraints | ja | 16 | 0.095 |
| Globale Optimierung | ja | 22 | 0.076 |
| Nur CraneDevices (manuell) | nein | - | 0.086 |
| Nur EdgeServer (manuell) | ja | - | 0.117 |

### 8.1.4 Umsetzung und Ergebnisse

Nach der Modellierung des Systems und der Policies wurden die allokationsrelevanten Daten aus dem Modell extrahiert und zur Ausführung des Constraint Programming Solvers weitergegeben. Die hier gemessenen Werte wurden in demselben System ausgeführt und sind daher in sich vergleichbar. Es wurden die Aspekte der Machbarkeit, der gestuften hierarchischen Policies, der On-Demand Ereignisse und der Skalierbarkeit des Konzepts untersucht.

#### 8.1.4.1 Evaluation Basisszenario

**Aufbau des Szenarios**

Für das Basisszenario wurde ein System mit 3 Objekterkennungsfunktionen und 4 Edge Geräten mit der zugehörigen Policy Beschreibung aufgebaut. Es wurden entweder nur die Constraints oder nur eine Optimierung, nämlich die gesamte Optimierung der Latenzkosten, betrachtet.

**Messungen**

Um die Auswirkungen der verschiedenen Merkmale des Konzepts zu analysieren, wurden getrennte Messungen durchgeführt, wie in Tabelle 8.3 dargestellt.

Bei dem überschaubaren Szenario werden die Auswirkungen einer bestimmten Modellierung deutlich. Obwohl das Szenario relativ kompakt ist, ergeben sich 64 verschiedene Allokationsmöglichkeiten. Diese ergeben sich aus der Kombination der 3 Funktionen und 4 Edge Geräte. Damit wird deutlich, dass bei größeren Szenarien die manuelle Beherrschung der zunehmenden Komplexität kaum möglich ist. Bereits durch den Einsatz von Constraint Policies konnte das Problem auf 19 machbare Allokationsvarianten reduziert werden.

**Analyse**

Durch die Messungen wurde die Machbarkeit der Modellierung und Berechnung in einem überschaubaren Beispiel gezeigt. Darüber hinaus wird die Komplexität des Allokationsproblems bereits in dem relativ einfachen Anwendungsfall deutlich und wie das Problem durch die Verwendung von Constraints von 64 auf 19 machbare Varianten reduziert werden kann. Das Basisszenario zeigt, dass eine statische manuelle Allokation als Baseline, z.B. dass alle Funktionen auf den Edge Recheneinheiten der Kräne berechnet werden, nicht machbar ist. In diesem Fall und mit der Einstellung der Policy-Constraints ist z.B. klar, dass die Baseline in diesem Fall nicht realisierbar ist.

Darüber hinaus wurde die gewichtete Summenoptimierung angewendet, um die Gesamtlatenz zu minimieren. Die Gesamtlatenz im System wurde dadurch verbessert und in diesem vereinfachten Fall wurde das globale Minimum gefunden. Gleichzeitig hat sich allerdings der Zeitaufwand des Solvers leicht erhöht. Daraus ist ersichtlich, dass in dem einfachen Szenario die Berücksichtigung von Optimierungen einen minimalen Einfluss auf das Zeitverhalten hat. Für eine sinnvolle Betrachtung werden in den folgenden Szenarien Fälle mit mehr Teilnehmern untersucht.

**Zusammenfassung - Basisszenario**

Im Basisszenario wurde die Machbarkeit des Konzepts der Allokationsmethode erfolgreich evaluiert und mit einer manuellen pauschalen Allokation verglichen.

#### 8.1.4.2 Evaluation Hierarchische Policies Szenarien

**Aufbau des Szenarios:**

Ausgehend vom Basisszenario mit Optimierung wurde der Einsatz von hierarchischen Policies untersucht. Zum einen die stufenweise Optimierung als eine Möglichkeit, eine multikriterielle Optimierung durchzuführen. Zum anderen die Beschreibung von Clustern und Applikationen mit Policies.

Hierzu wurden die folgenden Optimierungen eingefügt.

- Node B-AGX und Node C-NX optimieren (als gewichtete Summe)
- Optimierung von Cam1
- Cam1 und Cam2 zu einer Applikation zusammenfassen, da diese gemeinsam einen Kran überwachen und diesen optimieren.

Damit wurde die stufenweise Optimierung durchgeführt. D.h. zuerst die Optimierung der Applikation (Cam1+Cam2), dann Cam1 und schließlich die Knoten.

**Messungen**

Tabelle 8.4: Hierarchische Policies Messungen

| **Szenario** | **Realisier-barkeit** | **Solver Zeit [ms]** | **Gesamte Latenz [s]** |
|---|---|---|---|
| Nur Policy Constraints | ja | 16 | 0.095 |
| Globale Optimierung | ja | 24 | 0.089 |
| Hierarchische Policies | nein | 23 | 0.078 |

Es wurden verschiedene Messungen mit und ohne Stufen sowie mit Clustern und Applikationen durchgeführt, die in Tabelle 8.4 zusammengefasst sind.

**Analyse**

Ausgehend vom Basisszenario konnte das Eingreifen der hierarchischen komponentenspezifischen Policies validiert werden. Im Szenario ohne hierarchische Stufen, aber mit den zusätzlichen Policies für die Nodes B und C, konnte eine modifizierte Zuordnung erfolgen. Im Szenario mit den hierarchischen Policies wurde zunächst die Applikation optimiert, die aus den Tasks für Cam1 und Cam2 besteht, und dann die Policies der Nodes und der restlichen Tasks berücksichtigt. Diese Allokation führt zu einem relativ ähnlichen Gesamtlatenzwert wie die Variante der gewichteten Summenoptimierung aus dem Basisszenario. Allerdings wird in diesem Fall die Allokation an die Bedürfnisse der jeweiligen Komponenten angepasst.

Alle berechneten Szenarien sind realisierbar, und beim Zeitverhalten ist eine leichte Erhöhung der mittleren Berechnungsdauer im Vergleich zum Basisszenario festzustellen. Der Grund dafür könnte die Durchführung von 3 lexikographischen Optimierungen im Vergleich zu nur einer gewichteten Summenoptimierung im Basisszenario sein.

**Zusammenfassung - Hierarchische Policies-Szenarien**

Mit der Einbeziehung zusätzlicher Policies und Cluster sowie Applikationen wurden die komponentenspezifischen Policies und die stufenweise Optimierung als Teil der Allokationsmethode evaluiert. Dabei zeigte sich der Vorteil gegenüber der Basisallokation ohne Optimierung in der Reduzierung der Gesamtlatenz bei gleichzeitiger bedarfsgerechter komponentenspezifischer Allokation.

#### 8.1.4.3 Evaluation On-Demand Szenarien

**Aufbau des Szenarios**

Ausgehend vom Basisszenario mit hierarchischen Policies und der Cluster Beschreibung wurden die folgenden Szenarien mit Änderungen im System aufeinander aufbauend bewertet:

Tabelle 8.5: Hierarchische Policies Messungen

| **Szenario** | **Realisier-barkeit** | **Solver Zeit [ms]** | **Gesamte Latenz [s]** |
|---|---|---|---|
| 1-Zusätzliche Funktionen | ja | 25 | 0.184 |
| 2-Neue Nodes | ja | 36 | 0.173 |
| 3-Node Ausfall | ja | 34 | 0.192 |
| 4-Überlastete Node | nein | 16 | - |
| 5-Node Austausch | ja | 36 | 0.17 |

1. Neue bzw. zusätzliche Funktionen: 3 Kameras werden dem System mit ihren eigenen Policies hinzugefügt.
2. Neue Nodes: Ein zweiter EdgeServer mit einem NX-Knoten wird hinzugefügt.
3. Ausfall eines Nodes: Basierend auf Szenario 2 fällt der Knoten A-U96 aus.
4. Überlastete Nodes: Basierend auf Szenario 3 ist der Knoten B-AGX nicht verfügbar.
5. Austausch von Nodes: Anstelle des überlasteten B-AGX aus Szenario 4 wird ein weiterer EdgeServer mit 2xU96 Nodes hinzugefügt.

**Messungen**

Die Ergebnisse der Messungen für die oben genannten Szenarien sind in Tabelle 8.5 zusammengefasst. In diesen Szenarien wurde die Anzahl der Tasks verdoppelt, da doppelt so viele Kameras verwendet werden. Anschließend konnte die Reaktion auf verschiedene Änderungen im System untersucht werden. Die update Allokation unter Berücksichtigung der vorherigen Allokationen und die vollständige Allokation wurden durchgeführt, um die Unterschiede zu analysieren.

**Analyse**

Im ersten On-Demand Szenario werden 3 Kameras mit 3 Aufgaben hinzugefügt. Die Gesamtlatenzkosten sind bei der vollständigen Allokation ohne vorherige Zuweisungen und bei der update Allokation mit vorheriger Zuweisung sowie bei der Gesamtoptimierung relativ ähnlich. Allerdings gibt es einen Unterschied in der Ausführungszeit des Solvers, der auch in den anderen Szenarien auftritt. Die Ausführungszeit des Solvers ohne vorherige Zuweisungen beträgt etwa 25-36 ms, während sie bei der Allokation mit vorherigen Zuweisungen etwa 16 ms beträgt. Dies zeigt, dass die verteilte update Allokation, bei der nur die sich ändernden Tasks und Nodes zugewiesen werden, schneller ausgeführt werden kann. Außerdem wurden die Gesamtlatenzkosten dadurch nicht verschlechtert und somit ist die update Allokation zu bevorzugen.

Im zweiten On-Demand Szenario wird ein neuer EdgeServer mit einer NX-Recheneinheit hinzugefügt. Bei der update Allokation mit vorheriger Zuweisung wurden die ersten drei Kameratasks mit der bisherigen Zuweisung belassen und die 3 neuen Tasks aus dem vorherigen Szenario neu zugewiesen. Bei der Ausführung ohne vorherige Zuweisung wurden die beiden ersten Kameras dem neuen EdgeServer zugewiesen, was eine aktive Verschiebung der zuvor ausgeführten Funktionen bedeuten würde. Dadurch verbessert sich die Gesamtlatenzzeit nur minimal, während die Verschiebung zahlreicher Funktionen zur Laufzeit zu einem Overhead führt. In Bezug auf das zeitliche Verhalten des Solvers ist Ähnliches zu beobachten wie im ersten On-Demand Szenario. Daher wird die Variante mit der update Allokation bevorzugt.

Im dritten und vierten OnDemand Szenario wird der Ausfall von zwei Nodes betrachtet. Zunächst der Ausfall des Nodes A-U96. Dieser wird durch eine Reallokation kompensiert. Auch hier wird eine Allokation mit vorheriger Zuweisung bevorzugt. Danach fällt der erste EdgeServer mit dem Knoten B-AGX aufgrund von Überlastung aus und es kann keine neue Zuordnung gefunden werden. Aus diesem Grund wären die Funktionen auf diesem Edge-Knoten nicht verfügbar. Dies wird jedoch im nächsten Szenario kompensiert.

Im fünften OnDemand Szenario kommt ein neuer EdgeServer hinzu, der über zwei U96 Nodes verfügt. Mit diesen Nodes können im Falle einer update Allokation die Funktionen zugewiesen werden, die im vorherigen überlasteten Node ausgeführt wurden. Betrachtet man das Zeitverhalten des Solvers, so bestätigt sich, dass die verteilte update Allokation vorteilhafter ist als eine vollständige Allokation.

**Zusammenfassung – On-Demand Szenarien**

Insgesamt wurde die Allokationsmethode mit verschiedenen On-Demand Szenarien validiert. Es wurde ermittelt, dass eine verteilte update Allokation mit vorheriger Zuweisung vorzuziehen ist, da die vom Solver benötigte Zeit kürzer ist und das Ergebnis im Vergleich zu einer vollständigen Allokation nicht nachteilig ist. Das bestätigt die Verwendung eines mehrstufigen Managementkonzepts, das in der ersten Stufe eine verteilte update Allokation nur für die geänderten Tasks und Nodes sucht.

#### 8.1.4.4 Evaluation Variationsszenarien

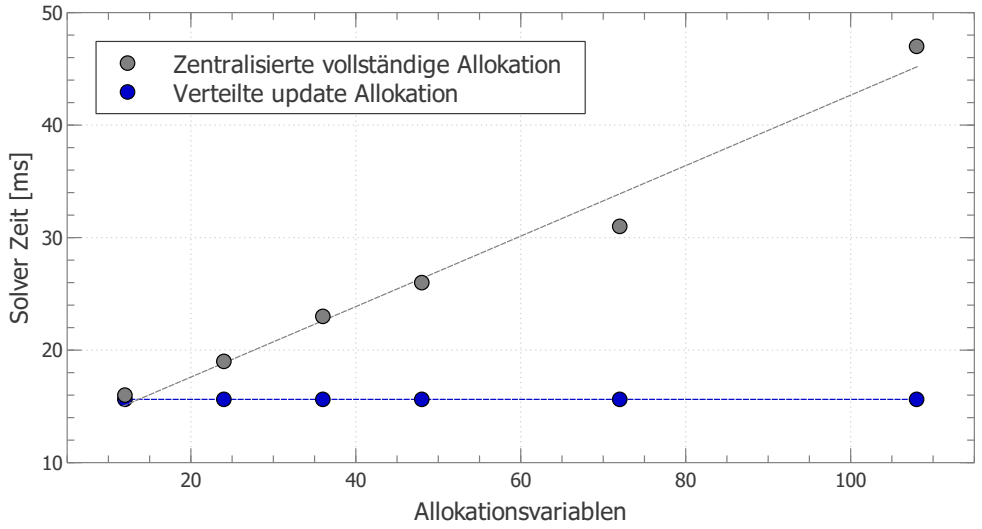
**Aufbau des Szenarios**

Ausgehend vom Basisszenario mit hierarchischen Policies und Clustern wurden die folgenden Variationen vorgenommen. Die Anzahl der Kameras (die indirekt der Anzahl der Funktionen entspricht), die Anzahl der Kräne (die der Anzahl der Edge-Geräte am Kran entspricht) und die Anzahl der Edge Server wurden in drei Stufen variiert, wie in Tabelle 8.6 dargestellt.

Tabelle 8.6: Aufbau Variationsszenarien

| **Teilnehmer** | **Szenario Wenig** | **Szenario Mittel** | **Szenario Viel** |
|---|---|---|---|
| Kamera Task | 3 | 6 | 9 |
| Edge Kran Nodes | 2 | 4 | 6 |
| Edge Server | 1 | 2 | 3 |

Abbildung 8.8: Anwendung 1 - Solver Zeit Skalierbarkeit




### Messungen

Daraus ergeben sich 27 gemessene Szenarien, die zur besseren Veranschaulichung durch Zusammenlegung von Edge Kran Nodes und Edge Server Nodes für die Auswertung zusammengefasst werden. Mit der unterschiedlichen Anzahl von Tasks, Nodes, Variablen und Constraints bieten die Messungen eine aussagekräftige Basis für die Analyse der Robustheit und Skalierbarkeit. Es wurden Szenarien mit 12 bis 108 Entscheidungsvariablen betrachtet.

### Analyse

Bei der Erhöhung der Anzahl der Kameratasks ohne Änderung der Anzahl der Edge Nodes sind zwei Sachverhalte auffällig (Szenarien 1-3). Für 9 Kameratasks wird keine Lösung gefunden, weil es zu wenige Edge Nodes gibt. Bei 6 Kameratasks wurde die erste Stufe der Zuweisung zuerst ausgeführt. D.h. die verteilte Update-Zuordnung unter Berücksichtigung der bisherigen Zuordnung der ersten 3 Kameraaufgaben und nur für die neu hinzugekommenen 3 Kameraaufgaben wird die Zuordnung berechnet. Für diesen Fall wird keine Lösung gefunden, womit die nächste Stufe der Allokation durchgeführt wird, nämlich die dezentrale Allokation mit allen Tasks und Nodes in einem Cluster. In diesem Fall wird eine Zuordnung gefunden, die einen Vorteil des gestuften Ressourcenmanagements zeigt.

In den nächsten drei Szenarien wurde die Anzahl von Nodes erhöht. Bei der Berechnung der vollständigen Allokation ist ein annähernd linearer Anstieg des Zeitverhaltens des Solvers in Abhängigkeit von der Anzahl der Tasks und Nodes im Netzwerk zu beobachten. Im Gegensatz dazu bleibt bei der update Allokation der veränderten Teilnehmer das Zeitverhalten des Solvers konstant.

Bei den letzten 3 Szenarien, bei denen die Anzahl der Nodes nochmals erhöht wird, ist dieses Verhalten ebenfalls zu beobachten. Der Anstieg des Zeitverhaltens zeigt, dass es linear mit der Anzahl der Nodes in einer vollständigen Allokation skaliert. Das ist ein vertretbares Verhalten, da in das entwickelte mehrstufige Ressourcenmanagement Mechanismen eingebaut sind, die das Zeitverhalten bei größeren Netzwerken reduzieren. Zunächst erfolgt die verteilte Allokation anhand der geänderten Tasks und Nodes, was die primäre Allokation für On-Demand Ereignisse. Dabei bleibt das Zeitverhalten des Solvers konstant, zum Beispiel bei 16 ms in diesem Anwendungsfall. Bei größeren Netzen wird die dezentrale Zuweisung verwendet, bei der die Zuweisung innerhalb von Clustern berechnet wird, wodurch sich das Zuweisungsproblem und damit der Zeitaufwand für den Solver verringert.

Darüber hinaus ist zu berücksichtigen, dass die Kosten bei einer allgemeinen Optimierung und bei komponentenspezifischen Policies in einem ähnlichen Bereich liegen. Der Vorteil der komponentenspezifischen Policies ist jedoch, dass die Allokation situationsabhängig ist und nicht nur eine generelle Optimierung betrachtet wird, die z.B. individuelle Anforderungen von Nodes ignoriert.

**Zusammenfassung - Variationsszenarien**

Insgesamt konnte die Robustheit und Skalierbarkeit der Allokationsmethode anhand der verschiedenen Szenarien mit unterschiedlicher Anzahl von Tasks und Nodes validiert werden. Dabei zeigten sich die Vorteile der gestuften Allokation, die zunächst nur die geänderten Teilnehmer einbezieht. Die Skalierung des Zeitverhaltens hängt bei der vollständigen Allokation linear von der Anzahl der Teilnehmer am Allokationsproblem ab, bei der verteilten update Allokation bleibt das Zeitverhalten konstant.

### 8.1.5 Bewertung und Diskussion

Anhand des Anwendungsfalls der Funktionsverlagerung bei kollaborativen Arbeitsmaschinen wurden die Aspekte der Machbarkeit der Allokationsmethode, der abgestuften komponentenspezifischen Policies, der Reaktion auf On-Demand Ereignissen und der Robustheit bewertet. Die folgenden Punkte wurden dabei bestätigt und spiegeln die Stärken des Konzepts wider.

Die Machbarkeit und Komplexität der Allokationsmethode wurde in einem vereinfachten Szenario demonstriert. In diesem Fall ergeben sich 64 Allokationsvarianten aus 12 Allokationsvariablen und bereits durch die Verwendung von Policy Constraints konnte das

Problem auf 19 machbare Allokationslösungen reduziert werden. Dadurch wird sowohl die Komplexität des Entscheidungsproblems als auch der Vorteil von Policy Constraints bei der Ermittlung realisierbarer Lösungen deutlich.

Die komponentenspezifischen Policies konnten stufenweise eingesetzt werden und haben die gesamte Latenz verbessert. Gleichzeitig konnte die Allokation entwurfskonform und für die jeweilige Situation optimiert werden, und zwar nur für die Komponenten, für die ein Bedarf besteht.

Die Allokation nach On-Demand Ereignissen, d. h. nach Änderungen im Netzwerk, profitiert von dem mehrstufigen Managementansatz, der nur die geänderten Tasks und Nodes in der ersten verteilten Update Allokation zuordnet. Das Zeitverhalten des Solvers blieb in diesem Fall konstant.

Die Robustheit der Allokationsmethode wurde in verschiedenen Szenarien mit 12 bis 108 Allokationsvariablen durch systematische Erhöhung der Task- und Teilnehmerzahl im Netzwerk validiert. Die Skalierbarkeit wird durch die Verwendung des mehrstufigen Managementansatzes erreicht, da das Zeitverhalten des Solvers trotz der erhöhten Anzahl von Komponenten konstant bleibt, indem die Allokation zunächst für die veränderten Komponenten berechnet wird.

Die Evaluation hat zudem gezeigt, dass folgende Punkte berücksichtigt werden müssen

Beim komponentenspezifischen Policy-Ansatz werden die situations- und bedarfsorientierten Allokationslösungen berechnet, die beim direkten Vergleich einzelner Kriterien wie der Gesamtlatenz nicht das globale Optimum darstellen. Wie im Konzept beschrieben, wird jedoch nicht ein Optimum gesucht, sondern eine machbare Lösung, die alle Constraints und Optimierungsbedürfnisse in Form von Policies erfüllt.

Das Zeitverhalten des Solvers ist bei einer zentralen vollständigen Allokation mit Berücksichtigung aller Komponenten höher als bei der verteilten Update Allokation, bei der nur die geänderten Komponenten berücksichtigt werden. Die Skalierung des Zeitverhaltens hängt beim Allokationsproblem mit vollständiger zentraler Allokation linear von der Anzahl der Komponenten ab, bleibt aber bei der verteilten Update Allokation konstant.

Aus den durchgeführten Messungen und Auswertungen lässt sich ableiten, dass die in dieser Arbeit entwickelte Allokationsmethode in Edge Computing Szenarien unterschiedlicher Größe eingesetzt werden kann. Die Skalierung des Konzepts profitiert von der mehrstufigen Allokationsberechnung. Insbesondere ist die Berechnung der Allokation nur für die geänderten Teilnehmer vorteilhaft, wie für die On-Demand Szenarien gezeigt wurde.

## 8.2 Anwendung 2 - Offloading im Kontext eines intelligenten Werksgeländes mit vernetzten Mikromobilen

### 8.2.1 Kontext und Anwendungsfall

Vernetzte Mikromobile gewinnen in automatisierten Umgebungen wie intelligenten Fabriken und Werksgeländen oder Logistikzentren zunehmend an Bedeutung. Für solche fahrerlosen Fahrzeuge (engl.: automated guided vehicles - AGVs) sind ein entsprechendes Rechensystem und eine Vernetzung unabdingbar.

**Systemkontext**

Der Systemkontext ist ein intelligentes Werksgelände, mit drei Arten von Mikromobilen, die autonom fahren. Hierzu gehören Indoor-AGVs (Automated Guided Vehicles) für logistische Aufgaben, Outdoor-AGVs für den Transport im Freien und fahrerlose Cargo-Bikes für den Personen- und Warentransport mit ihren jeweiligen Edge Devices. Darüber hinaus befinden sich in der Infrastruktur des Werksgeländes auch Recheneinheiten sowie weitere Sensoren. Dabei gibt es verschiedene Funktionen, die entweder auf den Rechenplattformen der Mikromobile oder in den Recheneinheiten der Hallen bzw. Infrastruktur ausgeführt werden. Die verschiedenen vernetzten Recheneinheiten bilden das Edge-Computing-Netzwerk für den Anwendungsfall.

Wie bei der ersten Anwendung sind mehrere Teilnehmer beteiligt, und ihre Anzahl sowie ihr Funktionsumfang können sich im Laufe der Zeit ändern. Daher wird diese Anwendung als repräsentatives Beispiel für ein Edge-Computing-System mit wechselnden Teilnehmern und Komponenten für die Evaluation der Allokationsmethode verwendet.

**Anwendungsfall**

Das Offloading von Funktionen zwischen beweglichen Agenten und Infrastruktur wird im Kontext des Werksgeländes untersucht. Jedes Mikromobil ist ein Agent und muss eine Kette von Funktionen ausführen, wie in Abbildung 8.9 dargestellt. Ausgehend von den Sensoren wird die Umgebung erfasst, in diesem Fall eine kamerabasierte Objekterkennung und -verfolgung. Zusammen mit anderen Informationen werden die Eingaben für die Routen- und Trajektorienplanung ermittelt und zur Berechnung der Fahrbefehle weitergegeben. Schließlich werden diese an die Fahrzeugsteuerung weitergegeben und ausgeführt.

Für die Evaluation der Allokationsmethode wurden die Umgebungserfassungsfunktionen für das Offloading genauer betrachtet. Diese Funktionen wurden ausgewählt, weil sie daten- und rechenintensiv sind, was bedeutet, dass die Ausführung auf verschiedenen Rechnerplattformen unterschiedlich performant ist. Darüber hinaus können durch die geschlossene Umgebung eines Werksgeländes verschiedene Beschränkungen festgelegt werden. So

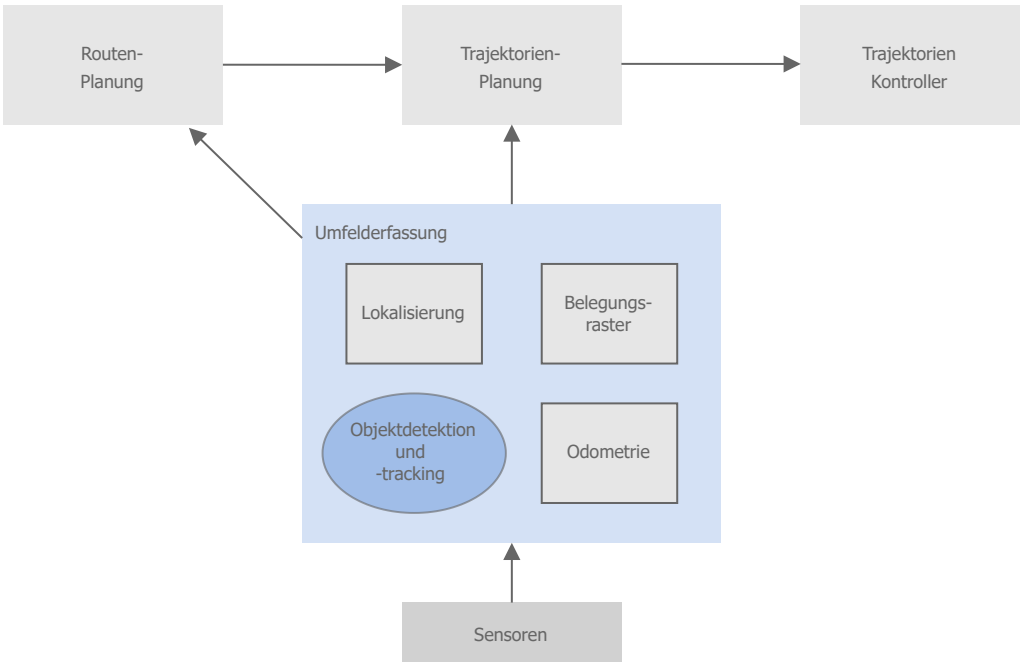



Abbildung 8.9: Funktionskette der autonomen Mikromobile

Tabelle 8.7: Edge Devices der Mikromobile und Infrastruktur

| **Teilnehmer** | **Anzahl im Basisszenario** | **Edge Device** |
|---|---|---|
| AGV-indoor | 3 | Nano |
| AGV-outdoor | 3 | NX |
| AGV-Lastenfahrrad | 2 | TX2 |
| Infrastuktur | 1 | AGX + 3xU96 |

können beispielsweise niedrigere Geschwindigkeiten als bei einem autonomen Fahrzeug im Straßenverkehr festgelegt werden, und die Konnektivität für das Offloading kann in dieser Umgebung kontrolliert werden. Für die Objekterkennung und -verfolgung stehen verschiedene Edge-Devices in den Mikromobilen sowie Edge-Devices in der Infrastruktur zur Verfügung, wie in Tabelle 8.7 aufgeführt.

Für die on-demand Reallokation wird der Fall betrachtet, dass Mikromobile beauftragt werden und somit neue Agenten und deren Funktionen zum Edge-Netzwerk hinzugefügt werden. Dann berechnet die vorgestellte Allokationsmethode basierend auf der Policy Beschreibung die Entscheidung, ob die Objekterkennungsfunktion auf den Edge Einheiten der Mikromobile oder in der Infrastruktur ausgeführt werden soll.

### 8.2.2 Aufbau und Vorgehen

Als Evaluationsgegenstand wird die Reallokation von Objekterkennungsfunktionen der Mikromobile untersucht. Zu diesem Zweck wurden die folgenden Dimensionen untersucht:

- Die Machbarkeit der Modellierung und Berechnung der Allokation.
- Die Vorteile der komponentenspezifische Policies
- Das Verhalten bei On-Demand Ereignisse in Form von zusätzlichen AGVs
- Die Robustheit und Skalierbarkeit des Ansatzes

Folgende Evaluationskriterien wurden zu diesem Zweck analysiert:

- Die Lösbarkeit (engl. Satisfiabilty) des Problems für die Analyse der Machbarkeit.
- Das Zeitverhalten der Berechnung für die Analyse der Skalierbarkeit
- Die Gesamtkosten für die Analyse der Anforderungen und Ereignisse

Die folgenden Informationen wurden für die Evaluation verwendet: Wie bei der ersten Anwendung wurden Messungen der Objekterkennungsfunktionen auf verschiedenen Hardwareplattformen durchgeführt, um die Policy Werte zu ermitteln. Für die drahtlose Kommunikation wurde WLAN (Wi-Fi 6) verwendet und die Kommunikationskosten in die Allokationsberechnung integriert.

Weiterhin wurden statische Offloading Strategien als Baseline einbezogen, die eine manuelle Allokation darstellen sollen. Auch wurde die Allokation mit und ohne Optimierung berechnet, um die Latenzzeiten und Energiekosten zu untersuchen.

**Vorgehen**

Zunächst wurde ein ***Basisszenario*** definiert, an dem die Machbarkeit des Ansatzes, bestehend aus Policy-Modellierung, Extraktion von Informationen und Allokationsberechnung in einem überschaubaren Rahmen, validiert wurde. Darüber hinaus wurden die multikriteriellen Lösungsvarianten und der Tradeoff zwischen Latenz und Energie analysiert. Das Basisszenario besteht aus 3 Mikromobilen jedes Typs und einem Infrastruktur-Device, wie in Tabelle 8.7 aufgelistet.

Als nächstes wurde in den ***komponentenspezifischen Policy Szenarien*** der Einfluss der Policy Beschreibung auf die Allokation untersucht. Ausgehend vom Basisszenario wurde für die einzelnen Fahrzeugtypen die Allokation mit verschiedenen Policies berechnet und mit der Allokation mit globalen Policies verglichen.

Im nächsten Schritt wurde das Hinzufügen von Mikromobilen als ***On-Demand-Szenarien*** durchgeführt und ausgewertet. Ausgehend vom Basisszenario wird ein Mikromobil pro

Fahrzeugtyp hinzugefügt und die Allokation einmalig für das zusätzliche Fahrzeug und einmal für das gesamte Edge Netzwerk berechnet und verglichen.

Als abschließende Untersuchung wurde die Anzahl der Mikromobile in den ***Variationsszenarien*** systematisch erhöht, um die Robustheit und Skalierbarkeit des Ansatzes mit 18 Szenarien und bis zu 120 Mikromobilen zu untersuchen.

### 8.2.3 Modellierung und Policy Beschreibung

Im Folgenden wird die zur Beschreibung des Basisszenarios verwendete Policy Modellierung gezeigt.

**Systemmodellierung**

Das System wurde analog zur ersten Anwendung in verschiedenen Ebenen modelliert.

Die logische Funktionsarchitektur in Abbildung 8.10. Die Softwarearchitektur in Abbildung 8.11 und die Hardwarearchitektur in Abbildung 8.12.

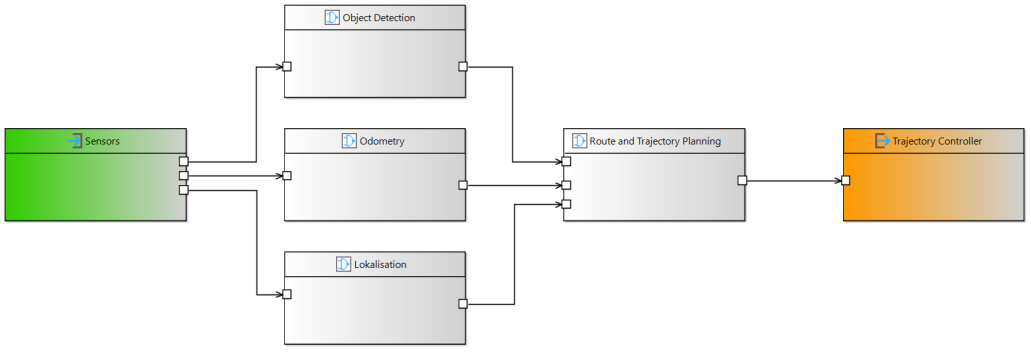



Abbildung 8.10: Anwendungsfall 2 - Logische Funktionsebene

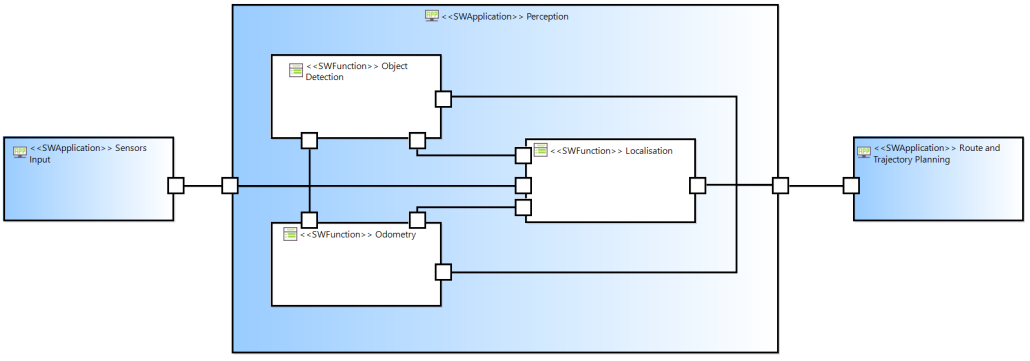



Abbildung 8.11: Anwendungsfall 2 - Software Ebene

**Policy Modellierung**

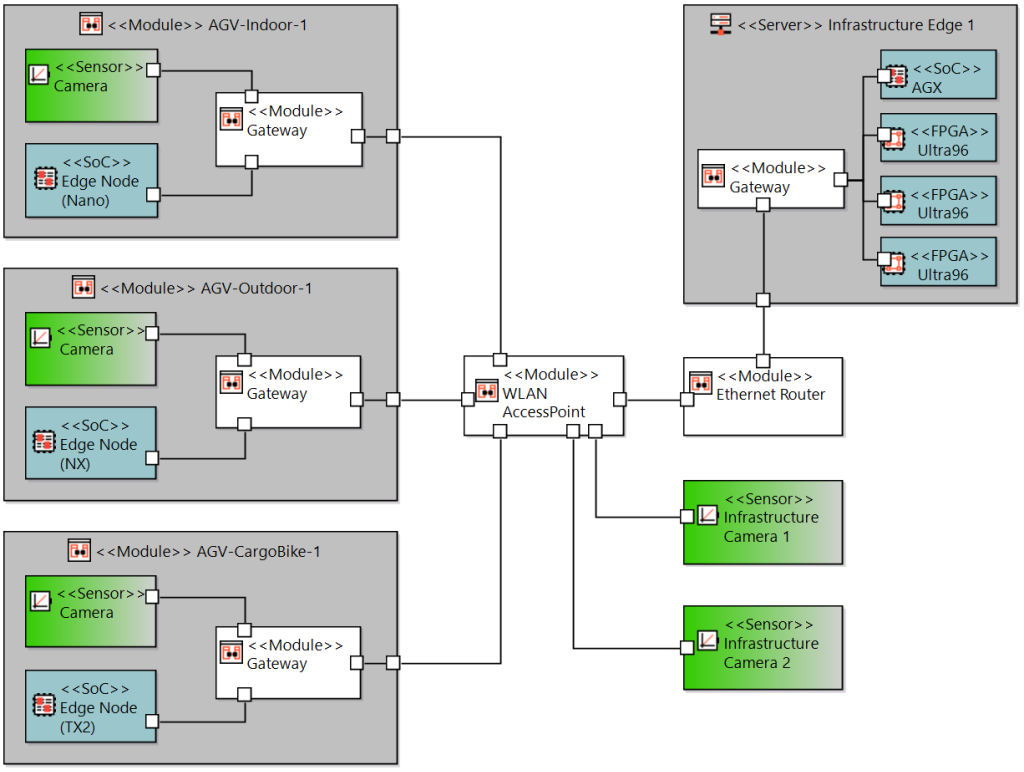



Abbildung 8.12: Anwendungsfall 2 - Hardware Ebene

Die Modellierung der Policies im Entwicklungswerkzeug erfolgt auf die gleiche Weise wie in der ersten Anwendung.

Die verwendeten Policy Werte wurden in Messungen auf verschiedenen Recheneinheiten ermittelt und sind in Tabelle 8.9 zusammengefasst.

### 8.2.4 Umsetzung und Ergebnisse

Nach der Modellierung der Policies wurden die allokationsrelevanten Daten aus dem Modell extrahiert und zur Ausführung des Constraint Programming Solvers weitergegeben. Die hier gemessenen Werte wurden in einem System ausgeführt und sind daher in sich vergleichbar. Untersucht wurden die Aspekte der Machbarkeit, der komponentenspezifischen Policies, der On-Demand-Ereignisse und der Skalierbarkeit des Konzepts.

#### 8.2.4.1 Evaluation Basisszenario

**Aufbau des Szenarios**

Für das Basisszenario wurde ein Aufbau mit 3 Mikromobilen pro Fahrzeugtyp und 1 Infrastruktur Edge Device verwendet, wie in Tabelle 8.8 beschrieben, sowie die Policy-Werte aus Tabelle 8.9. Für die Bewertung der Durchführbarkeit wurden entweder nur die Policy

Tabelle 8.8: Input Information Auszug

| **Komponente** | **Policy** | **Art** | **Wert** | **Einheit** |
|---|---|---|---|---|
| AGV-Indoor (Nano) | Cost | Yolo-Latenz | 0.0588 | s |
| AGV-Indoor (Nano) | Cost | Yolo-Energie | 0.24706 | Ws |
| AGV-Indoor (Nano) | Limit | Instanzen | 1 | - |
| ObjectDetection-Indoor | Limit | Latenz | 0.061 | s |
| AGV-Outdoor (NX) | Cost | Yolo-Latenz | 0.0303 | s |
| AGV-Outdoor (NX) | Cost | Yolo-Energie | 0.29697 | Ws |
| AGV-Outdoor (NX) | Limit | Instanzen | 1 | - |
| ObjectDetection-Outdoor | Limit | Latenz | 0.043 | s |
| AGV-CargoBike (TX2) | Cost | Yolo-Latenz | 0.0526 | s |
| AGV-CargoBike (TX2) | Cost | Yolo-Energie | 0.36316 | Ws |
| AGV-CargoBike (TX2) | Limit | Instanzen | 1 | - |
| ObjectDetection-CargoBike | Limit | Latenz | 0.061 | s |
| Infrastructure a (AGX) | Cost | Yolo-Latenz | 0.0313 | s |
| Infrastructure a (AGX) | Cost | Yolo-Energie | 0.23438 | Ws |
| Infrastructure a (AGX) | Limit | Instanzen | 4 | - |
| Infrastructure b (U96) | Limit | Yolo-Latenz | 0.0204 | s |
| Infrastructure b (U96) | Limit | Yolo-Energie | 0.16122 | Ws |
| Infrastructure b (U96) | Limit | Instanzen | 1 | - |

Tabelle 8.9: Policies Auszug

| **Komponente** | **Policy** | **Art** | **Cost Expression** | **Limit Expression** |
|---|---|---|---|---|
| AGV-Indoor (Nano) | Constraint | *leq* | Yolo-Instanzen | Instanzen |
| AGV-Outdoor (NX) | Constraint | *leq* | Yolo-Instanzen | Instanzen |
| AGV-CargoBike (TX2) | Constraint | *leq* | Yolo-Instanzen | Instanzen |
| Infrastructure a (AGX) | Constraint | *leq* | Yolo-Instanzen | Instanzen |
| Infrastructure b (U96) | Constraint | *leq* | Yolo-Instanzen | Instanzen |
| ObjectDetection-Indoor | Constraint | *leq* | Yolo-Latenz | Latenz |
| ObjectDetection-Outdoor | Constraint | *leq* | Yolo-Latenz | Latenz |
| ObjectDetection-CargoBike | Constraint | *leq* | Yolo-Latenz | Latenz |

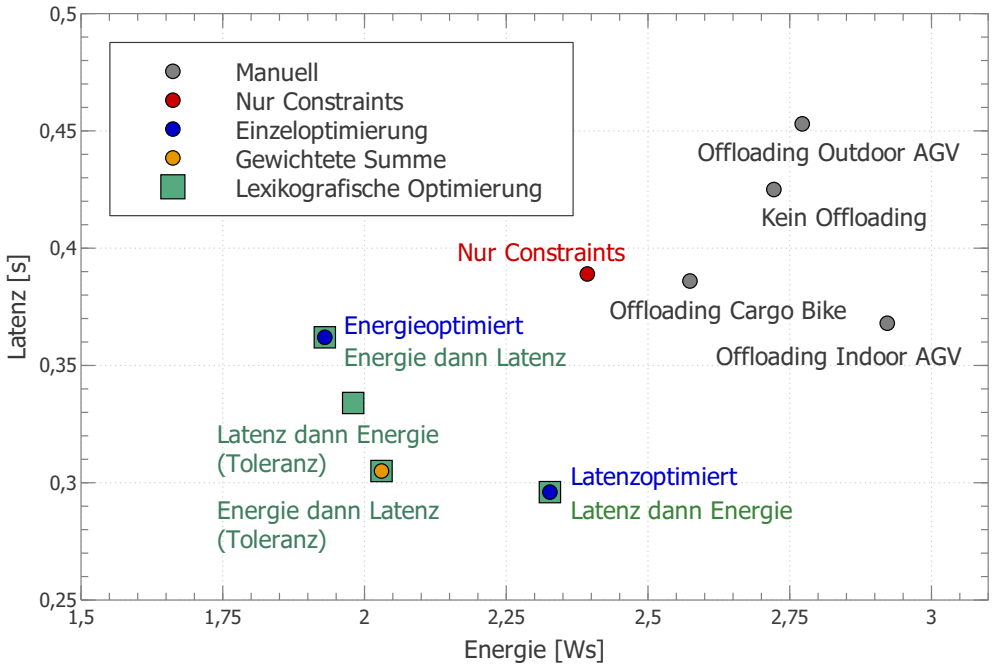



Abbildung 8.13: Basisszenario – Latenz vs. Energie Allokationsergebnisse

Constraints oder nur eine Optimierung, nämlich die Gesamtoptimierung von Latenzkosten oder Energiekosten, betrachtet. Anschließend wurde eine Analyse der verschiedenen multikriteriellen Lösungsansätze durchgeführt.

**Messungen**

Um die Auswirkungen der verschiedenen Lösungsansätze des Konzepts zu analysieren, wurden getrennte Messungen durchgeführt. Obwohl das Basisszenario relativ klein ist, ergeben sich aus der Kombination von 9 Aufgaben der Mikromobile und 5 möglichen Edge Devices über 1,9 Millionen ($5^9$) Allokationsmöglichkeiten. In diesem Szenario wurde die Machbarkeit der Modellierung und Berechnung validiert. Zusätzlich wird deutlich, dass bei großen Szenarien mit zunehmender Komplexität ein manuelles Management nicht praktikabel ist. Für das Szenario wurden verschiedene Berechnungsvarianten für das multikriterielle Problem mit Energie und Latenzzeit durchgeführt. Abbildung 8.13 zeigt die Gesamtergebnisse und Abbildung 8.14 zeigt die durchschnittlich benötigte Rechenzeit. Für eine kombinierte Analyse wurde ein Trade-off Wert gebildet, der sich aus den normalisierten gemittelten Energie- und Latenzwerten zusammensetzt. Der Trade-off Wert zeigt somit eine Gesamtminimierung sowohl der Energie als auch der Latenzzeit.

**Analyse**

Zuerst wurde die Berechnung nur auf der Grundlage von Policy Constraints (rot) durchgeführt, was die kürzeste Berechnungszeit zur Ermittlung einer machbaren Lösung zeigt. In diesem Fall ist die Lösung im Vergleich zur manuellen Zuteilung (grau) sogar besser,

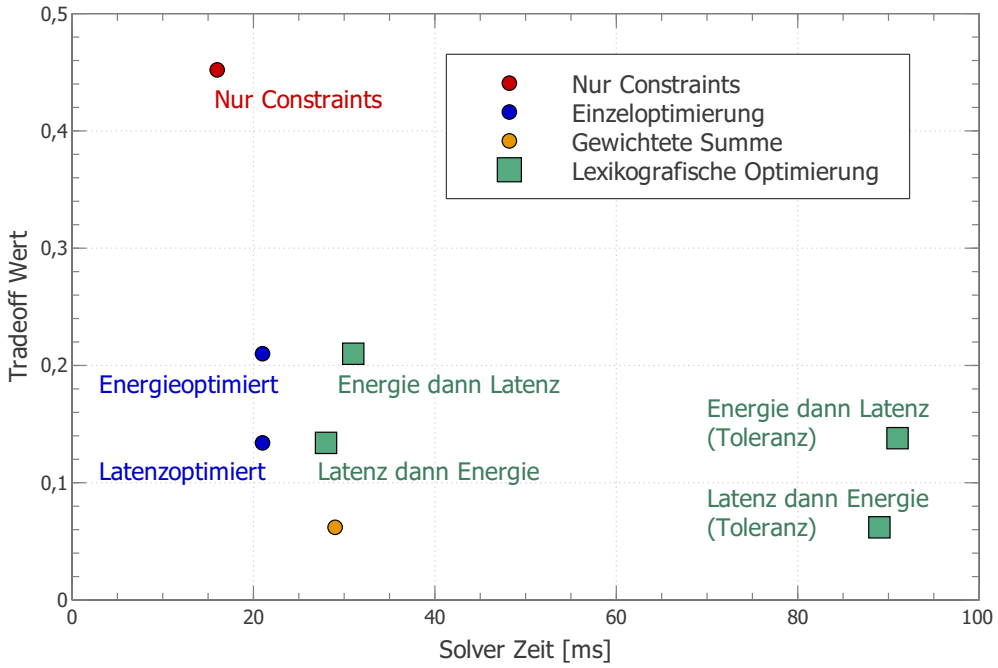



Abbildung 8.14: Basisszenario – Tradeoff Wert (Kombination von Latenz und Energie) vs Zeit Performance Allokationsergebnisse

entweder in Bezug auf die Latenzzeit oder den Energieverbrauch. Bei der manuellen Zuweisung wird eine feste Offloading Strategie gewählt, d. h. es wird jeweils nur ein Typ von Mikromobilen zum Offloading angehalten oder es findet kein Offloading statt. Die Einzeloptimierungen (blau) zeigen die getrennten Minimalwerte. Dabei ist die Rechenzeit des Solvers im Vergleich zur reinen Constraint Zuweisung (rot) leicht gestiegen.

Als Nächstes wurde die Optimierung mit gewichteter Summe (orange) durchgeführt, wodurch die Berechnungszeit ebenfalls leicht anstieg, allerdings wurde der geringste Tradeoff Wert zwischen Energie und Latenz gefunden. Als nächstes wurde die lexikographische Optimierung (grün) mit und ohne relative Toleranzen (+-15) durchgeführt. Die Zuordnung mit relativen Toleranzen (grün - Toleranz) erhöht die Rechenzeit erheblich (ca. 3x). Dies könnte auf die zusätzlichen kontinuierlichen Variablen zurückzuführen sein, die dem Allokationsproblem hinzugefügt werden, was die Komplexität deutlich erhöht.

Beim Vergleich von Energie und Latenzzeit zeigt sich, dass bei der lexikographischen Optimierung die Reihenfolge wichtig ist, da die zweite zusätzliche Optimierung keine große Veränderung bewirkt. Das Ergebnis ist daher vergleichbar mit der Variante, bei der nur die Energie oder die Latenzzeit optimiert wurde. Anders verhält es sich bei der Berechnung mit relativen Toleranzen, wo der Trade-off Wert in der Summe niedriger ist.

Betrachtet man die Energie vs. Latenz (siehe Abbildung 8.13) sowie die Zeitleistung des Solvers (siehe Abbildung 8.14) gemeinsam, so ist zu erkennen, dass mit allen Berechnungsvarianten eine machbare Allokation gefunden und verbessert werden kann. Selbst

die Variante mit nur Constraints (rot) bietet Vorteile im Vergleich zu einer statischen Offloading-Strategie (grau). Damit wurde die Machbarkeit des Ansatzes validiert. Jede multikriterielle Berechnungsvariante bietet Vor- und Nachteile, ihr Einsatz ist abhängig vom Anwendungsfall und der Konfiguration des Edge Systems. Generell kann festgestellt werden, dass die Berechnung, die nur auf Policy Constraints (rot) basiert, eine machbare Lösung in der kürzesten Zeit finden kann.

Bei der Berücksichtigung von Policy-Optimierungen ist die Variante mit gewichteten Summen zu bevorzugen, da keine signifikante zusätzliche Rechenzeit benötigt wird. Die lexikographische Optimierung ohne relative Toleranzen zeigt bei kleinen Systemen keine Verbesserung, was sich aber bei großen Systemen ändern kann. Die Einbeziehung relativer Toleranzen verbessert die Ergebnisse, erhöht aber die Lösungszeit. Aus diesem Grund sollte die lexikografische Optimierung nur in Verbindung mit einem Timeout und zusammen mit dem gestuften Managementansatz verwendet werden, und zwar eher auf einer höheren Ebene oder wenn eine zusätzliche Optimierung erforderlich ist.

**Zusammenfassung - Basisszenario**

Die Durchführbarkeit des Ansatzes wurde hiermit in einer zweiten Anwendung evaluiert und die verschiedenen multikriteriellen Optimierungsoptionen wurden analysiert. Die Allokationsmethode mit nur Policy Constraints findet günstigere Allokationen im Vergleich zu einer statsichen Offloading-Strategie. Die multikriterielle Optimierung mittels gewichteter Summe zeigt das geringste Zeitverhalten, während die lexikographische Optimierung mit Toleranzbereichen das höchste Zeitverhalten zeigt. Die Verwendung von Policy Optimierungen erhöht generell die Komplexität des Problems und verbessert gleichzeitig die Gesamtkosten.

#### 8.2.4.2 Evaluation komponentenspezifische Policies Szenarien

Aufbauend auf dem Basisszenario werden zwei Aspekte untersucht. Die zusätzlichen komponentenspezifischen Policies und die multikriterielle Optimierung mit hierarchischen Bedarfen.

Tabelle 8.10: Komponentenspezifische Policies – Input Information Auszug

| Komponente | Policy | Art | Wert | Einheit |
|---|---|---|---|---|
| ObjectDetection-Indoor-2 | Limit | Latenz | 0.025 | s |
| ObjectDetection-CargoBike-3 | Limit | Latenz | 0.043 | s |

**Messungen**

Für die Evaluation wurden die Ergebnisse des Basisszenarios mit gewichteter Summe mit den zusätzlichen Policy Constraints und zusätzlichen Policy Optimierungen verglichen.

Tabelle 8.11: Komponentenspezifische Policies – Policies Auszug

| **Komponente** | **Policy** | **Art** | **Cost Expression** | **Limit Expression** |
|---|---|---|---|---|
| AGV-Indoor-3 (Nano) | Optimierung | min | Yolo-Energie | - |
| AGV-Outdoor-3 (NX) | Optimierung | min | Yolo-Energie | - |
| AGV-CargoBike-2 (TX2) | Optimierung | min | Yolo-Latenz | - |
| Infrastructure a (AGX) | Optimierung | min | Yolo-Instanzen | - |
| ObjectDetection-Indoor-2 | Constraint | *leq* | Yolo-Latenz | Latenz |
| ObjectDetection-CargoBike-3 | Constraint | *leq* | Yolo-Latenz | Latenz |

Bei der Verwendung der Reihenfolge der hierarchischen Bedarfe wird eine lexikographische Optimierung mit und ohne Toleranzbereiche durchgeführt. Das heißt, zuerst die Komponenten Constraints, gefolgt von den globalen Optimierungen und dann die Komponenten Optimierungen. Die Ergebnisse der Messungen sind in den Abbildungen 8.15 und 8.16 dargestellt.

**Analyse**

Zunächst wurden die zusätzlichen komponentenspezifischen Constraint Policies (blau - constraints) zur Allokation hinzugefügt. Das Zeitverhalten des Solvers änderte nicht im Vergleich zum vorherigen Szenario. Die Werte für die Gesamtlatenz, die Energie und den Tradeoff änderten sich aufgrund der zusätzlichen Constraints. Dies spricht für die Allokationsmethode, die primär auf der Constraint Beschreibung basiert und somit schnell eine machbare Allokation findet.

Als nächstes wurden die zusätzlichen komponentenspezifischen Policies zur Optimierung (blau – constraints und komponenten Optimierung) hinzugefügt und in der Variante mit zusätzlicher globaler Optimierung mit und ohne Toleranzbereiche verglichen (grün). Das Zeitverhalten des Solvers ist im Vergleich zu den Varianten ohne Policy Optimierung höher. Die globale Optimierung erhöht das Zeitverhalten nochmals. Da komponentenspezifische Optimierungen durchgeführt werden, ist dies zu Lasten eines schlechteren gesamten Tradeoff-Optimierungswertes.

Als letztes wurde die hierarchische Bedarfsreihenfolge aktiviert. D.h., dass zuerst die Constraint Policies bottom-up, dann die Optimierungspolicies (top-down) betrachtet werden. Während der lexikographischen Optimierung werden also zuerst die globalen Optimierungen durchgeführt, gefolgt von den komponentenspezifischen Policies.

In der Gesamtbetrachtung von Energie vs. Latenz ist die Variante mit globaler Optimierung wie im Basisszenario genauso gut wie das Hinzufügen komponentenspezifischer

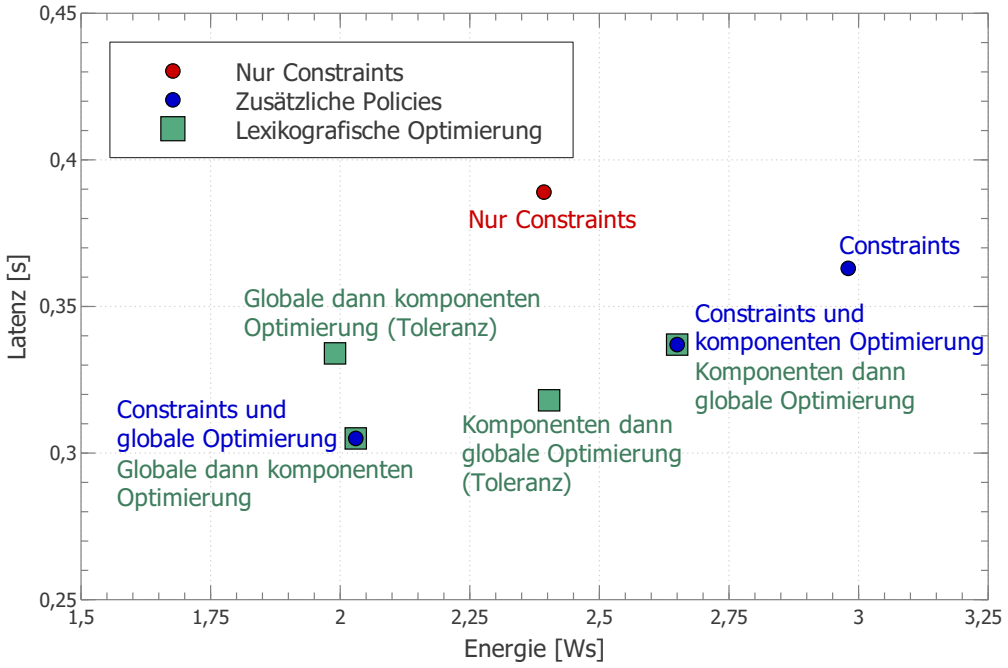



Abbildung 8.15: Komponentenspezifische Policies – Latenz vs. Energie Allokationsergebnisse

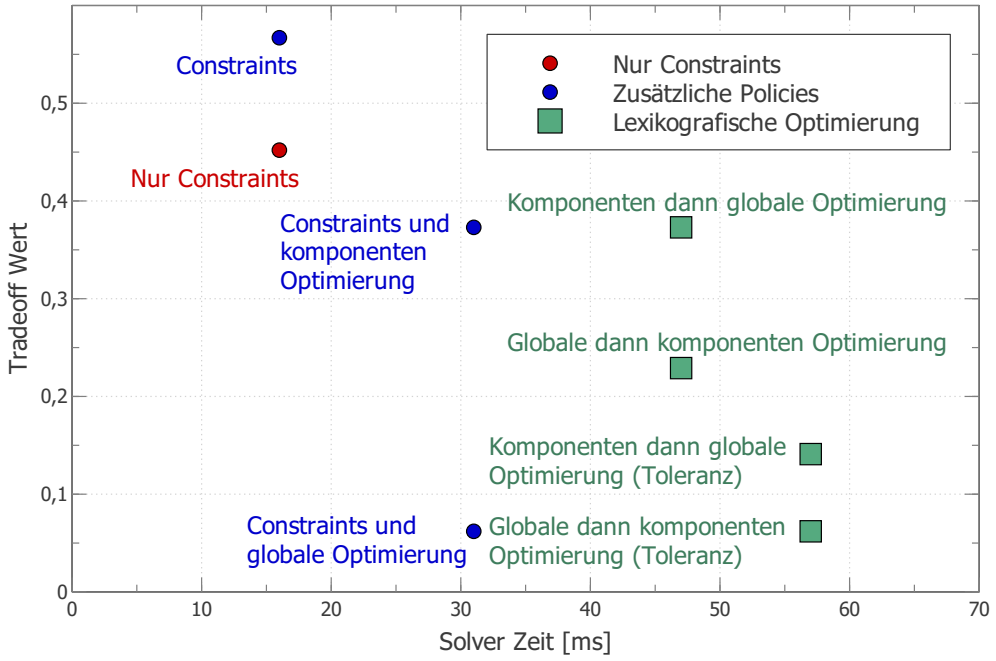



Abbildung 8.16: Komponentenspezifische Policies – Tradeoff Wert (Kombination von Latenz und Energie) vs Zeit Performance Allokationsergebnisse

Policies. Allerdings geht es nicht primär um die Optimierung der Gesamtwerte, sondern darum, dass jede Komponente eine bedarfsgerechte Allokation erhält.

Bei der zusätzlichen Betrachtung des Zeitverhaltens vs. Tradeoff Wert wird deutlich, dass die Varianten mit vielen lexikographischen Optimierungen und Toleranzbereichen mehr Zeit benötigen als die Varianten, die primär die Constraint Policies berücksichtigen. Diese können jedoch z.B. in eine erste vollständige Allokationsberechnung oder in den stufenweisen Management Ressourcen Ansatz einbezogen werden. Im ersten Schritt wird in kürzester Zeit eine machbare Allokation unter Berücksichtigung der Constraint Policies berechnet, und wenn das System stabil ist, kann eine optimierte Allokation berechnet und eine Reallokation vorgenommen werden.

**Zusammenfassung – Komponentenspezifische Policies Szenarien**

Die Machbarkeit und die Auswirkungen der komponentenspezifischen Policies konnten aufgezeigt werden. Es ist klar, dass die Optimierung nicht im Mittelpunkt der Allokation stehen sollte, da sie den Zeitaufwand für den Solver erheblich erhöht. Mit den komponentenspezifischen Policies können Allokationen berechnet werden, die für die Bedürfnisse der Komponenten optimiert werden können. In jedem Fall sollte der hierarchische und stufenweise Managementansatz verwendet werden, um die Vor- und Nachteile von Allokationen auf Basis von Constraints oder mit zusätzlichen Optimierungsstrategien auszugleichen.

#### 8.2.4.3 Evaluation On-Demand Szenarien

**Aufbau des Szenarios**

Ausgehend vom Basisszenario werden die Auswirkungen auf die Allokation untersucht, wenn Mikromobile hinzugefügt werden. Es wird ein Vergleich durchgeführt, wenn die Allokation nur für das zusätzliche AGV oder für das gesamte System berechnet wird.

**Messungen**

Die verteilte update Allokation und die zentralisierte vollständige Allokation wurden für das Hinzufügen eines Mikromobils pro Fahrzeugtyp gemessen. Die Ergebnisse sind in Abbildung 8.17 zu sehen.

**Analyse**

Mit dem Hinzufügen eines Mikromobils und einer vollständig zentralisierten Allokation ist in jedem Szenario ein Anstieg der Rechenzeit des Solvers zu beobachten. Die Ergebnisse bleiben ähnlich wie bei der verteilten Allokation, bei der nur die Änderungen im System berücksichtigt werden. Das Zeitverhalten des Solvers bleibt konstant (ca. 16 ms), was wiederum, wie im ersten Anwendungsfall, für den Einsatz des gestuften Managementansatzes spricht.

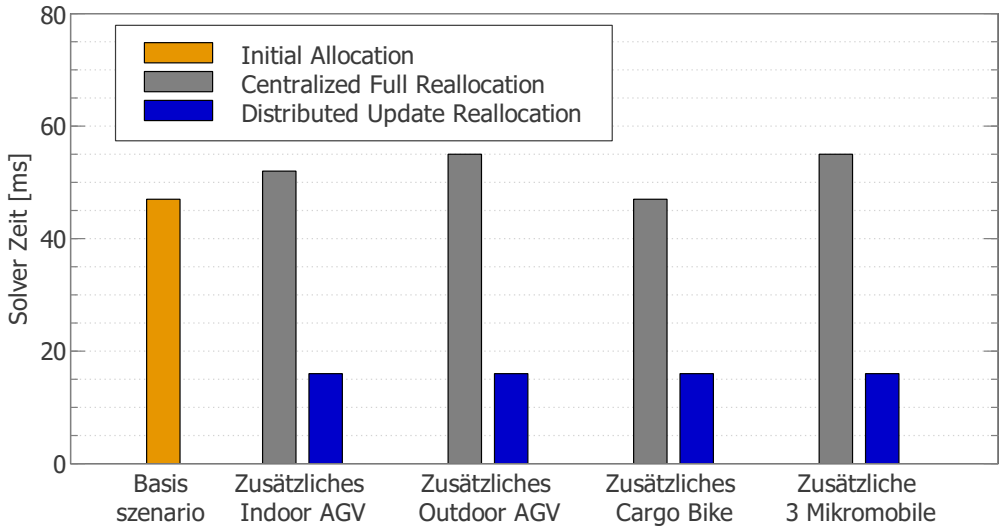



Abbildung 8.17: On-Demand Szenarien – Messungen

**Zusammenfassung – On-Demand Szenarien**

Das Verhalten der Allokationsmethode während On-Demand Ereignisse, in diesem Fall das Hinzufügen von Mikromobilen im System, wurde analysiert. Es wurde gezeigt, dass die Allokationsmethode von der vorher festgelegten Allokation und einem abgestuften Ressourcenmanagement profitiert, um in kurzer Zeit eine realisierbare Allokation zu finden.

#### 8.2.4.4 Evaluation Variationsszenarien

**Aufbau des Szenarios**

Anhand der On-Demand Szenarien wurden die folgenden Variationen vorgenommen, um die Skalierbarkeit des Ansatzes zu untersuchen. Die Anzahl der Mikromobile, die zusätzliche Aufgaben und EdgeDevices in das System bringen, und die Anzahl der EdgeNodes in der Infrastruktur wurden in drei Stufen variiert, wie in der Tabelle 8.12 zu sehen ist. Daraus ergeben sich (3 initial/vollständige/update x 3 wenig/mittel/viel x 2 komponentenspezifisch/global) 18 Szenarien mit über 120 Teilnehmern, mit denen die Robustheit und Skalierbarkeit des Ansatzes untersucht wurden.

**Messungen**

Die Varianten wurden jeweils mit und ohne globale Optimierung sowie mit zentralisierter vollständiger und verteilter Allokation durchgeführt, um die Auswirkungen auf das Zeitverhalten zu analysieren. Mit 10 bis über 120 Teilnehmern im Edge-Netz und 48 bis 3000 Allokationsvariablen bieten die gemessenen Szenarien eine gute Grundlage, um die Robustheit und Skalierbarkeit zu analysieren. Die Messergebnisse sind in den Abbildungen 8.18 und 8.19 zu sehen.

Tabelle 8.12: Variationsszenarien

| **Teilnehmer** | **Szenario Wenig** | **Szenario Mittel** | **Szenario Viel** |
|---|---|---|---|
| Indoor | 3 + 1 | 10 + 3 | 30 + 10 |
| Outdoor | 3 + 1 | 10 + 3 | 30 + 10 |
| Cargo Bike | 3 + 1 | 10 + 3 | 30 + 10 |
| Infrastruktur Nodes | 1 | 3 + 1 | 9 + 3 |

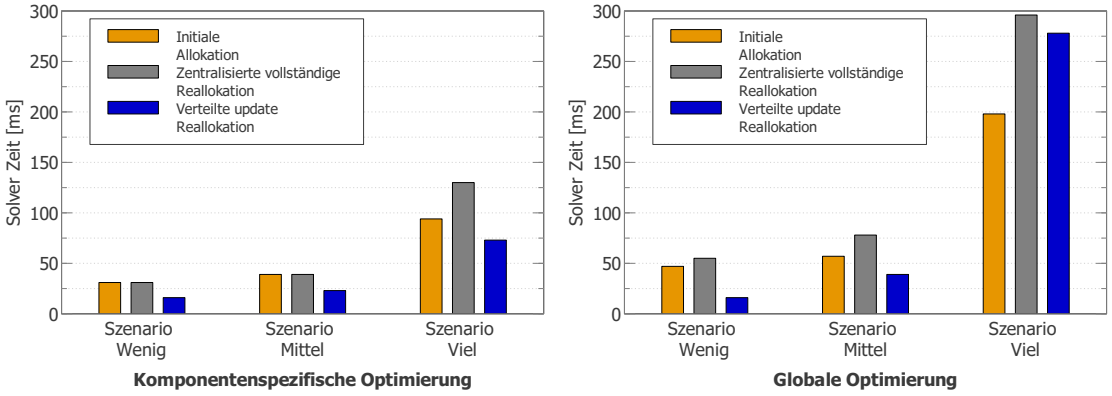



Abbildung 8.18: Variation Szenarien – Messungen

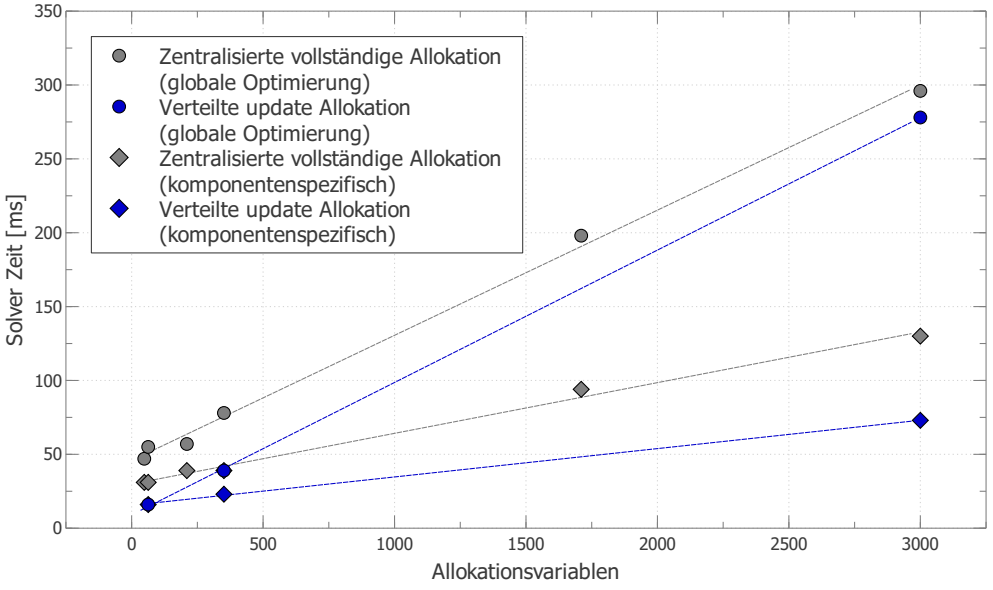



Abbildung 8.19: Skalierung der Rechenzeit des Allokationsverfahrens

**Analyse**

Betrachtet man zunächst die Varianten ohne zusätzliche Teilnehmer jeweils in wenig, mittel und viel (orange), so ist der Unterschied zwischen mit und ohne globale Optimierung deutlich sichtbar. Das Zeitverhalten des Solvers steigt bei der Variante mit komponentenspezifischen und mit globalen Optimierung linear an, bei der Variante mit globaler Optimierung ist die Steigung jedoch höher (graue Trendlinien), so dass bei mehr Teilnehmern bzw. Variablen im Allokationsproblem der Unterschied im Zeitverhalten größer wird als bei der komponentenspezifische Variante (blaue Trendlinien).

Beim Hinzufügen zusätzlicher Teilnehmer werden die zentralisierte vollständige Allokation (graue Säulen), die alle Teilnehmer berücksichtigt, und die verteilte update Allokation (blaue Säulen) analysiert. Die verteilte Allokation berücksichtigt die vorherige Zuweisung und berechnet daher die Reallokation nur für die geänderten Teilnehmer. Für jede Variante ist zu erkennen, dass die vollständige Allokation eine höhere Rechenzeit des Solvers aufweist als die verteilte update Allokation. Dies bestätigt die Messungen der ersten Anwendung und die Vorteile der stufenweisen Ressourcenverwaltung, die zunächst eine verteilte update Allokation vorsieht. Das Zeitverhalten dieser beiden Varianten steigt ebenfalls linear mit der Anzahl der Teilnehmer bzw. Variablen. Hierbei ist ebenfalls zu beachten, dass die Varianten mit globaler Optimierung eine höhere Steigung (grau Trendlinie) aufweisen, wodurch die Differenz zur komponetenspezifischen Variante (blaue Trendlinie) größer wird.

Interessant ist, dass bei wenigen Teilnehmern (wenig/mittel) die verteilte update Verteilung mit globaler Optimierung ein geringeres Zeitverhalten aufweist als die jeweilige Basisvariante ohne zusätzliche Teilnehmer, aber bei vielen Teilnehmern das Zeitverhalten (viel) deutlich höher ist als die Basisvariante und keinen großen Vorteil bietet. Dies bestätigt die Tatsache, dass globale Optimierungen bei dieser Allokationsmethode generell die Rechenzeit des Solvers verschlechtern und je größer das Allokationsproblem ist, die globalen Optimierungen vermieden werden sollten.

**Zusammenfassung - Variationsszenarien**

Insgesamt konnte mit den verschiedenen Szenarien mit unterschiedlichen Teilnehmerzahlen und Allokationsvariablen die Robustheit und Skalierbarkeit des Allokationsverfahrens evaluiert werden. Die Vorteile der stufenweisen Allokation, die sich bereits in den anderen Evaluationsszenarien und im anderen Anwendungsfall gezeigt haben, wurden für ein wesentlich größeres Edge System bestätigt. Die verteilte update Allokation ist bei Änderungen im System wie dem Hinzufügen von Mikromobilen in diesem Anwendungsfall vor einer zentralisierten vollständigen Allokation zu bevorzugen.

Es konnte auch bestätigt werden, dass das Zeitverhalten der Allokationsmethode für große Edge Systeme linear skaliert, was für den direkten Einsatz im Betrieb entscheidend ist.

Allerdings steigt die Rechenzeit bei globalen Optimierungen stärker an, so dass diese Variante für sehr große Edge Systeme nicht zu bevorzugen ist. Stattdessen können die konzipierten Mechanismen des Allokationsverfahrens genutzt werden, wie z. B. die dezentrale Allokation durch Clusterbildung, wodurch die Allokation in kleinere Allokationsprobleme aufgeteilt wird und in diesen bedarfsgerecht und on-demand auf globale Optimierungen zurückgegriffen werden kann.

### 8.2.5 Bewertung und Diskussion

Analysiert wurden die Aspekte Machbarkeit, multikriterielle Optimierungsvarianten, komponentenspezifische Bedarfe, On-Demand Ereignisse und Skalierbarkeit des Konzeptes. Die folgenden Punkte wurden bestätigt und spiegeln die Stärken des Konzepts wider.

Die Machbarkeit der Allokationsmethode, aber auch die Komplexität des Allokationsproblems wurde in einem überschaubaren Basisszenario mit 10 Edge-Teilnehmern demonstriert. Dabei ergaben sich ca. 1,9 Millionen verschiedene Allokationsmöglichkeiten. Die im Allokationsverfahren einsetzbaren multikriteriellen Optimierungsvarianten wurden evaluiert und die gewichtete Summe zeigte die geringste Rechenzeit, während die lexikographische Optimierung mit relativen Toleranzen die höchste Rechenzeit aufwies.

Die Verwendung von komponentenspezifischen Policies wurde evaluiert und es konnte gezeigt werden, dass durch die Verwendung von Constraint Policies machbare Allokationen in einer vergleichbar kurzen Zeit gefunden werden, wie durch die Verwendung von Optimierungen. Das Verhalten der Allokationsmethode für On-Demand Ereignisse, in diesem Fall das Hinzufügen von Teilnehmern oder Mikromobilen im Edge Netzwerk, bestätigt, dass eine verteilte update Allokation, die nur die geänderten Teilnehmer berücksichtigt, einen Vorteil gegenüber der zentralisierten vollständigen Allokation bietet, da das Zeitverhalten des Solvers nicht erhöht wird.

Robustheit und Skalierbarkeit wurden durch systematische Erhöhung der Teilnehmerzahl im Edge Netzwerk bestätigt. Die Erhöhung der Teilnehmerzahl zeigt, dass die Allokationsmethode mit und ohne globale Optimierungen linear skaliert. Bei der Skalierung von Szenarien mit bis zu 3000 Allokationsvariablen bestätigten sich die Vorteile des gestuften Ressourcenmanagements, ebenso wie für den ersten Anwendungsfall. Die verteilte update Allokation reduziert die Rechenzeit des Solvers.

Die Bewertung hat auch gezeigt, dass die folgenden Punkte berücksichtigt werden müssen Die Analyse der Varianten der Mehrzieloptimierung zeigt, dass die Verwendung globaler Optimierungsstrategien im Allgemeinen die Komplexität des Problems erhöht. Die globale Optimierungsvariante nimmt schneller zu, so dass sie bevorzugt in Kombination mit den Mechanismen zur dezentralen Allokation innerhalb von Clustern eingesetzt wird, was

einen zusätzlichen Vorteil der stufenweisen Allokationsmethode darstellt. Bei sehr großen Szenarien steigt auch die verteilte update Allokation linear an, die bei kleineren Szenarien konstant bleibt. Dies spricht ebenfalls für den Einsatz von Clustern im Edge und den gestaffelten Einsatz der verteilten update Allokation, so dass nur die geänderten Teilnehmer innerhalb eines Clusters in die Allokation einbezogen werden.

Aus den Messungen lässt sich ableiten, dass die in dieser Arbeit entwickelte Allokationsmethode auch in einem anderen Anwendungsfall und für unterschiedlich viele Teilnehmer im Edge System eingesetzt werden kann. Verschiedene Optimierungsvarianten haben unterschiedliche Auswirkungen auf das Zeitverhalten des Solvers. Das Zeitverhalten skaliert in allen Varianten linear, allerdings wird wie im ersten Anwendungsfall das gestufte Ressourcenmanagement bevorzugt, bei dem die Allokationsberechnung zunächst nur für die geänderten Teilnehmer statt für das gesamte System durchgeführt wird.

## 8.3 Diskussion und Transferpotential in andere Anwendungen

### 8.3.1 Diskussion der Evaluation

Anhand der beiden Anwendungsfälle (i) Offloading der Perzeption von fahrerlosen Transportfahrzeugen im industriellen Umfeld und (ii) die Veränderungen im System bei der Personenerkennung im Umfeld von kollaborativen Arbeitsmaschinen, konnte die entwickelte Allokationsmethode in zwei industriellen Edge Systemen validiert werden. Dabei wurden die Aspekte der Machbarkeit, der gestuften komponentenspezifischen Policies, der multikriteriellen Optimierungsvarianten, der On-Demand-Ereignisse und der Skalierbarkeit des Konzepts evaluiert.

Generell konnte die zentrale These bestätigt werden, dass auf Basis von Constraint Policies in kurzer Zeit eine machbare Allokation gefunden werden kann. Kombiniert mit dem Ansatz der modellgetriebenen kontinuierlichen Entwicklung wird sichergestellt, dass die Allokation die beschriebenen Anforderungen, Beschränkungen und Ziele aus der Entwurfsphase erfüllt.

Für die Integration der Allokationsmethode im laufenden Betrieb wurden die Auswirkungen von On-Demand-Ereignissen und das Zeitverhalten des Solvers anhand verschiedener Szenarien ausführlich analysiert. Im ersten Anwendungsfall, in Szenarien mit bis zu 108 Allokationsvariablen, und im zweiten Anwendungsfall mit über 120 Teilnehmern und bis zu 3000 Allokationsvariablen, zeigte sich, dass die Allokationsmethode linear mit der Anzahl der Teilnehmer skaliert. Die lineare Skalierung ist eine wichtige Eigenschaft für die Integration im Betrieb.

Die verschiedenen multikriteriellen Optimierungsvarianten wurden untersucht und es zeigte sich, dass komponentenspezifische Policies in Kombination mit der hierarchischen Vorgehensweise (Top-Down für Optimierungspolicies) keinen großen Einfluss auf das Zeitverhalten des Solvers haben. Somit können die komponentenspezifischen Anforderungen verbessert werden. Andererseits führen globale Optimierungen mit Toleranzbereichen zwar zu einem linearen, aber schnelleren Anstieg des Zeitverhaltens des Solvers, so dass globale Optimierungspolicies in Kombination mit den Mechanismen für die dezentrale Allokation innerhalb von Clustern bevorzugt eingesetzt werden, um die Größe des Allokationsproblems zu reduzieren.

In beiden Anwendungsfällen wurden die Vorteile einer gestuften Ressourcenverwaltung bestätigt. Die verteilte update Allokation für On-Demand Ereignisse hat gezeigt, dass die Rechenzeit des Solvers bei kleinen Allokationsproblemen konstant bleibt und bei sehr großen Problemen linear skaliert und im Vergleich zu einer zentralisierten vollständigen Allokation klein bleibt. Die Tatsache, dass die verteilte Allokation die vorherigen Zuweisungen einbezieht und somit nur die Allokation für die geänderten Teilnehmer berechnet, ist eine wichtige Eigenschaft für die Integration des Allokationsverfahrens in den Betrieb.

Mit den bestätigten Eigenschaften der Allokationsmethode und dem erarbeiteten Edge Framework kann also eine automatische und effiziente Allokation von Anwendungen zu Recheneinheiten im Edge Computing Kontext im Betrieb durchgeführt werden, die die Einschränkungen und Ziele aus der Entwurfsphase erfüllt. Auf dieser Grundlage kann eine automatische Reaktion und Selbstanpassung der Datenverarbeitungsinfrastruktur ermöglicht werden, so dass flexible und resiliente Edge Systeme implementiert werden können.

### 8.3.2 Transferpotential in andere Anwendungen

Autonome Recheninfrastrukturen für Edge-Computing-Systeme sind sowohl für verteilte Rechensysteme in der Nähe von Sensoren und Maschinen als auch für die Vernetzung mit Anwendungen in der Cloud relevant. Der Aspekt der effizienten Allokation ist daher in mehreren Anwendungsbereichen von Bedeutung. Der Lösungsansatz dieser Arbeit wurde anhand von industriellen Anwendungen im Bereich Industrie 4.0 am Beispiel von kollaborativen Arbeitsmaschinen und fahrerlosen Transportsystemen in der Fabrik evaluiert. Die Allokationsmethode ist auf andere Anwendungsbereiche übertragbar. Die folgenden Anwendungen aus Projekten der Forschergruppe verdeutlichen das Transferpotenzial der Ansätze und Erkenntnisse dieser Arbeit.

***Intelligente Smart-City-Infrastrukturen:*** Ein weiteres Beispiel für ein Edge-Cloud-System ist eine intelligente Verkehrsinfrastruktur zur Unterstützung des autonomen Fahrens an innerstädtischen Kreuzungen. In diesem Zusammenhang kann die Allokationsmethode genutzt werden, um die effiziente Zuordnung von Funktionen auf den Recheneinheiten in der Kreuzung und in den Servern zu bestimmen. Der Anwendungsfall ist unter anderem in dem Projekt GreenEdge-FuE relevant.

***E/E-Architekturen für das autonome Fahren:*** Im Fahrzeugbereich werden zunehmend High-Performance-Computing-Ansätze verfolgt, um die notwendigen KI-basierten Funktionen für das autonome Fahren in das Fahrzeug integrieren zu können. Zu diesem Zweck werden Konzepte für verteiltes Rechnen im Fahrzeug und in Kooperation mit der umgebenden Infrastruktur entwickelt. In diesem Zusammenhang kann die entwickelte Allokationsmethode genutzt werden, um über effiziente Betriebsstrategien oder das Offloading von Funktionen zwischen dem Fahrzeug und der Infrastruktur zu entscheiden. Der Anwendungsfall ist unter anderem in den Projekten Emdrive und CeCas relevant.

***Industrial Internet of Things:*** Ein weiteres Edge-Cloud-System wird beispielsweise für die vorausschauende Wartung und Fernwartung von Pumpensystemen in der Prozessindustrie benötigt. Hierfür werden Funktionen von den Sensoren an den Maschinen über die Gateways und die lokalen Server bis in die Cloud verteilt. Die entwickelte Allokationsmethode ermöglicht dabei die Berücksichtigung von Sicherheitszonen und erlaubten Zugriffen bei der Allokation von Funktionen. Der Anwendungsfall ist unter anderem in dem Projekt SASVI relevant.

# Kapitel 9

# Zusammenfassung und Schlussfolgerung

In diesem Kapitel werden die erzielten Ergebnisse zusammengefasst und diskutiert.

## 9.1 Zusammenfassung

Edge Computing Infrastrukturen zeichnen sich durch zunehmende Rechenleistung und Konnektivitätsmöglichkeiten sowie einen hohen Grad an Dezentralisierung aus. Um Ausfallzeiten und lange Redesign Schleifen zu reduzieren, werden Selbstanpassungsfähigkeiten (engl. self-adaptation capabilities) benötigt, wie z.B. die automatische Reallokation von ausgeführten Tasks zu Rechenknoten. Dabei sollte die Reallokation den verschiedenen Bedarfen, Einschränkungen und Spezifikationen aus dem Design entsprechen und gleichzeitig sollte die Berechnung der Entscheidung schnell sein, um zur Laufzeit integriert werden zu können.

In dieser Arbeit wurde daher eine policy-basierte Allokationsmethode entwickelt, die zur Laufzeit bei Bedarf eine designkonforme Reallokation von Tasks berechnet. Die Integration der Allokationsmethode in den laufenden Betrieb wird durch eine Kombination von Constraint Programming, schrittweisen multikriteriellen Lösungsansätzen und mehrstufigem Ressourcenmanagement erreicht.

Es wurde der Frage nachgegangen, wie eine automatische und effiziente Allokation zur Laufzeit realisiert werden kann. Hierzu wurde zunächst ein modellgetriebener Entwicklungsprozess analysiert und um Phasen zur kontinuierlichen Adaption erweitert. Das Allokationsproblem im Edge Kontext und die Definition von Effizienz wurden untersucht, um die allokationsrelevanten Informationen als komponentenspezifische Policies zu beschreiben. Das Konzept der Policy-Beschreibung kann somit die Bedürfnisse, Optimierungsziele und Einschränkungen auf verschiedenen Ebenen in einer Schichtenarchitektur erfassen. Es wurde eine leichtgewichtige Allokationsmethode entwickelt, um automatisch über die Zuweisung von Funktionen während des Betriebs zu entscheiden. Dazu werden die allokationsrelevanten Informationen aus dem Policy Modell extrahiert und in ein ausführbares Constraint Programm transformiert. Darüber hinaus wurden verschiedene multikriterielle Lösungsansätze untersucht und Mechanismen entwickelt, um mit minimalem Rechenaufwand machbare und effiziente Lösungen zu finden. Für die Integration und Demonstration wurden die notwendigen Hardware- und Softwarekomponenten eines Edge Frameworks untersucht und erweitert. Basierend auf einer serviceorientierten Architektur, containerbasierten Anwendungen und einer Orchestrierungslösung konnte die Allokationsmethode zur Steuerung des Deployments von Anwendungen im Edge Netzwerk integriert werden.

Die policy-basierte Allokationsmethode wurde anhand von zwei Anwendungsfällen erfolgreich demonstriert und evaluiert: (i) Offloading der Perzeption von fahrerlosen Transportfahrzeugen im industriellen Umfeld und (ii) die Systemänderungen bei der Personenerkennung im Umfeld von kollaborativen Arbeitsmaschinen. Evaluiert wurden die Aspekte Machbarkeit, komponentenspezifische Policies, multikriterielle Optimierungsvarianten, On-Demand Ereignisse und Skalierbarkeit des Konzeptes.

## 9.2 Schlussfolgerung

Die entwickelten Konzepte und die Evaluationsergebnisse zeigen, dass die Beschreibung von Komponentenbedarfen als Policies geeignet ist, um die allokationsrelevanten Informationen zur Entwurfszeit (Dev) zu erfassen. Auf dieser Basis kann die Allokation zur Laufzeit (Ops) entwurfskonform neu berechnet werden. Die Integration in einen modellgetriebenen kontinuierlichen Entwicklungsprozess für selbstadaptive Systeme wurde erfolgreich durchgeführt. Darüber hinaus erfolgt die Allokationsberechnung auf Basis der Policies wie gefordert im Millisekundenbereich.

Insgesamt konnte die zentrale These bestätigt werden, dass auf der Basis von Constraint Policies in kurzer Zeit eine machbare Allokation gefunden werden kann. Für die Integration der Allokationsmethode in den laufenden Betrieb wurden die Auswirkungen von On-Demand-Ereignissen und das Zeitverhalten des Solvers anhand verschiedener Szenarien detailliert analysiert. Mit über 120 Teilnehmern und bis zu 3000 Allokationsvariablen konnte gezeigt werden, dass die Allokationsmethode linear mit der Anzahl der Teilnehmer skaliert. Die lineare Skalierung ist eine wichtige Eigenschaft für die Integration im Betrieb.

In beiden Anwendungsfällen wurden die Vorteile einer mehrstufigen Ressourcenverwaltung bestätigt. Es wurde gezeigt, dass die verteilte update Allokation bei on-demand Ereignissen die Rechenzeit des Solvers im Vergleich zur zentralisierten vollständigen Allokation reduziert. Ein wichtiger Aspekt für die Integration der Allokation in den laufenden Betrieb ist, dass die verteilte Allokation die vorherigen Zuweisungen berücksichtigt und somit nur die Allokation für die geänderten Teilnehmer berechnet wird.

Mit dem Policy Modell, der leichtgewichtigen Allokationsmethode und dem Edge Framework ist eine designkonforme Reallokation möglich, was Redesign Schleifen und Ausfallzeiten reduzieren kann. Insgesamt werden dadurch die selbstadaptiven Fähigkeiten der Edge Computing Infrastruktur verbessert. Die Fähigkeit, auf Veränderungen zu reagieren, erhöht dabei die Resilienz und Verfügbarkeit von Edge Computing Systemen. Zudem ist die Allokationsmethode auf andere Anwendungsbereiche wie Smart-City-Infrastrukturen, E/E-Architekturen für autonome Fahrzeuge oder industrielle Internet-of-Things-Systeme übertragbar.

# Teil IV

# Anhang

# Eigene Veröffentlichungen

Veröffentlichungen von oder mit Beteiligung des Autors dieser Arbeit. Die mit * gekennzeichneten Veröffentlichungen wurden in dieser Arbeit explizit diskutiert.

Erstautor


1. V. P. Betancourt, M. Kirschner, M. Kreutzer, and J. Becker, "Policy-based task allocation at runtime for a self-adaptive edge computing infrastructure," in *2023 IEEE 15th International Symposium on Autonomous Decentralized System (ISADS)*, pp. 1–8, 2023*.

2. V. P. Betancourt, M. Kirschner, M. Kreutzer, and J. Becker, "A policy model for task reallocation at runtime in edge computing infrastructures," , 2023 *(Accepted)**.

3. V. P. Betancourt, B. Liu, and J. Becker, "Towards policy-based task self-reallocation in dynamic edge computing systems," in *2021 IEEE 19th International Conference on Industrial Informatics (INDIN)*, IEEE, jul 2021*.

4. V. P. Betancourt, B. Liu, and J. Becker, "Model-based development of a dynamic container-based edge computing system," in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, oct 2020*.

5. V. P. Betancourt, T. Glock, A. Kharitonov, M. Kern, B. Liu, E. Sax, and J. Becker, "Linking intrusion detection system information and system model to redesign security architecture," in *2020 IEEE International Systems Conference (SysCon)*, IEEE, aug 2020*.

6. V. P. Betancourt, T. Glock, M. Kern, E. Sax, and J. Becker, "Modellbasiertes entwicklungswerkzeug für den entwurf und die analyse von angriffsresistenten industrie 4.0 systemen," in *Automation 2018*, pp. 729–744, VDI Verlag, 2018*.

Co-Autor

1. S. Nitzsche, B. Pachideh, V. Pazmino, N. Link, C. Schauer, L. Theurer, V. Haas, P. Marquardt, S. Biniaminov, and J. Becker, "Neuromorphic vision mit spiking neural networks zur sturzerkennung im betreuten wohnen," 2021.

2. M. Kern, B. Liu, V. P. Betancourt, and J. Becker, "Model-based attack tree generation for cybersecurity risk-assessments in automotive," in *2021 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, sep 2021.

3. B. Liu, T. Glock, V. P. Betancourt, M. Kern, E. Sax, and J. Becker, "Model driven development process for a service-oriented industry 4.0 system," in *2020 9th International Conference on Industrial Technology and Management (ICITM)*, IEEE, feb 2020*.

4. F. Oszwald, P. Obergfell, B. Liu, V. P. Betancourt, and J. Becker, "Model-based design of service-oriented architectures for reliable dynamic reconfiguration," in *SAE Technical Paper Series*, SAE International, apr 2020.

5. B. Liu, V. P. Betancourt, Y. Zhu, and J. Becker, "Towards an on-demand redundancy concept for autonomous vehicle functions using microservice architecture," in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, oct 2020*.

6. M. Kern, E. Taspolatoglu, F. Scheytt, T. Glock, B. Liu, V. P. Betancourt, J. Becker, and E. Sax, "An architecture-based modeling approach using data flows for zone concepts in industry 4.0," in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, oct 2020.

7. T. Glock, V. P. Betancourt, M. Kern, B. Liu, T. Reib, E. Sax, and J. Becker, "Service-based industry 4.0 middleware for partly automated collaborative work of cranes," in *2019 8th International Conference on Industrial Technology and Management (ICITM)*, IEEE, mar 2019*.

8. B. Liu, V. P. Betancourt, T. Glock, M. Kern, E. Sax, and J. Becker, "Model-driven design of tools for multi-domain systems with loosely coupled metamodels," in *2019 IEEE International Systems Conference (SysCon)*, IEEE, apr 2019*.

9. T. Glock, T. Gross, M. Kern, V. P. Betancourt, E. Sax, and J. Becker, "Scenario-based development of an industry 4.0 domain description language for a plant architecture," in *2018 7th International Conference on Industrial Technology and Management (ICITM)*, IEEE, mar 2018*.

# Betreute studentische Arbeiten

1. C. Park, *Entwicklung einer Entscheidungsmethode für energieeffiziente Offloading in Edge Cloud Plattform für Mikromobilität.* Masterarbeit, Karlsruher Institut für Technologie (KIT), 2022.
2. R. Huang, *Concept for Perfomance Prediction of AI Applications on Heterogeneous Edge Devices.* Masterarbeit, Karlsruher Institut für Technologie (KIT), 2022.
3. B. Qin, *Implementierung, Konfiguration und Evaluation von Reallokationsmechanismen in dynamische Edge Computing Orchestratoren.* Bachelor, Karlsruher Institut für Technologie (KIT), 2021.
4. A. Ackigoez, *Systematische Analyse von Hardware/Software Einflussfaktoren für dynamische Edge Computing Szenarien.* Bachelor, Karlsruher Institut für Technologie (KIT), 2020.
5. P. Mehl, *Konzept und Implementierung eines container-basierten Frameworks für die dynamische Service Allokation auf Edge-Geräten.* Bachelor, Karlsruher Institut für Technologie (KIT), 2019.
6. B. Akkir, *Konzept und Implementierung einer Entscheidungsmethode für die dynamische Service-Allokation auf Edge-Geräten.* Bachelor, Karlsruher Institut für Technologie (KIT), 2019.
7. L. Wang, *Modellbasiertes HW/SW Co-Design für Embedded Systems.* Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.
8. P. Baumer, *Methodik zur modellbasierten Architekturbeschreibung von Cyber-Physical Systems im industriellen Umfeld.* Bachelor, Karlsruher Institut für Technologie (KIT), 2018.
9. B. Liu, *Analyse von Konzepten für ein Multi-Domänen Meta-Modell für ein Internet of Things Entwicklungswerkzeug.* Master, Karlsruher Institut für Technologie (KIT), 2018.
10. M. Ghawi, *Anforderungsanalyse für ein I4.0 Security Entwicklungswerkzeug.* Bachelor, Karlsruher Institut für Technologie (KIT), 2017.

# Literaturverzeichnis


[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, oct 2016.

[2] S. Josilo, *Task Placement and Resource Allocation in Edge Computing Systems*. Stockholm: KTH Royal Institute of Technology, 2020.

[3] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184–206, feb 2015.

[4] K. Ashton, "That 'internet of things' thing," 1999.

[5] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, IEEE, nov 2015.

[6] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud," in *Proceedings of the sixth conference on Computer systems - EuroSys '11*, ACM Press, 2011.

[7] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems*, vol. 14, pp. 54–62, may 1999.

[8] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, jan 2003.

[9] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Computing Surveys*, vol. 40, pp. 1–28, aug 2008.

[10] M. Handte, G. Schiele, V. Matjuntke, C. Becker, and P. J. Marrón, "3pc," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 7, pp. 1–19, apr 2012.

[11] M. Rohr, S. Giesecke, M. Hiel, W.-J. van den Heuvel, H. H. Weigand, and W. Hasselbring, "A classification scheme for self-adaptation research," 2006.

[12] M. Salehie and L. Tahvildari, "Self-adaptive software," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, pp. 1–42, may 2009.

[13] V. Majuntke, S. VanSyckel, D. Schäfer, C. Krupitzer, G. Schiele, and C. Becker, "Comity: Coordinated application adaptation in multi-platform pervasive systems," *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 11–19, 2013.

[14] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, "Composing adaptive software," *Computer*, vol. 37, pp. 56–64, jul 2004.

[15] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability," *IEEE Software*, vol. 23, pp. 62–70, mar 2006.

[16] P. Lalanda, J. A. Mccann, and A. Diaconescu, "Autonomic computing architectures," 2013.

[17] D. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, pp. 25–31, feb 2006.

[18] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering (FOSE '07)*, IEEE, may 2007.

[19] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. Papadopoulos, "A development framework and methodology for self-adapting applications in ubiquitous computing environments," *Journal of Systems and Software*, vol. 85, pp. 2840–2859, dec 2012.

[20] D. Menasce, H. Gomaa, S. Malek, and J. Sousa, "SASSY: A framework for self-architecting service-oriented systems," *IEEE Software*, vol. 28, pp. 78–85, nov 2011.

[21] A. R. da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Computer Languages, Systems and Structures*, vol. 43, pp. 139–155, oct 2015.

[22] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Springer International Publishing, 2017.

[23] D. M. Fernández, W. Böhm, A. Vogelsang, J. Mund, M. Broy, M. Kuhrmann, and T. Weyer, "Artefacts in software engineering: a fundamental positioning," *Software and Systems Modeling*, vol. 18, pp. 2777–2786, jan 2019.

[24] "ISO/IEC/IEEE systems and software engineering – architecture description."

[25] J. Gausemeier and S. Moehringer, "VDI 2206- a new guideline for the design of mechatronic systems," *IFAC Proceedings Volumes*, vol. 35, pp. 785–790, dec 2002.

[26] A. Braun, "Modellbasierte unterstützung der produktentwicklung - potentiale der modellierung von produktentstehungsprozessen am beispiel des integrierten produktentstehungsmodells (ipem) = model based support of product development - potentials of modelling product engineering processes using the example of the integrated product engineering model (ipem)," 2013.

[27] J. Gausemeier, A. Trächtler, and W. Schäfer, *Semantische Technologien im Entwurf mechatronischer Systeme*. Carl Hanser Verlag GmbH & Co. KG, jun 2014.

[28] B. Combemale and M. Wimmer, "Towards a model-based DevOps for cyber-physical systems," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pp. 84–94, Springer International Publishing, 2020.

[29] D. Weyns, "Software engineering of self-adaptive systems," in *Handbook of Software Engineering*, pp. 399–443, Springer International Publishing, 2019.

[30] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, pp. 1–32, Springer Berlin Heidelberg, 2013.

[31] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, pp. 1–26, Springer Berlin Heidelberg, 2009.

[32] J. Holtmann, R. Bernijazov, M. Meyer, D. Schmelter, and C. Tschirner, "Integrated and iterative systems engineering and software requirements engineering for technical systems," *Journal of Software: Evolution and Process*, vol. 28, pp. 722–743, may 2016.

[33] J. Gausemeier, S. Korf, M. Porrmann, K. Stahl, O. Sudmann, and M. Vaßholz, "Development of self-optimizing systems," in *Lecture Notes in Mechanical Engineering*, pp. 65–115, Springer Berlin Heidelberg, 2014.

[34] J. Gausemeier, U. Frank, J. Donoth, and S. Kahl, "Specification technique for the description of self-optimizing mechatronic systems," *Research in Engineering Design*, vol. 20, pp. 201–223, feb 2009.

[35] J. G. Lamm and T. Weilkiens, "Method for deriving functional architectures from use cases," *Systems Engineering*, vol. 17, pp. 225–236, may 2013.

[36] T. Weilkiens, J. G. Lamm, S. Roth, and M. Walker, *Model-Based System Architecture*. John Wiley & Sons, Inc, sep 2015.

[37] K. Pohl, H. Hönninger, R. Achatz, and M. Broy, eds., *Model-Based Engineering of Embedded Systems*. Springer Berlin Heidelberg, 2012.

[38] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages," *ACM SIGPLAN Notices*, vol. 35, pp. 26–36, jun 2000.

[39] T. Kosar, S. Bohra, and M. Mernik, "Domain-specific languages: A systematic mapping study," *Information and Software Technology*, vol. 71, pp. 77–91, mar 2016.

[40] T. Kühne, "Matters of (meta-) modeling," *Software and Systems Modeling*, vol. 5, pp. 369–385, jul 2006.

[41] J. Sprinkle, B. Rumpe, H. Vangheluwe, and G. Karsai, "3 metamodelling," in *Model-Based Engineering of Embedded Real-Time Systems*, pp. 57–76, Springer Berlin Heidelberg, 2010.

[42] T. Mens, "Model transformation: A survey of the state of the art," in *Model-Driven Engineering for Distributed Real-Time Systems*, pp. 1–19, John Wiley & Sons, Inc., mar 2013.

[43] T. Mens and P. V. Gorp, "A taxonomy of model transformation," *Electronic Notes in Theoretical Computer Science*, vol. 152, pp. 125–142, mar 2006.

[44] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–645, 2006.

[45] A. Metzger, "A systematic look at model transformations," in *Model-Driven Software Development*, pp. 19–33, Springer-Verlag.

[46] S. Sendall and W. Kozaczynski, "Model transformation: the heart and soul of model-driven software development," *IEEE Software*, vol. 20, pp. 42–45, sep 2003.

[47] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF*. Addison-Wesley, 2009.

[48] K. Toczé and S. Nadjm-Tehrani, "A taxonomy for management and optimization of multiple resources in edge computing," *Wirel. Commun. Mob. Comput.*, vol. 2018, pp. 7476201:1–7476201:23, 2018.

[49] D. W. Pentico, "Assignment problems: A golden anniversary survey," *European Journal of Operational Research*, vol. 176, pp. 774–793, jan 2007.

[50] Naren, A. K. Gaurav, N. Sahu, A. P. Dash, G. S. S. Chalapathi, and V. Chamola, "A survey on computation resource allocation in IoT enabled vehicular edge computing," *Complex & Intelligent Systems*, jul 2021.

[51] J. He, Y. Li, H. Li, H. Tong, Z. Yuan, X. Yang, and W. Huang, "Application of game theory in integrated energy system systems: A review," *IEEE Access*, vol. 8, pp. 93380–93397, 2020.

[52] W. Guo, Z. Chang, X. Guo, D. N. K. Jayakody, and T. Ristaniemi, "Resource allocation for edge computing-based blockchain: A game theoretic approach," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, IEEE, jun 2020.

[53] W. Wei, X. Fan, H. Song, X. Fan, and J. Yang, "Imperfect information dynamic stackelberg game based resource allocation using hidden markov for cloud computing," *IEEE Transactions on Services Computing*, vol. 11, pp. 78–89, jan 2018.

[54] G. Wang and F. Xu, "Regional intelligent resource allocation in mobile edge computing based vehicular network," *IEEE Access*, vol. 8, pp. 7173–7182, 2020.

[55] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, pp. 30–39, jan 2017.

[56] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, pp. 112–116, aug 2016.

[57] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing," *ACM Computing Surveys*, vol. 52, pp. 1–37, sep 2020.

[58] B. Confais, A. Lebre, and B. Parrein, "An object store service for a fog/edge computing infrastructure based on IPFS and a scale-out NAS," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, IEEE, may 2017.

[59] B. Amento, B. Balasubramanian, R. J. Hall, K. Joshi, G. Jung, and K. H. Purdy, "FocusStack: Orchestrating edge clouds using location-based focus of attention," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, IEEE, oct 2016.

[60] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *2014 International Conference on Future Internet of Things and Cloud*, IEEE, aug 2014.

[61] S. Nanda and T. cker Chiueh, "A survey on virtualization technologies," 2005.

[62] C.-H. Hong, I. Spence, and D. S. Nikolopoulos, "GPU virtualization and scheduling methods," *ACM Computing Surveys*, vol. 50, pp. 1–37, oct 2017.

[63] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, ACM Press, 2003.

[64] A. Kivity, "kvm : the linux virtual machine monitor," 2007.

[65] N. Haydel, S. Gesing, I. Taylor, G. Madey, A. Dakkak, S. G. de Gonzalo, and W.-M. W. Hwu, "Enhancing the usability and utilization of accelerated architectures via docker," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, IEEE, dec 2015.

[66] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures – a technology review," in *2015 3rd International Conference on Future Internet of Things and Cloud*, IEEE, aug 2015.

[67] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, p. 2, 2014.

[68] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ACM, aug 2015.

[69] V. Cardellini, F. L. Presti, M. Nardelli, and F. Rossi, "Self-adaptive container deployment in the fog: A survey," in *Algorithmic Aspects of Cloud Computing*, pp. 77–102, Springer International Publishing, 2020.

[70] S. Rahul and R. Aron, "Fog computing architecture, application and resource allocation: A review," 2021.

[71] Z. Lin, J. Liu, J. Xiao, and S. Zi, "A survey: Resource allocation technology based on edge computing in IIoT," in *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, IEEE, nov 2020.

[72] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, pp. 587–597, mar 2018.

[73] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, IEEE, apr 2018.

[74] J. P. Champati and B. Liang, "Single restart with time stamps for computational offloading in a semi-online setting," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, IEEE, may 2017.

[75] S. Josilo and G. Dan, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Transactions on Networking*, vol. 27, pp. 85–97, feb 2019.

[76] L. Xiang, B. Li, and B. Li, "Coalition formation towards energy-efficient collaborative mobile computing," in *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, IEEE, aug 2015.

[77] Z. Chang, Z. Zhou, T. Ristaniemi, and Z. Niu, "Energy efficient optimization for computation offloading in fog computing system," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, IEEE, dec 2017.

[78] X. Chen and J. Zhang, "When d2d meets cloud: Hybrid mobile task offloadings in fog computing," in *2017 IEEE International Conference on Communications (ICC)*, IEEE, may 2017.

[79] O. Oleghe, "Container placement and migration in edge computing: Concept and scheduling models," *IEEE Access*, vol. 9, pp. 68028–68043, 2021.

[80] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, and S. Papavassiliou, "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Computer Networks*, vol. 195, p. 108177, aug 2021.

[81] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, pp. 3246–3257, aug 2018.

[82] S. Patil-Karpe, S. H. Brahmananda, and S. Karpe, "Review of resource allocation in fog computing," in *Smart Intelligent Computing and Applications*, pp. 327–334, Springer Singapore, sep 2019.

[83] A. Denger, J. Fritz, D. Denger, P. Priller, C. Kaiser, and A. Stocker, "Organisationaler wandel durch die emergenz cyber-physikalischer systeme: Die fallstudie AVL list GmbH," *HMD Praxis der Wirtschaftsinformatik*, vol. 51, pp. 827–837, oct 2014.

[84] J. Gausemeier, A. Czaja, O. Wiederkehr, R. Dumitrescu, C. Tschirner, and D. Steffen, "Studie: Systems engineering in der industriellen praxis," in *Tag des Systems Engineering*, pp. 113–122, Carl Hanser Verlag GmbH & Co. KG, nov 2013.

[85] M. A. Lopez-Pena, J. Diaz, J. E. Perez, and H. Humanes, "DevOps for IoT systems: Fast and continuous monitoring feedback of system availability," *IEEE Internet of Things Journal*, vol. 7, pp. 10695–10707, oct 2020.

[86] V. P. Betancourt, B. Liu, and J. Becker, "Model-based development of a dynamic container-based edge computing system," in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, oct 2020.

[87] V. P. Betancourt, B. Liu, and J. Becker, "Towards policy-based task self-reallocation in dynamic edge computing systems," in *2021 IEEE 19th International Conference on Industrial Informatics (INDIN)*, IEEE, jul 2021.

[88] I. Alfonso, K. Garcés, H. Castro, and J. Cabot, "Self-adaptive architectures in IoT systems: a systematic literature review," *Journal of Internet Services and Applications*, vol. 12, dec 2021.

[89] B. Liu, V. P. Betancourt, T. Glock, M. Kern, E. Sax, and J. Becker, "Model-driven design of tools for multi-domain systems with loosely coupled metamodels," in *2019 IEEE International Systems Conference (SysCon)*, IEEE, apr 2019.

[90] K. Kernschmidt, B. Vogel-Heuser, G. Barbieri, and C. Fantuzzi, "Interdisziplinäre modellbasierte entwicklung mechatronischer systeme basierend auf sysml zur steigerung der wiederverwendung," in *VDI-Kongress Automation (VDI KA 2014)*, Baden-Baden, Germany: VDI Verlag, 2014.

[91] S. Heymann, J. Jasperneite, S. Schröck, and A. Fay, "Beschreibung von produktionsprozessen in modularisierten produktionsanlagen für industrie 4.0," in *Automation 2015*, VDI Verlag, 2015.

[92] T. Holm, M. Obst, A. Fay, L. Urbas, T. Albers, S. Kreft, and U. Hempen, "Dezentrale intelligenz für modulare automation," *atp edition - Automatisierungstechnische Praxis*, vol. 56, p. 34, nov 2014.

[93] B. Böhm, J. Vollmar, S. Unverdorben, A. Calà, and S. Wolf, "Ganzheitlicher modellbasierter entwurf von systemarchitekturen für industrielle anlagen," in *Automation 2020*, pp. 887–898, VDI Verlag, 2020.

[94] V. P. Betancourt, M. Kern, and B. Liu, "Entwicklungswerkzeug für anwendungsoptimierte hardwarebasierte sicherheitstechnologien für i4.0-anwendungen *syskit_hw* : Abschlussbericht, syskit – abschlussbericht," tech. rep., FZI Forschungszentrum Informatik, Karlsruhe, 2020.

[95] V. P. Betancourt, T. Glock, M. Kern, E. Sax, and J. Becker, "Modellbasiertes entwicklungswerkzeug für den entwurf und die analyse von angriffsresistenten industrie 4.0 systemen," in *Automation 2018*, pp. 729–744, VDI Verlag, 2018.

[96] T. Glock, T. Gross, M. Kern, V. P. Betancourt, E. Sax, and J. Becker, "Scenario-based development of an industry 4.0 domain description language for a plant architecture," in *2018 7th International Conference on Industrial Technology and Management (ICITM)*, IEEE, mar 2018.

[97] B. Liu, T. Glock, V. P. Betancourt, M. Kern, E. Sax, and J. Becker, "Model driven development process for a service-oriented industry 4.0 system," in *2020 9th International Conference on Industrial Technology and Management (ICITM)*, IEEE, feb 2020.

[98] V. P. Betancourt, T. Glock, A. Kharitonov, M. Kern, B. Liu, E. Sax, and J. Becker, "Linking intrusion detection system information and system model to redesign security architecture," in *2020 IEEE International Systems Conference (SysCon)*, IEEE, aug 2020.

[99] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, mar 1955.

[100] V. Sarcar, "Adapter patterns," in *Java Design Patterns*, pp. 47–52, Apress, 2016.

[101] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, pp. 369–395, apr 2004.

[102] K. N. Brown and I. Miguel, "Uncertainty and change," in *Handbook of Constraint Programming*, pp. 731–760, Elsevier, 2006.

[103] L. Perron and V. Furnon, "Or-tools," 2022.

[104] IBM and Cplex, "V20. 1: User's manual for cplex," *International Business Machines Corporation*, 2021.

[105] C. Köhler-Schute and J. Amberg, *Industrie 4.0: ein praxisorientierter Ansatz*. KS-Energy-Verlag, 2015.

[106] T. Glock, S. Otten, S. Rebmann, and E. Sax, "Modellbasierte planung und konfiguration von verteilten funktionsumfängen in der feldebene model-based planning and configuration of distributed functions in the field," in *Automation 2017*, VDI Verlag, 2017.

[107] acatech, *Cyber-Physical Systems*. Springer Berlin Heidelberg, 2011.

[108] T. Glock, V. P. Betancourt, M. Kern, B. Liu, T. Reib, E. Sax, and J. Becker, "Service-based industry 4.0 middleware for partly automated collaborative work of cranes," in *2019 8th International Conference on Industrial Technology and Management (ICITM)*, IEEE, mar 2019.

[109] A. Araldo, A. D. Stefano, and A. D. Stefano, "Resource allocation for edge computing with multiple tenant configurations," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ACM, mar 2020.

[110] B. Liu, V. P. Betancourt, Y. Zhu, and J. Becker, "Towards an on-demand redundancy concept for autonomous vehicle functions using microservice architecture," in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, oct 2020.

[111] X. Ye and S. H. Hong, "An AutomationML/OPC UA-based industry 4.0 solution for a manufacturing system," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, sep 2018.

[112] V. P. Betancourt, M. Kirschner, M. Kreutzer, and J. Becker, "Policy-based task allocation at runtime for a self-adaptive edge computing infrastructure," in *2023 IEEE 15th International Symposium on Autonomous Decentralized System (ISADS)*, pp. 1–8, 2023.

[113] V. P. Betancourt, M. Kirschner, M. Kreutzer, and J. Becker, "A policy model for task reallocation at runtime in edge computing infrastructures," , 2023.

[114] S. Nitzsche, B. Pachideh, V. Pazmino, N. Link, C. Schauer, L. Theurer, V. Haas, P. Marquardt, S. Biniaminov, and J. Becker, "Neuromorphic vision mit spiking neural networks zur sturzerkennung im betreuten wohnen," 2021.

[115] M. Kern, B. Liu, V. P. Betancourt, and J. Becker, "Model-based attack tree generation for cybersecurity risk-assessments in automotive," in *2021 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, sep 2021.

[116] F. Oszwald, P. Obergfell, B. Liu, V. P. Betancourt, and J. Becker, "Model-based design of service-oriented architectures for reliable dynamic reconfiguration," in *SAE Technical Paper Series*, SAE International, apr 2020.

[117] M. Kern, E. Taspolatoglu, F. Scheytt, T. Glock, B. Liu, V. P. Betancourt, J. Becker, and E. Sax, "An architecture-based modeling approach using data flows for zone concepts in industry 4.0," in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, oct 2020.

[118] C. Park, *Entwicklung einer Entscheidungsmethode für energieeffiziente Offloading in Edge Cloud Plattform für Mikromobilität.* Masterarbeit, Karlsruher Institut für Technologie (KIT), 2022.

[119] R. Huang, *Concept for Perfomance Prediction of AI Applications on Heterogeneous Edge Devices.* Masterarbeit, Karlsruher Institut für Technologie (KIT), 2022.

[120] B. Qin, *Implementierung, Konfiguration und Evaluation von Reallokationsmechanismen in dynamische Edge Computing Orchestratoren.* Bachelor, Karlsruher Institut für Technologie (KIT), 2021.

[121] A. Ackigoez, *Systematische Analyse von Hardware/Software Einflussfaktoren für dynamische Edge Computing Szenarien.* Bachelor, Karlsruher Institut für Technologie (KIT), 2020.

[122] P. Mehl, *Konzept und Implementierung eines container-basierten Frameworks für die dynamische Service Allokation auf Edge-Geräten.* Bachelor, Karlsruher Institut für Technologie (KIT), 2019.

[123] B. Akkir, *Konzept und Implementierung einer Entscheidungsmethode für die dynamische Service-Allokation auf Edge-Geräten.* Bachelor, Karlsruher Institut für Technologie (KIT), 2019.

[124] L. Wang, *Modellbasiertes HW/SW Co-Design für Embedded Systems.* Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.

[125] P. Baumer, *Methodik zur modellbasierten Architekturbeschreibung von Cyber-Physical Systems im industriellen Umfeld.* Bachelor, Karlsruher Institut für Technologie (KIT), 2018.

[126] B. Liu, *Analyse von Konzepten für ein Multi-Domänen Meta-Modell für ein Internet of Things Entwicklungswerkzeug.* Master, Karlsruher Institut für Technologie (KIT), 2018.

[127] M. Ghawi, *Anforderungsanalyse für ein I4.0 Security Entwicklungswerkzeug.* Bachelor, Karlsruher Institut für Technologie (KIT), 2017.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Quellcodeverzeichnis