Beitrag zur Selbstorganisation in der Intralogistik

Zur Erlangung des akademischen Grades eines

Dr.-Ing.

von der Fakultät Maschinenbau
der Technischen Universität Dortmund
genehmigte Dissertation

Peter Detzner, M. Sc.

aus

Laurahütte

Tag der mündlichen Prüfung: 03. Mai 2023

1. Gutachter: Prof. Dr. Dr. h. c. Michael ten Hompel

2. Gutachter: Prof. Dr. Steffen Bondorf

Dortmund, 2023

Kurzfassung

In der Natur existieren viele dezentral organisierte Systeme. Ein bekanntes Beispiel hierfür ist eine Insektenkolonie, bei dem jedes Insekt die ihm lokal verfügbaren Informationen für seine eigene Handlung oder für die Interaktion mit anderen Insekten verwendet. Gemeinsam arbeiten diese an komplexeren, globaleren Zielen der Kolonie. Es entsteht eine Emergenz auf Basis von einfacher Interaktion. Hieraus ergibt sich Frage, weshalb die Intralogistik und der darin enthaltene Materialfluss bisher nicht dezentral organisiert sind. Denn die immer komplexer werdenden materialflusstechnischen Abläufe benötigen neue Ansätze, damit sie idealerweise effizient gelöst werden können. Ein möglicher Ansatz hierfür ist eine Erhöhung der Autonomie der einzelnen Teilnehmer, die konsequente Dezentralisierung der Prozesse und insbesondere der Kommunikation. Somit muss neben der Interaktion der Teilnehmer auch die Kommunikation zwischen diesen auf der Organisationsebene betrachtet werden.

In der vorliegenden Dissertation wird ein Konzept für einen selbstorganisierenden Materialfluss vorgestellt und umgesetzt. Grundlage hierfür ist eine konsequente Modularisierung, was zu einer Dezentralisierung eines monolithischen Systems führt. Dabei wird die Gesamtintelligenz des Systems aufgeteilt und auf seine Teilnehmer verteilt. Konkret bedeutet dies, dass der Materialfluss mithilfe einer domänenspezifische Sprache beschrieben und formalisiert wird. Aufbauend hierauf erfolgt eine Virtualisierung des Materialflusses, sodass eine eigenständige Entität entstehen kann, welche mit Fahrerlosen Transportfahrzeugen in Verhandlung tritt und sich über die zu leistenden Transportaufträge abstimmt. Gemeinsam verfolgen alle Teilnehmer das globale Ziel der (effizienteren) Abarbeitung von Transportaufträgen und tragen somit zur Selbstorganisation in der Intralogistik bei. Die eingeführte Selbstorganisation hat allerdings auch einen Einfluss auf die Kommunikation. Denn mit der Erhöhung der Autonomie, wie in der Natur zu beobachten, steigt der Bedarf zur Synchronisation der Teilnehmer und daher auch der Kommunikationsaufwand. Dies hat zur Folge, dass die Kommunikation mit der Anzahl an Teilnehmern skalieren muss. Hierfür wird die inhärente Fähigkeit zur Kommunikation der Teilnehmer ausgenutzt, um ein dezentral organisiertes Kommunikationsframework auf Basis einer nullbalancierten, vollständigen Baumstruktur zu entwickeln. Unter Verwendung eines quelloffenen Simulationsframeworks für Rechnernetze wird das dezentral organisierte Kommunikationsframework und seine Bestandsteile analytisch und simulativ ausgewertet. Anhand zweier Anwendungsfällen wird die Anwendbar- und Machbarkeit das Konzepts eines selbstorganisierenden Materialflüssen in der Intralogistik gezeigt und die aus der Interaktion der Teilnehmer entstandene Kommunikation unter Verwendung des dezentral organisierten Kommunikationsframeworks evaluiert.

Abstract

In nature there are many systems being organized in a decentral manner. A well-known example is an insect colony, in which each insect uses information locally available for its own action or for interaction with other insects. Together, these insects work toward more complex, global goals of the colony. This is how emergence occurs based on simple interaction. This raises the question of why intralogistics and the material flow it contains have not yet been organized in a decentralized manner. After all, increasingly complex material flow processes require new approaches so that they can ideally be solved. One possible approach is an increase in the autonomy of the individual participants, the consistent decentralization of processes, and, in particular, communication. Thus, in addition to the interaction of the participants, the communication between them must also be considered on an organizational level.

In this dissertation, a concept for a self-organizing material flow is presented and implemented. The basis for this is a consistent modularization, which leads to a decentralization of a monolithic system. In this process, the overall intelligence of the system is divided and distributed among its participants. In concrete terms, the material flow is described and formalized using a domain-specific language. Based on this, the material flow is virtualized, so that an independent entity that negotiates with automated guided vehicles and coordinates the transport orders to be performed can emerge. All participants collaborate in order to reach common global goals of (more efficient) processing of transport orders and thus contribute to self-organization in intralogistics. In addition, the self-organization that has been introduced also influences the respective communication. With the increase of autonomy, as observed in nature, the need for synchronization of the participants growth as well as the effort of communication. As a consequence, communication has to scale with the number of participants. For this purpose, the inherent ability of participants to communicate is exploited to develop a framework for decentralized communication based on a null-balanced complete tree structure. With the help of an open source simulation framework for computer networks, the decentrally organized communication framework and its components are evaluated analytically and simulatively. On the basis of two use cases, the applicability and feasibility of the concept of a self-organizing material flow in intralogistics is demonstrated and the communication resulting from the interaction of the participants is evaluated using the decentrally organized communication framework.

Danksagungen

Diese vorliegende Arbeit wäre ohne die Hilfe mehrerer Personen nicht möglich gewesen, von denen viele vielleicht gar nicht wissen, dass sie zu dieser Arbeit beigetragen haben.

Zuallererst möchte ich bei meinen Eltern *Gabriela und Waldemar Detzner* und meinen Geschwistern *Nicole und Rafael Detzner* für ihre unermüdliche Unterstützung bedanken. Auch wenn es vielleicht nicht direkt ersichtlich ist, so habt ihr aber einen erheblichen Beitrag dazu geleistet.

Dann möchte ich mich bei meinem Doktorvater Herrn *Prof. Dr. Dr. h.c. Michael ten Hompel* für die Gelegenheit, seine Unterstützung, die Betreuung und die wissenschaftlichen Gestaltungsfreiräume bedanken. Ein weiterer groß Dank gehört auch Herrn *Prof. Dr.-Ing. Steffen Bondorf*, und zwar nicht nur für seine Bereitschaft als Zweitgutachter zu fungieren, sondern auch für die stets motivierenden und hilfreichen Gespräche sowie die gemeinsamen Arbeiten. Diese waren zwar sehr fordernd und fördernd, aber immer progressiv und zielführend.

Diese Arbeit ist während meiner Tätigkeit als Wissenschaftlicher Mitarbeiter am Fraunhofer Institut für Materialfluss und Logistik IML in Dortmund entstanden. Ein besonderer Dank gilt meinem Vorgesetzten *Dr.-Ing. Sören Kerner*, der mir zu jedem Zeitpunkt sein absolutes Vertrauen entgegen gebracht und stets an mich geglaubt hat. Aber auch allen meinen anderen Kolleginnen und Kollegen und den studentischen bzw. wissenschaftlichen Hilfskräften danke ich für die zahlreichen praktischen Diskussionen, Hilfestellungen bei der Zusammenstellung von Wissen und für die Reflexion von Ergebnissen (in alphabetischer Reihenfolge): *Marius Brehler*, *Anja Eikholt*, *Jana Gödeke*, *Maximilian Hörstrup*, *Jana Jost*, *Patrick Laskowski*, *Dennis Lünsch*, *Oliver Stolz* und *Lars Tönning*.

Abschließend möchte ich mich bei einigen Personen bedanken, die in irgendeiner Weise zu meiner Forschung beigetragen haben (erneut in alphabetischer Reihenfolge): *Nasir Delic*, *Heiko Fechtner*, *Henriette Gert*, *Christoph Hoffmann*, *Paul Ihly*, *Hosun Lee*, *Guillaume Mammi*, *Pierre Paranque*, *Ann-Kathrin und Olaf Poneta*, *Kathrin und Marc Sangermann*, *Moritz Schäfer* und *Dennis Weyland*.

Inhaltsverzeichnis

Kapitel 1
Einleitung

> "*A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable*"
>
> — Leslie Lamport

1.1 Ausgangslage und Motivation

In der Mythologie hatte die Göttin des Zufalls bzw. des Glücks den Beinamen *Automatia*: Sie galt als diejenige, die alles lenkt(e) ohne eine Beteiligung der Menschen [VBM77, S. 85]. Hiervon lässt sich das Wort *Automatisierung* ableiten, welches für das selbsttätige Handeln steht und im Vordergrund der vierten industriellen (R-)Evolution – auch bekannt als Industrie 4.0 (I4.0) – neben der *Produktion* und *Logistik* zu finden ist [BtV14]. Ein Teilbereich der Logistik ist der Materialfluss, welcher sich mit der Lagerung, Verpackung und Bewegung von Waren jeglicher Art befasst. Je nach Branche und Produkt machen die Kosten für den innerbetrieblichen Materialfluss, zu dem der Transport gehört, einen Anteil von 50 % und mehr aus [Mar14, S. 29]. Allein diese Feststellung sollte daher als starke Motivation für die Automatisierung des Materialflusses in der Produktion dienen.

Fabriken müssen daher den nächsten Schritt in der Digitalisierung und Automatisierung gehen: zum einen um die Kosten für den innerbetrieblichen Materialfluss zu reduzieren und zum anderen um ihre Flexibilität, Produktivität, Schnelligkeit und Qualität zu steigern, wie z. B. durch die Transformation der statischen Produktionslinien in flexible, vollständig digital integrierte Einrichtungen [HZW+08]. Getrieben wird dies durch die Erwartungshaltung der Kunden. Produkte sollen individuell konfigurierbar oder personalisiert sein [Kor10, BFK+14]. Daher ist das Ziel, dass eine Produktion und somit ihre Logistik nach einer (Re-)Konfiguration der Prozesse nicht manuell neu konfiguriert werden muss, sondern dies automatisch erfolgt. Die

Komponenten organisieren sich selbstständig durch Kommunikationskanäle über die zu leistenden Arbeitsabfolgen [NFH10, BLS+17].

Um dies zu erreichen, sieht Monostori [Mon14] die reale Welt mit der digitalen Welt verschmelzen. Neueste Trends und Technologien der Informatik, der Informations- und Kommunikationstechnik und der Fertigungstechnik bilden dabei ein cyberphysisches Produktionssystem (CPPS). Um Flexibilität und Produktivität in einem CPPS zu gewährleisten und die Logistik an neue Anforderungen anpassen zu können, ist die Verwendung von standardisierten Komponenten notwendig [Sch18a, S. 1 f.]. Hierbei sind die Interkonnektivität von Maschinen, die dezentrale Entscheidungsfindung, die Transparenz von Informationen und die technische Unterstützung des Menschen durch Maschinen die vier Designprinzipien für die Industrie 4.0 [HPO16].

Die Standardisierung von Schnittstellen begünstigt eine Interkonnektivität von Komponenten in einem CPPS [HPO16]. Mithilfe von offenen, standardisierten Schnittstellen können sogar Abhängigkeiten von Herstellern vermieden werden [SAD+20, S. 29]. Dabei setzt die Interkonnektivität auch eine Selbstbeschreibung voraus [BKR+16, S. 11]. Komponenten können nicht nur ihren Zustand beschreiben, sondern diesen und den der Umgebung wahrnehmen. Basierend auf offenen, standardisierten Schnittstellen können Komponenten in einem CPPS hinzugefügt, ausgetauscht oder entfernt werden, sich eigenständig ins System integrieren und (re-)konfigurieren [ST09].

Für die dezentrale Entscheidungsfindung ist die Transformation der hierarchischen Automationspyramide in eine Dekomposition von modularen Komponenten (vgl. Abb. 1.1) notwendig [Mon14]. In der klassischen Automationspyramide werden Entscheidungen hierarchisch in der obersten Schicht getroffen und an tiefere weitergeleitet. Die Daten hingegen werden auf den tieferen Schichten erfasst und aggregiert, bevor diese zur nächsthöheren Schicht weitergeleitet werden. Diese Daten und die beinhalteten Informationen werden für höhere Schichten abstrahiert. Durch die Abstraktion können Informationen verloren gehen, die für eine höhere Schicht von Bedeutung sein könnten. Wird eine Komponente in einer Schicht angepasst oder ausgetauscht, so betrifft dies andere Komponenten auf der gleichen Schicht (horizontal). Wird eine ganze Schicht angepasst oder ausgetauscht, so betrifft dies die benachbarten (vertikalen) Schichten. Anpassungen bzw. Änderungen sind folglich mit einem Zeitaufwand verbunden und daher kostenintensiv.

Eine Dekomposition einer Automationspyramide in ein verteiltes System aus selbstorganisierenden, dezentralen Entitäten und Diensten setzt eine Kommunikation voraus. Hier sind lediglich diejenigen Komponenten miteinander verbunden, welche an einem Datenaustausch untereinander interessiert sind. Diese Verbindungen werden genutzt, um eine Entscheidung zu treffen. Unter Umständen kann es sein, dass diese Entscheidung für die entsprechende Komponente suboptimal ist, jedoch einem globalen Ziel dient [BFK+14, DRVD20]. Somit werden Entscheidungen nicht zentral getroffen, sondern dezentral durch eine Aushandlung unter den Teilnehmern [WH07, S. 7 ff.].

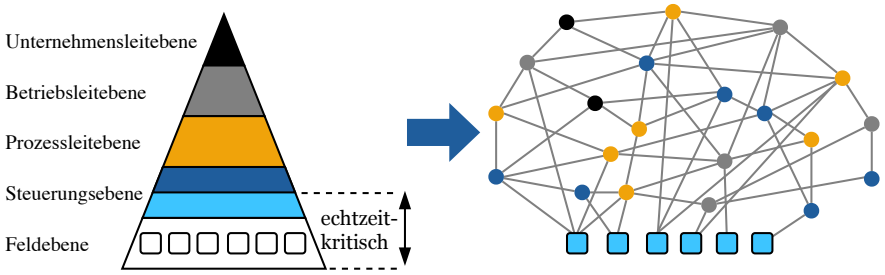


Abb. 1.1: Dekomposition der starren, hierarchischen Automationspyramide in dynamische, selbstorganisierende, dezentrale Entitäten und Dienste innerhalb einer Fabrik, basierend auf [Ver13]

1.2 Problemstellung

Zentralisierte Systeme sind nicht mehr in der Lage die steigende Systemkomplexität zu bewältigen, sodass die Dezentralisierung der Systeme als eine mögliche Lösung vorgeschlagen wird [MVD+15, HPO16, HHH+19], um komplexe monolithische Systeme aufzubrechen. Dies ermöglicht, durch die gesteigerte Autonomie der Teilnehmer flexibel auf Änderungen zu reagieren, wie beim Materialtransport [DMK+16, HPO16]: Unter der Annahme, dass jede Maschine nur einen endlichen Materialbestand besitzt, sind Nachfüllungen unumgänglich. Benötigt eine Maschine Material, so ist eine Möglichkeit, dass diese das Material eigenständig anfordert. Die Zustellung des Materials kann dann manuell durch einen Menschen oder durch die Verwendung von Fördertechnik bzw. autonomen Komponenten erfolgen. Ein Fahrerloses Transportfahrzeug (FTF) ist hierfür ein Beispiel für eine mobile Fördertechnik. Je nach Grad der Autonomie könnte das FTF sich um den Auftrag bewerben und die Maschine bzw. der Transportauftrag entscheidet eigenständig, wer den Zuschlag für den Transport erhält [BLS+17].

Für FTF existieren erste Bestrebungen zur Standardisierung der Kommunikationsschnittstelle, aber auch diese haben Limitierungen. Beispielsweise können bei der Standardisierungsinitiative VDA 5050[1] durch den Verband der Automobilindustrie e.V. (VDA) Auftrags- und Statusdaten zwischen einer zentralen Leitsteuerung und FTF für intralogistische Prozesse ausgetauscht werden. Soll sich ein FTF eigenständig um einen Transportauftrag bewerben, so muss dieses sich seiner Fähigkeiten bewusst sein, bevor ein Angebot abgegeben werden kann. Hierfür wird zum einem die Selbstbeschreibung der Fähigkeiten eines FTF benötigt und zum anderen die formale Beschreibung eines Transports, also was genau transportiert werden soll. Beide Beschreibungen sind hierbei wichtige Komponenten für einen dezentral selbstorganisierenden Transport.

Voraussetzung für eine Interkonnektivität unter den Teilnehmern oder die dezentrale Entscheidungsfindung ist die Kommunikation zwischen diesen [MAS+18].

[1] VDA 5050 – Schnittstelle zur Kommunikation zwischen Fahrerlosen Transportfahrzeugen und einer Leitsteuerung

Während es Standardisierungsbestrebungen für zentralisierte Kommunikationssysteme gibt [TPS+19], sollte die Gelegenheit nicht verpasst werden, neue Kommunikationsansätze zu untersuchen. Schließlich soll die Anzahl an vernetzten Geräten weltweit jährlich um Milliarden wachsen [Sta16, Sta20]. Dies impliziert auch einen Anstieg der Vernetzung innerhalb von Fabriken. Mit einer steigenden Anzahl an Geräten steigt auch der Bedarf an einer nahtlosen Zusammenarbeit und Konnektivität. Die wachsende Anzahl von Geräten wird dazu führen [Sta18], dass neue Skalierungsmechanismen auf verschiedenen Ebenen benötigt werden. Dies betrifft z. B. die Handhabung des Datenverkehrs [AHK+18], die Verarbeitung von Daten [FBE+16] oder die logische Vernetzung von Komponenten. Während sich in der Industrie 4.0 Systeme dezentral organisieren sollen, erfolgt die Kommunikation meist über zentralisierte Broker bzw. Server. Fällt dieser zentralisierte Broker aus, kann das einen Ausfall des gesamten Systems nach sich ziehen. Zentralisierte Ansätze können Engpässe verursachen, haben eine mangelnde Koordination, besitzen veraltete Daten oder es bedarf großer Kapazitäten zur Speicherung von Informationen. Daher sind zentralisierte Ansätze ungeeignet für die Bewältigung von dynamischen Systemen [dRB14], sodass dezentralisierte Ansätze benötigt werden.

1.3 Zielsetzung

Diese Arbeit ist dem Thema der dezentralen Organisation des automatisierten Transports innerhalb eines CPPS gewidmet. Daher wird am Beispiel des selbstorganisierenden Materialflusses untersucht, ob sich die dezentrale Organisation unter Verwendung einer serverlosen Kommunikation realisieren lässt. Grundlage für einen selbstorganisierenden Materialfluss sind Bausteine wie die Selbstbeschreibung der Betriebsmittel bezüglich ihrer Fähigkeiten und Funktionen, die automatische Konfigurationen und Integrationen der Betriebsmittel und die autonome Aushandlung eines Betriebsmittels für die Ausführung einer gegebenen Aufgabe. Konkret handelt es sich bei einer Aufgabe bspw. um einen Transport und bei einem Betriebsmittel um ein Fahrerloses Transportfahrzeug. Mit der Zusammenführung der Selbstbeschreibung des Betriebsmittels und des Prozesses entsteht ein ganzheitliches Informationsmodell. Dieses dient als Grundlage für die Erstellung einer digitalen/virtuellen Entität/Repräsentation vom Materialfluss. Diese Basis erlaubt es, ein dynamisch (re-)konfigurierbares Transportsystem zu schaffen, in dem mehrere (verschiedene) Betriebsmittel (Ressourcen) zur Bewältigung von Transportaufgaben existieren. Der Materialfluss kann sich so selbst organisieren. Jede Transportaufgabe versucht sich hierbei eigenständig fertigzustellen, während direkt mit den Betriebsmitteln verhandelt wird. Der selbstorganisierende Materialfluss ist nur ein Beispiel, denn die entwickelten bzw. abgeleiteten Prinzipien lassen sich auch verallgemeinern und dienen nur der exemplarischen Umsetzung.

Ein weiterer wesentlicher Punkt bei der Realisierung eines solchen Systems ist ebenfalls die serverlose Kommunikation. Hierfür wird die inhärent gegebene Kommunikationsfähigkeit der oben genannten Betriebsmittel für die Vernetzung verwen-

det. Allerdings hat die konsequente Dezentralisierung, unabhängig von der Anwendung, einen erhöhten Kommunikationsbedarf zur Folge, nicht zuletzt, da sich die Betriebsmittel über die zu leistenden Aufgaben abstimmen müssen. Um das Potenzial von dezentral organisierter Kommunikation am Beispiel des selbstorganisierenden Materialflusses identifizieren und abschätzen zu können, muss die Frage beantwortet werden, wie diese mit der Anzahl an Teilnehmern skaliert.

Eine reale Umsetzung der Szenarien wäre mit enormen Kosten verbunden. Daher ist die simulationsgestützte Entwicklung eine wichtige Ergänzung in dieser Arbeit. Die Implementierung eines solchen selbstorganisierenden Materialflusses, bei der eine dezentral organisierte Kommunikation erfolgt, wird mithilfe eines Simulations-Frameworks für Rechnernetze evaluiert. Das verwendete Simulations-Framework bietet neben der Möglichkeit, verschiedenste Kommunikationstechnologien, -medien und -protokolle zu verwenden, auch eine flexible, schnelle und kostengünstige Integration. Ferner hilft das Simulations-Framework, die Skalierbarkeit und die Integration der Applikation bzw. der Kommunikation zu evaluieren. Durch die Durchführung von Simulationsstudien kann eine Aussage zur Performanz und zum Potenzial der Applikation und der dezentral organisierten Kommunikation gegeben werden.

1.4 Publikationen

Teile der hier dargelegten Forschungsergebnisse wurden durch den Autor im Vorfeld veröffentlicht. Diese veröffentlichten Publikationen werden im Folgenden aufgelistet und an den entsprechenden Stellen innerhalb der Arbeit referenziert.

1. Detzner, Peter und Salhofer, Peter – “Analysing FIWAREs Platform – Potential Improvements” in: Proceedings of the 53th Hawaii International Conference on Systems Sciences, Hawaii, 2020, DOI: 10.24251/HICSS.2020.809.

2. Detzner, Peter; Kirks, Thomas und Jost, Jana – “A Novel Task Language for Natural Interaction in Human-Robot Systems for Warehouse Logistics” in: 14th International Conference on Computer Science Education (ICCSE), Aug. 2019, S. 725–730, DOI: 10.1109/ICCSE.2019.8845336.

3. Detzner, Peter; Pose, Tim; Fumagalli, Luca und Matteucci, Matteo – “Towards a Plug and Play Architecture for a Material Flow Handling System” in: IEEE Conference on Open Systems (ICOS), Nov. 2019, S. 28–33, DOI: 10.1109/ICOS47562.2019.8975705.

4. Detzner, Peter; Gödeke, Jana und Bondorf, Steffen – “Low-Cost Search in Tree-Structured P2P Overlays: The Null-Balance Benefit” in: IEEE 46th Conference on Local Computer Networks (LCN) (LCN 2021). Edmonton, Canada, DOI: 10.1109/LCN52139.2021.9525004.

5. Krämer, Larissa; Althäuser, Rico; Detzner, Peter; Leveling, Jens; Brehler, Marius und ten Hompel, Michael – "Towards a Concept for Blockchain-based Cyber-Physical Production Systems" in: 17. Fachkolloquium der WGTL e. V., 2021, DOI: 10.2195/lj_Proc_kraemer_en_202112_01.

6. Detzner, Peter; Otto, Boris; Möller, Frederik und Kerner, Sören – "Introduction to the Minitrack on Open Platform Ecosystems in Logistics: Business Models and Technologies" in: Proceedings of the 55th Hawaii International Conference on Systems Sciences, Maui: Hawaii (Virtual Conference), 2022, DOI: 10.24251/HICSS.2022.596.

7. Lünsch, Dennis; Detzner, Peter; Ebner, Andreas und Kerner, Sören – "SWAP-IT: A Scalable and Lightweight Industrie 4.0 Architecture for Cyber-Physical Production Systems" in: IEEE 18th International Conference on Automation Science and Engineering (CASE), Mexico City, 2022, DOI: 10.1109/CASE49997.2022.9926665.

8. Detzner, Peter; Gödeke, Jana und Bondorf, Steffen – "Peer Discovery in Tree-Structured P2P Overlay Networks by Means of Connected Dominating Sets" in: IEEE 47th Conference on Local Computer Networks (LCN), Edmonton, Canada, 2022, DOI: 10.1109/LCN53696.2022.9843678.

9. Detzner, Peter; Ebner, Andreas; Hörstrup, Maximilian und Kerner, Sören – "PFDL: A Production Flow Description Language for Order-Controlled Production," in: IEEE 22nd International Conference on Control, Automation and Systems (ICCAS), Busan, Korea, 2022, DOI: 10.23919/ICCAS55662.2022.10003953.

10. Detzner, Peter; Brehler, Marius; Pecorella, Tommaso und Kerner, Sören – "Introduction to the Minitrack on Self-Adaptive Systems and Applications" in: Proceedings of the 56th Hawaii International Conference on Systems Sciences, Maui: Hawaii, 2022, DOI: https://hdl.handle.net/10125/103463.

1.5 Wissenschaftlicher Beitrag und Aufbau der Arbeit

Die hier vorliegende Arbeit hat sowohl einen wissenschaftlichen als auch einen praktischen Nutzen. Zum einen liefert die Arbeit einen Beitrag auf der wissenschaftlichen Seite zu verschiedenen Fragestellungen im Bereich der Selbstorganisation des Materialflusses und der hierfür entwickelten selbstorganisierenden Kommunikation. Mit dem Aufbrechen der bestehenden zentralisierten Strukturen und dem Wandel zu dezentral organisierten Systemen wird eine zunehmende Erhöhung der Verfügbarkeit, der Speicher- und der Rechenfunktionen in Geräten sowie auch neue Möglichkeiten für diese erschaffen, welche viel näher an der Produktion sind. Aus

praktischer Sicht werden die Daten somit nur noch lokal vorgehalten, ohne dass die Daten in eine Cloud übertragen bzw. aus dieser abgerufen werden müssen. Um nur einige weitere Vorteile zu nennen, werden unnötige Datenübertragungen vermieden, der Energieverbrauch wird minimiert, aber auch die Sicherheitsbedenken bzgl. Daten entfallen, da diese lokal vorgehalten werden.

Der mit dieser Arbeit angesprochene Adressatenkreis umfasst sowohl Wissenschaftler als auch die Mitarbeiter von erwerbswirtschaftlichen Unternehmen. Zu den letzteren zählen bspw. die Hersteller von Fahrerlosen Transportfahrzeugen, produzierende Unternehmen im Allgemeinen oder auch (System-)Integratoren, welche bei der Umsetzung einer I4.0-Applikation behilflich sind.

Die wissenschaftlichen Beiträge sind eng mit der Struktur der Arbeit verbunden, welche in sieben Kapitel gegliedert ist (vgl. Abb. 1.2). Zunächst wird in Kapitel 1 die Arbeit motiviert, ihre zugrunde liegende Problemstellung aufgeführt und die sich hieraus abgeleiteten Ziele vorgestellt.

Kapitel 2 befasst sich mit den Vorüberlegungen zum selbstorganisierenden Materialfluss. Hier erfolgt eine Betrachtung des Begriffs I4.0 und wie dieser im Kontext dieser Arbeit zu verstehen ist. Im Weiteren werden in diesem Kapitel auch ein Konzept bzw. eine Referenzarchitektur und die dafür benötigten Bausteine identifiziert. Im darauffolgenden Kapitel 3 wird die Vernetzung in cyberphysischen Produktionssystemen vorgestellt. Zum einen erfolgt neben der genaueren Untersuchung eines Teilnehmers auch die Betrachtung, welche Systemarchitekturen zur Kommunikation möglich sind, insbesondere in Bezug auf cyberphysische Produktionssysteme. Zum anderen werden nicht-funktionale Anforderungen an die Kommunikation abgeleitet und diskutiert. Dieses Kapitel schließt damit, welchen Einfluss das physikalische Medium auf die Kommunikation hat und welche Schlussfolgerungen aufgrund dessen getroffen werden.

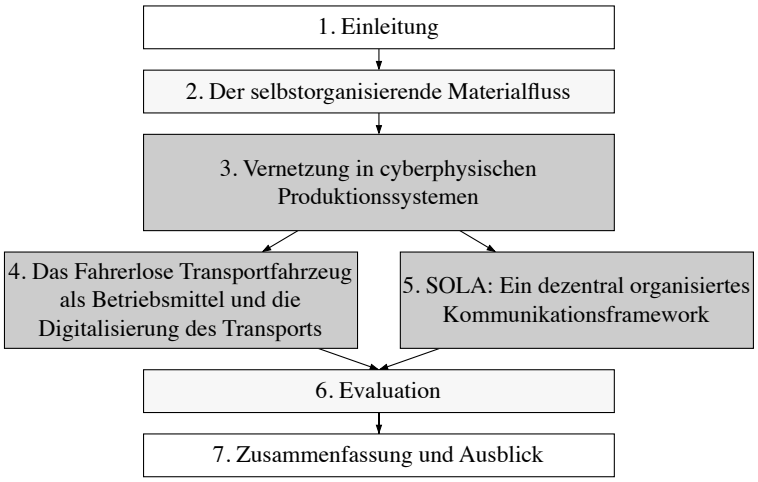


Abb. 1.2: Struktur der vorliegenden Arbeit

In Kapitel 4 wird das ganzheitliche Informationsmodell für die Intralogistik eingeführt. Dieses besteht aus einer Selbstbeschreibung des Betriebsmittels (hier: Fahrerloses Transportfahrzeug) und der digitalen Abbildung des Transports. Grundlage hierfür ist eine deklarative, domänenspezifische Beschreibungssprache. Bei der Selbstbeschreibung des Betriebsmittels handelt es sich konkret um ein Datenmodell. Anwendung kann dieses Datenmodell zur Ausführung eines cyberphysischen Produktionssystems erhalten, um Fahrzeuge zur Laufzeit diesem hinzuzufügen. Neben den benötigten Informationen für eine An- und Abmeldung können die angebotenen Statusinformationen u. a. für eine Optimierung verwendet werden. Die Überführung des Transports in eine virtuelle, einheitliche (formale) digitale Entität beinhaltet die Beschreibung von sequenziellen oder konkurrierenden Transporten, Randbedingungen und Ressourcenbeschränkungen, aber auch Möglichkeiten zur Synchronisation. Das digitale Abbild kann sowohl eine eigenständige Entität darstellen als auch in ein übergeordnetes System integriert werden.

Nach der Modellierung des Informationsmodells wird in Kapitel 5 eine plattformunabhängige, serviceorientierte Kommunikationsarchitektur vorgestellt. Diese unterstützt eine serverlose Kommunikation, welche in Abhängigkeit von der Anzahl der Teilnehmer skaliert. Neben der Suche nach Informationen, Services oder Entitäten unterstützt diese auch eine Verteilung von Ereignissen bzw. Nachrichten innerhalb einer Gruppe bzw. für alle Teilnehmer im Netzwerk. Dieser dezentral organisierte Ansatz ermöglicht, dass die Daten nur dort vorgehalten werden, wo sie auch benötigt werden. Darüber hinaus verbindet diese Architektur ein traditionelles Publish-Subscribe mit verteilten Abfragen, ohne dass eine zentrale Instanz benötigt wird.

In Kapitel 6 wird das intralogistische Informationsmodell und die entwickelte Kommunikationsarchitektur zusammengeführt und evaluiert. Dies erfolgt durch einen simulativen Aufbau zweier Anwendungsfälle, bei der u. a der hier entstandene Kommunikationsaufwand betrachtet wird. Ferner wird der hier erarbeitete Ansatz kritisch betrachtet.

Diese Arbeit schließt mit Kapitel 7, in dem die durchgeführten Arbeiten zusammengefasst und sich daraus ergebende Anschlussmöglichkeiten aufgezeigt werden.

Alle in dieser Arbeit erarbeiteten und entwickelten Ergebnisse wurden mit expliziten Referenzen der Literatur an den entsprechenden Stellen gekennzeichnet. Ferner wurden alle Abbildungen im Rahmen dieser Arbeit selbstständig erstellt. Abbildungen, welche eine Referenz zu einer Literatur haben, wurden an entsprechender Stelle gekennzeichnet.

Kapitel 2
Der selbstorganisierende Materialfluss

> "*Self-organization can be defined as the spontaneous creation of a globally coherent pattern out of local interactions.*"
>
> — Francis Heylighen

Um einen selbstorganisierenden Materialfluss im Kontext der Industrie 4.0 zu realisieren, muss dieser zunächst näher beleuchtet werden. Daher erfolgt zunächst eine Einordnung des Begriffs Industrie 4.0 und wie dieser in der vorliegenden Arbeit zu verstehen ist (vgl. Abschn. 2.1). Darauf erfolgt die nähere Betrachtung des Förderns und Transportierens von Dingen mittels Fahrerloser Transportfahrzeuge (vgl. Abschn. 2.2). Als Abschluss für dieses Kapitel werden die beiden Begriffe der Industrie 4.0 und des Förderns und Transportierens mithilfe von Fahrerlosen Transportfahrzeugen miteinander verbunden. Dies geschieht durch die Herausarbeitung eines Konzepts für den selbstorganisierenden Materialfluss und welche Bausteine für eine Realisierung benötigt werden (vgl. Abschn. 2.3).

2.1 Die Entwicklung der Industrialisierung

Edmond Cartwright erfand im Jahre 1784 einen Kraftstuhl, welcher mit Dampfkraft betrieben wurde. Diese dampfkraftbetriebene Webmaschine gilt als der Startschuss für die erste industrielle Revolution. Im Jahr 1870 begann die zweite industrielle Revolution, bei der die Massenproduktion und die Arbeitsteilung im Vordergrund standen. Diese wurde mit der Einführung der Elektrizität begleitet. Mit der Einführung der speicherprogrammierbaren Steuerung (SPS) im Jahr 1969 begann mit der Automatisierung von Fertigungsprozessen die dritte industrielle Revolution. Im Vordergrund standen dabei die Elektronik und die Informationstechnik. [Yül18]

Nach der Mechanisierung, der Elektrifizierung und der Automatisierung befinden wir uns gegenwärtig in der vierten industriellen Revolution (vgl. Abb. 2.1). Eckpfeiler sind hier die Vernetzung von Teilnehmern, die Transparenz von Infor-

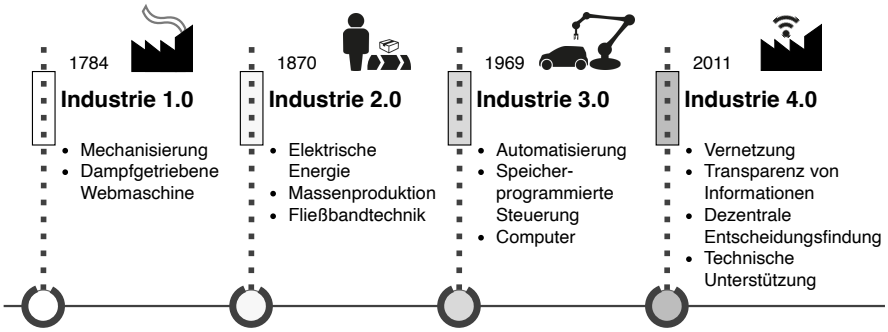


Abb. 2.1: Die vier industriellen Revolutionen und ihre Schwerpunkte, basierend auf [Yül18, HPO16]

mationen, die dezentrale Entscheidungsfindung und die technische Unterstützung. Die Vernetzung sieht vor, dass verschiedenste Maschinen, Sensoren und Geräte miteinander kommunizieren und interagieren können. Dabei werden standardisierte Schnittstellen verwendet. Durch die eben genannte Vernetzung verschmilzt die reale mit der virtuellen Welt. Ausgetauschte Informationen können in Kontext gebracht werden. Kontextbezogene Informationen sind die Grundlage für die weitere Entscheidungsfindung. Die dezentrale Entscheidungsfindung basiert auf der Vernetzung und der Informationstransparenz. Lokale Informationen werden mit globalen Informationen verknüpft. Teilnehmer agieren auf Grundlager dieser aggregierten Informationen. Die technische Unterstützung sieht vor, dass die immer komplexeren Prozesse beherrschbar bleiben. Diese Unterstützung ist insbesondere für den Menschen wichtig, da sich seine Rolle von der des Maschinenbedieners zu der des Problemlösers entwickelt. [HPO16]

Schlüsseltechnologien der vierten industriellen Revolution sind die Künstliche Intelligenz, Big Data, Robotik, Cloud und Fog Computing, das Internet der Dinge und die Mensch-Technik-Interaktion [AIM10, BtV14, tH20, BKR+16, Gt10]. All diese Technologien können interdisziplinär angewendet werden. So kann bspw. die Künstliche Intelligenz für Analysen im Bereich Big Data zur Qualitätssicherung [tH20, S. 262] oder zur vorausschauenden Wartung genutzt [WE16] werden. Aber auch der Mensch rückt mehr in den Vordergrund. Die Technik passt sich unter Berücksichtigung sozialer Regeln an den Menschen an [KJ20].

Die vierte industrielle Revolution wird begleitet von vielen verschiedenen Initiativen [APP19]. Die erste, die in diesem Kontext genannt wurde, ist die *Industrie 4.0* in Deutschland [ZLZ15]. Weitere bekannte Initiativen sind *Future of Manufacturing* [GD13] aus dem Vereinigten Königreich Großbritannien und Nordirland, Japans *Society 5.0* [Ish16], *Manufacturing USA* [Man] aus den Vereinigten Staaten von Amerika oder Frankreichs *Industrie du Futur* [Eur17b]. Diese Initiativen empfehlen neben der Standardisierung von Schnittstellen auch die Verwendung einer dienstorientierten Architektur (engl.: *service-oriented architecture*) [OAS06]. SOA soll bei der Dekomposition eines komplexen, monolithischen Systems in ein Öko-

system von einfachen und gut definierten Komponenten helfen [AIM10]. Die dienstorientierte Architektur ist ein Paradigma für die Organisation und Nutzung verteilter Fähigkeiten [OAS06]. Dabei bietet jede Komponente genau eine Funktionalität an. Diese wird in einem Dienst gekapselt und über einen Service angeboten, unabhängig davon, welches physikalische Gerät diesen Dienst ausführen wird [BtV14, S. 27 f.]. Komplexe Funktionalität wird durch die Verknüpfung von Diensten, die sogenannte Dienstekomposition, erreicht [AP07, MM04].

Durch die Anwendung von Informations- und Kommunikationstechnik (IKT) auf mechanische und elektronische Komponenten entsteht so ein cyberphysisches System (CPS) [tH20, S. 46]. Hierbei steht die Vernetzung von Objekten wie Sensoren, Aktoren, Maschinen, Gebäuden und Menschen im Vordergrund [BtV14, S. 15 f.]. Mit der Anwendung des CPS auf die Domäne Produktion entsteht dadurch das CPPS [Mon14, BtV14]. Logistikkomponenten wie Fahrerlose Transportfahrzeuge und Produktionsanlagen werden miteinander vernetzt. Diese Komponenten können sich selbst konfigurieren und (teilweise) dezentral selbstorganisieren und sind somit folglich flexibel [tH20, S. 14]. Die Logistik 4.0 ist somit als horizontale Verbindung in der Wertschöpfungskette anzusehen und folglich ein wichtiger Bestandteil der adaptiven Fabrik [Pla16a].

2.2 Fördern und Transportieren mittels Fahrerloser Transportfahrzeuge

Die innerbetriebliche Logistik, kurz Intralogistik, umfasst nach [Arn06, S. 1] die Organisation, Steuerung, Durchführung und Optimierung des innerbetrieblichen Materialflusses, der Informationsströme und des Warenumschlags in der Industrie, im Handel und in öffentlichen Einrichtungen.

Dabei werden mithilfe des Materialflusses Dinge gelagert, transportiert, zusammengeführt oder verteilt [HSD18, Vorwort]. Der Transport selbst ist allgemein als eine Fortbewegung von Gütern oder Personen mit technischen Mitteln anzusehen [HSD18, S. 125]. Transporte werden im idealen logistischen Raum durchgeführt, welcher leer, kontinuierlich und kinodynamisch ist und in dem keinerlei Lagerung von Objekten vorgesehen ist, sondern eine ständige Bewegung dieser [Roi22]. Die VDI-Richtlinie 2411 [Ing70] bezeichnet die Fortbewegung von Arbeitsgegenständen in einem System als *Fördern*. In [HSD18, S. 125] wird zwischen *Fördern* und *Transportieren* unterschieden. Findet das Transportieren innerbetrieblich statt, also innerhalb eines Gebäudes oder Werkes, so wird dies als *Fördern* bezeichnet. Dieser innerbetriebliche Transport zeichnet sich dabei besonders über eine Beförderung über eine kurze Distanz aus. Wird ein Gut oder eine Person über weite Entfernungen oder außerhalb eines Betriebes bewegt, so handelt es sich um einen außerbetrieblichen *Transport*.

Anwendung findet der Transport als Materialflussoperator u. a. in Stückgutsortier- und Kommissioniersystemen [HSD18, Kap. 5], beim Sortieren und Zwischenlagern oder in der Verteilung von Materialien in der Produktion [tSB11b, S. 45]. Trans-

porte können unter Zuhilfenahme technischer Geräte und Hilfsmittel, auch Fördermittel genannt, durchgeführt werden. Fördermittel können, wie in Abb. 2.2 dargestellt, in die zwei Hauptkategorien *Stetigförderer* und *Unstetigförderer* unterteilt werden. *Stetigförderer* zeichnen sich durch feste Auf- und Abgabestellen und eine kontinuierliche Arbeitsweise aus [HSD18], welche agentenbasiert realisiert werden kann [Lib11]. *Unstetigförderer* hingegen arbeiten nach dem Prinzip des Aussetzbetriebs. Der Aussetzbetrieb selbst kann dabei folgende Komponenten beinhalten: Lastfahrten, Leerfahrten (Anfahrt), Anschlussfahrten und Stillstandzeiten. Die Zeitanteile für die jeweilige Komponente können hierbei variieren. Unstetigförderer können die Lastaufnahme bzw. -abgabe meist nur an bestimmten, vorher definierten Orten durchführen. Im Gegensatz zu Stetigförderern besitzen Unstetigförderer eine hohe Anpassungsfähigkeit und sind zudem selten ortsfest, wie zum Beispiel Flurförderzeuge [VDI07].

Fahrerlose Transportfahrzeuge, eine Untermenge der Flurförderzeuge, hielten 1954 ihren Einzug in die Industrie [Ull15, S. 2]. Aufgrund ihrer Vielzahl von Sensoren, Aktoren und einer komplexen Mechanik gelten FTF dabei als das komplexeste technische Hilfsmittel, um einen Transport durchzuführen. Darüber hinaus müssen FTF ihre Umgebung wahrnehmen, aber auch die entsprechenden Sicherheitsanforderungen der Umgebung erfüllen [SK16]. Darüber hinaus können sich FTF automatisch bewegen und autonom handeln. Dabei kann die Lastaufnahme sowohl *automatisch* als auch *manuell* erfolgen. Die *automatische* Lastaufnahme bedeutet, dass ein FTF die Last eigenständig und ohne ein externes Zutun aufnehmen und abgeben kann. Bei der *manuellen* Lastaufnahme kann die Aufnahme bzw. die Abgabe der Last durch eine Kollaboration mit dem Menschen erfolgen. Ein Fahrerloses Transportfahrzeug bringt neben ökologischen auch ökonomische Vorteile mit sich. So sind die Transportschritte und deren Zeiten vorhersehbar und planbar [Ull15, S. 33].

In Abb. 2.3 ist das Zustandsdiagramm für ein FTF mit automatischer Lastaufnahme für einen einzelnen Transportauftrag dargestellt. Sobald das Fahrzeug einen Transportauftrag erhält, wird die Abholung initiiert und das Fahrzeug fährt zum Abholort. Nach der Ankunft am Abholort beginnt die Beladung durch Warenaufnahme. Nach erfolgreicher Beladung wird der Zielort für die Lieferung der Ware angefahren. Am Zielort selbst wird die Entladung durchgeführt. Nach erfolgreicher Entladung kann das FTF entweder eine Anschlussfahrt (z. B. zum Parkplatz)

Abb. 2.2: Vereinfachte Systematik der Fördermittel, basierend auf [HSD18, S. 129]

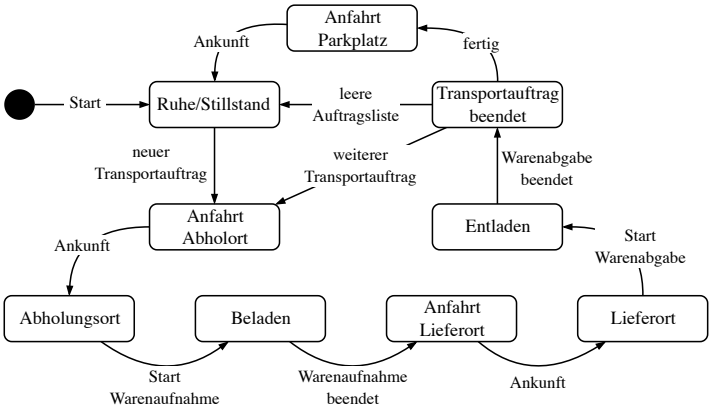


Abb. 2.3: Vereinfachtes Zustandsdiagramm eines Unstetigförderer für einen einzelnen Transportauftrag, basierend auf [HSD18, S. 130]

durchführen, einen neuen Transportauftrag abarbeiten oder an dem Ort verbleiben. Beim Verbleib an dem aktuellen Ort wird dieser jedoch blockiert. Ein neuer Transportauftrag kann bedeuten, dass ein neuer Ort angefahren werden muss, an dem die Abholung erfolgen soll, oder das FTF befindet sich bereits an dem Ort. Im letzten Fall würde das FTF mit der Beladung direkt beginnen.

Fahrerlose Transportfahrzeuge können wie in Abb. 2.4 dargestellt sowohl zentral als auch dezentral organisiert werden. Eine zentrale Organisation (vgl. Abb. 2.4a) sieht vor, dass die FTF durch eine übergeordnete Instanz wie die Leitsteuerung koordiniert werden [Ull15, S. 122 ff.]. Ein Fahrerloses Transportsystem (FTS) besteht somit aus einer Leitsteuerung und einer Flotte an FTF. Die Leitsteuerung kennt alle relevanten Informationen (bspw. Auftragsdauer, Auftragswarteschlange, ...) der FTF und hat somit einen globalen Informationszugriff. Folglich kann diese zentrale Instanz dem geeignetsten FTF einen Auftrag zuweisen. Zentralisierte Ansätze benötigen eine stabile und vorhersehbare Umgebung [MIA17]. Fällt eine Kom-

(a) Die Leitsteuerung verwaltet die Fahrerlosen Transportfahrzeuge

(b) Die Intelligenz wandert in die Fahrerlosen Transportfahrzeuge, sodass diese autonomer werden und miteinander interagieren können

Abb. 2.4: Zentrale und dezentrale Organisation von Fahrerlosen Transportfahrzeugen, basierend auf [DRVD20]

ponente aus, so muss eine gesamte Neuplanung des Prozesses angestoßen werden [Duf90]. Fällt jedoch die zentrale Instanz aus, so lässt sich eine Flotte von FTF nicht mehr koordinieren.

Damit Anforderungen wie Skalierung, Flexibilität oder Robustheit [Mon14] innerhalb der Industrie 4.0 in Bezug auf FTF erfüllt werden können, werden neue Ansätze zur Organisation benötigt. Eine Möglichkeit ist die dezentrale Organisation (vgl. Abb. 2.4b) von FTF [FdS$^+$21]. Dabei verteilt sich die Intelligenz auf die einzelnen FTF, sodass sich jedes FTF selbstständig verwaltet. In der Industrie sind zentral organisierte Systeme geläufiger als dezentral orientierte [DRVD20]. In Tabelle 2.1 ist ein Vergleich von zentralen und dezentralen Architekturen zur Steuerung von FTF dargestellt. Die Tabelle berücksichtigt u. a. die aktuelle Verbreitung in der Industrie, die Algorithmen zur Handhabung der Organisation der FTF und welches Optimum die zentrale oder dezentrale Organisation bedient. Weitere wichtige Punkte sind die Skalierung der Systeme, die Systemkomplexität oder die Robustheit in einer dynamischen Umgebung. Letzteres beschreibt, wie robust sich das System in Fehlerfällen oder in stetig wechselnden Umgebungen, z. B. mit Hindernissen, verhält. Sind alle Fahrzeuge in einer Menge von FTF hinsichtlich ihrer Eigenschaften (z. B. Fahrzeugabmessungen oder Ladungsträger) gleich, so handelt es sich um eine homogene Flotte. Existieren in der Flotte verschiedene Typen, so ist diese Flotte heterogen.

In der wissenschaftlicheren Literatur wird häufig zwischen einem FTF und einem autonomen mobilen Robotern (AMR) unterschieden [OST$^+$20, ATA$^+$21, KÇP17, FdS$^+$21]. Im Gegensatz zu FTF benötigen AMR, z. B. in einem CPPS, keine zusätzliche Infrastruktur (optische Marker, Magnetstreifen usw.) und haben eine erhöhte Autonomie in der Wahrnehmung der Umgebung. Im Rahmen dieser Arbeit erfolgt keine trennscharfe Unterscheidung zwischen einem FTF und einem AMR und daher werden diese Begriffe als äquivalent angesehen.

Tabelle 2.1: Vergleich der zentralen und dezentralen Organisation von Fahrerlosen Transportfahrzeugen, basierend auf [DRVD20]

Kriterium	**Zentral**	**Dezentral**
Verbreitung in der Industrie	weit	kaum
Algorithmen	bekannt	bekannt
Informationszugriff	global	lokal
Optimum	globales	lokales
Skalierung	klein	groß
System-Komplexität	einfach	komplex
Robustheit in dynamischen Umgebungen	gering	hoch

2.3 Logistik 4.0: Selbstorganisation in der Intralogistik

Um mit der klassischen Automationspyramide zu brechen (vgl. Abb. 1.1) und eine Fabrik in ein selbstorganisierendes System zu transformieren, muss der Begriff der Selbstorganisation näher beleuchtet werden. Ein selbstorganisierendes System besteht aus einer Vielzahl unabhängiger miteinander interagierender Teilnehmer, wobei jeder Teilnehmer nur eine begrenzte Sicht hat [ST09]. Kein Teilnehmer hat folglich eine globale Sicht auf das System. Durch die Interaktion der Teilnehmer miteinander entsteht eine Selbstorganisation des Systems von unten nach oben heraus [Hey99]. Häufig ist die Anpassungsfähigkeit der Teilnehmer bzw. des Systems eine implizite Annahme, bei der es sich um eine emergente Eigenschaft handeln kann [dGM+13, S. 7]. Die Anpassungsfähigkeit eines Systems bedeutet, dass sich dieses und somit auch seine Teilnehmer an eine sich ständig wechselnde Umgebung anpasst [Hey99]. Der externe Einfluss auf das System wirkt sich zunächst auf dieses aus und anschließend auch auf die einzelnen Teilnehmer von oben nach unten. Von den Teilnehmern wird erwartet, dass diese kollaborativ arbeiten, um das Ziel des Systems zu erreichen [EM13]. Ein Ziel dieser Arbeit ist die Ermöglichung der Selbstorganisation, daher soll für diese Arbeit gelten:

Selbstorganisation
Selbstorganisation setzt eine konsequente Dezentralisierung voraus. Es existiert eine Menge von unabhängigen Teilnehmern. Jeder Teilnehmer kennt seinen eigenen Zustand und besitzt die Fähigkeit, mit anderen Teilnehmern zu kommunizieren und zu interagieren. Teilnehmer können entweder ihre eigenen Interessen, aber auch ein gemeinsames Ziel verfolgen, sodass ein globaler Zustand erreicht wird. Dabei hat kein Teilnehmer ein globales Wissen über alle Teilnehmer oder die Ziele.

Die Dezentralisierung bezieht sich auf den Punkt, wo die Entscheidung getroffen wird. In einem zentralisierten System trifft eine zentrale Instanz die Entscheidung, in einem dezentralisierten System hingegen trifft jede Instanz eine eigenständige Entscheidung. Durch eine Dezentralisierung des Systems und die Umwandlung in selbstorganisierende Teilnehmer entstehen einige Vorteile. Hierbei handelt es sich u. a. um die lose Kopplung von Teilnehmern und somit Offenheit zu Erweiterbarkeit, die Sicherstellung der Zuverlässigkeit des Systems bei Ausfall von Teilnehmern, die Performanz hinsichtlich der parallelen Ausführung von Aufgaben, die Flexibilität bei Anpassungen und Änderungen aufgrund verschiedenster Teilnehmer, die geringeren Kosten bei Änderungen und die Verteilung der Teilnehmer. Dem gegenüber stehen hier die Nachteile hinsichtlich der zusätzlichen Kommunikation für die Interaktion, die verstreute Verteilung von Informationen über (alle) Teilnehmer, die lokalen Optima bei der Entscheidungsfindung aufgrund des begrenzten Wissens eines Teilnehmers, die Sicherstellung der Sicherheit, ein ungewolltes, chaotisches Verhalten und die Erhöhung der Komplexität im Vergleich zu zentral organisierten Systemen. [MVD+15, NFH10]

Wird nun die Selbstorganisation auf den Materialfluss bzw. den Transport (vgl. Abschn. 2.2) angewendet, so ergibt sich folgende Definition für den weiteren Verlauf dieser Arbeit:

Selbstorganisierender Materialfluss im Kontext der Industrie 4.0
Der selbstorganisierende Materialfluss setzt eine konsequente Dezentralisierung voraus. Es existiert eine Menge von unabhängigen Teilnehmern, wie z. B. ein Transportauftrag oder ein Fahrerloses Transportfahrzeug. Für jeden Teilnehmer existiert ein digitales Abbild, welches seinen eigenen Zustand kennt und die Fähigkeit besitzt, mit anderen Teilnehmern zu kommunizieren und zu interagieren. Fahrerlose Transportfahrzeuge bieten Transportdienste an. Jeder Transportauftrag steht für sich selbst und versucht, sich eigenständig fertigzustellen. Ein Transportauftrag verhandelt dabei mit Fahrerlosen Transportfahrzeugen, bis der Transportauftrag einem FTF die Zuweisung des Transports gibt. Eine nachträgliche Änderung der Zuweisung ist ebenfalls möglich.

Diese Definition ist bewusst so formuliert, dass es sich im Folgenden um die Entitäten *Transportauftrag* und das *Fahrerlose Transportfahrzeug* handelt. Ferner existiert kein übergeordnetes System mit einer globalen Sicht, welches die Zuweisung von Transporten an Teilnehmer (hier FTF) übernimmt. Jedoch besteht die Möglichkeit, einen Teilnehmer zu definieren, welcher eine (fast) globale Sicht auf eine Klasse von Teilnehmern hat. In dem selbstorganisierenden Materialfluss ist hervorzuheben, dass jeder Transportauftrag eigenständig ist und versucht, sich selbstständig fertigzustellen. Dabei liegt der Fokus des Transportauftrags immer auf den eigenen Interessen (lokales Optimum). So kann dieser Transportauftrag, welcher bspw. den Transport einer Palette repräsentiert, mit beliebig vielen FTF über den Transport verhandeln. Im Anschluss entscheidet sich der Transportauftrag für ein FTF, welches den Warentransport übernimmt [BLS+17]. Die hier getroffenen Entscheidungen basieren auf den verfügbaren Informationen des Transportauftrags und des FTF. Dies kann unter Umständen nicht zu einem globalen Optimum führen, da der Transportauftrag oder das Fahrzeug nur ein lokales und somit beschränktes Wissen haben [MIA17, DRVD20]. Ein Transportauftrag könnte sich selbst höher priorisieren und somit die Auftragsreihenfolge eines FTF ändern, was wiederum einen Einfluss auf die globale Reihenfolge aller Transporte und somit auf die Produktion haben kann. Dieses Szenario setzt den Transportauftrag in den Fokus und beschreibt, wie dynamisch sich der Transport und die verfügbaren Ressourcen organisieren. Im Kontext der Produktion handelt es sich hierbei um eine auftragsgesteuerte Produktion (engl.: *order-controlled production*) [Pla16a]. Eine Anwendung des Paradigmas OCP auf die Logistik ermöglicht die auftragsgesteuerte Logistik (engl.: *order-controlled logistic*).

In Abb. 2.5 ist die Überführung des bisherigen Materialflusses (links) in einen selbstorganisierenden Materialfluss, bestehend aus einer Anzahl aus dezentralen Teilnehmern (rechts) innerhalb eines CPPS dargestellt. Während in dem bisherigen System das Enterprise Resource Planning (ERP) direkt mit dem Flottenmanagementsystem (FMS) interagiert (linker Teil), entfällt dies in einem

CPPS (rechts). Hier zerbricht das FMS in eine Menge an dezentralen Teilnehmern. Jedes Fahrzeug steht für sich selbst, sodass die zentrale Instanz nicht benötigt wird.

Ein CPPS zeichnet sich dadurch aus, dass die reale Welt mit der virtuellen Welt verschmilzt [Mon14]. Es existiert folglich für jeden Transportauftrag und für jedes FTF eine virtuelle Repräsentation bzw. ein digitales Abbild innerhalb des CPPS (vgl. Kapitel 4). Dabei werden folgende Voraussetzungen für eine Selbstorganisation des Transports gestellt: Der Transport muss sich selbst wahrnehmen und somit seine Anforderungen kennen. Zusätzlich muss der Transport auch seine Umgebung wahrnehmen. Dies bedeutet, dass der Transportauftrag z. B. FTF im CPPS auffinden kann. Durch die Interaktion erfolgt dann die Zuweisung. Dies ist nur ein Beispiel, denn neben den FTF können auch andere Entitäten, wie bspw. Menschen oder Stetigförderer, die Transporte ausführen. Allerdings handelt es sich bei den FTF um das komplexeste technische Hilfsmittel, um einen Transport durchzuführen. Lässt sich der Transport und folglich der Materialfluss mithilfe eines FTF organisieren, so können auch Entitäten damit oder mit bestimmten Anpassungen bedient werden.

Im technischen Umfeld wird die Selbstorganisation häufig mit agentenbasierten Ansätzen realisiert [NFH10]. Agentenbasierte Ansätze gehören zu dem Forschungsfeld der Verteilten Künstlichen Intelligenz, einem Teilgebiet der Künstlichen Intelligenz. Komplexe Probleme werden in Teilprobleme aufgeteilt, verteilt, gelöst, zusammengeführt und die Lösung betrachtet [BG88]. Agenten zeichnen sich durch Autonomie, soziale Fähigkeit, Reaktionsfähigkeit und Proaktivität aus [WJ95]: Die Autonomie eines Agenten ist dadurch gekennzeichnet, dass dieser ohne die Zuhandlung eines Menschen handelt, seinen eigenen Zustand kennt und auch Aktionen ausführen kann [Cas95]. Die soziale Fähigkeit ist durch die Interaktion der Agenten untereinander gegeben [GK94]. Die Reaktionsfähigkeit ermöglicht die Wahrnehmung der Umgebung und die Anpassung an diese [WJ95]. Die Proaktivität impliziert, dass der Agent die Initiative übernimmt und nicht nur lediglich auf die Änderungen in seiner Umgebung reagiert [WJ95].

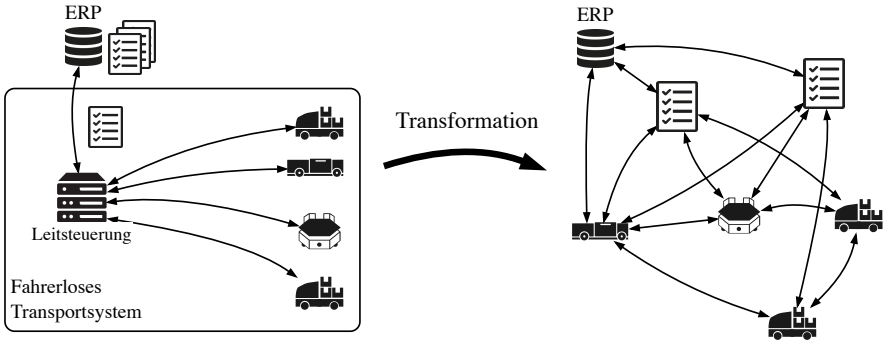


Abb. 2.5: Überführung des bisherigen Materialflusses (links) in einen selbstorganisierenden Materialfluss innerhalb eines cyberphysischen Produktionssystems (rechts)

In der nachfolgenden Abb. 2.6 ist eine modulare, agentenbasierte Architektur dargestellt [LDE+22]. Die vertikalen Blöcke sind nach ihrer Funktionalität gruppierte Module (hier: *ERP-Modul*, *Materialfluss-Modul* und *FTF-Modul*). Das Konzept des Moduls wird nun an Beispielen des Fahrerlosen Transportfahrzeugs und des Transportauftrags erläutert. Das FTF-Modul repräsentiert die Möglichkeiten, Dinge mithilfe von Fahrerlosen Transportfahrzeugen zu transportieren. Innerhalb des Moduls existiert eine Vielzahl an logischen FTF-Agenten. Jeder logische Agent repräsentiert im Folgenden ein Fahrerloses Transportfahrzeug (1:1). Dass ein logischer Agent auch mehrere FTF (1:n) repräsentiert, ist nicht ausgeschlossen. Dieser logische Agent kann u. a. auf dem FTF ausgeführt werden oder in einer dedizierten Virtualisierungsumgebung wie der (Edge-)Cloud. Logische Agenten repräsentieren Assets, welche entweder physisch oder logisch sein können [HHH+19]. Ein Beispiel für ein physisches Asset wäre ein Fahrerloses Transportfahrzeug; ein Beispiel für ein logisches Asset wäre ein Transportauftrag. Für die Kommunikation zwischen dem logischen Agenten und dem FTF kann ein eigener Kommunikationskanal (proprietäre Kommunikation) existieren. Über diesen erhält der logische Agent auch die Beschreibung seiner Fähigkeiten (Asset-Informationsmodell), sodass hieraus die entsprechenden Dienste innerhalb des CPPS angeboten werden können. Dieser Kommunikationskanal kann auch genutzt werden, um bereits vorhandene Systeme anzubinden.

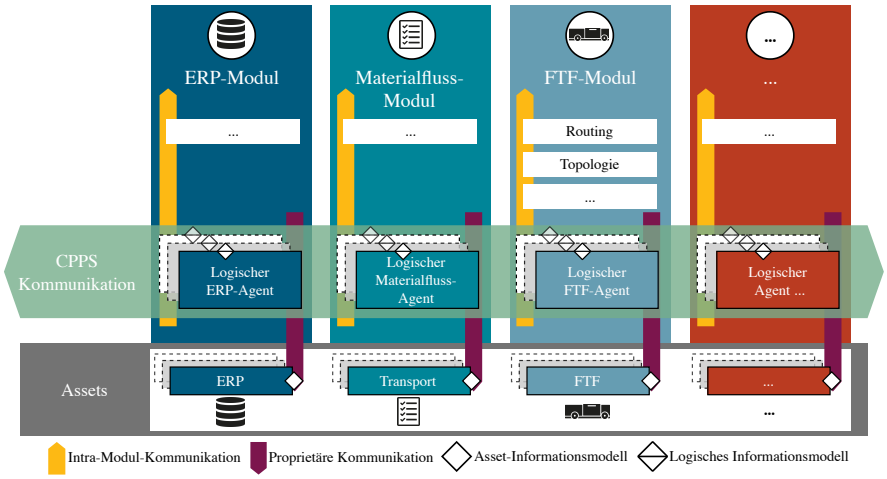


Abb. 2.6: Dezentralisierung der Applikation durch eine konsequente Modularisierung, basierend auf [LDE+22]

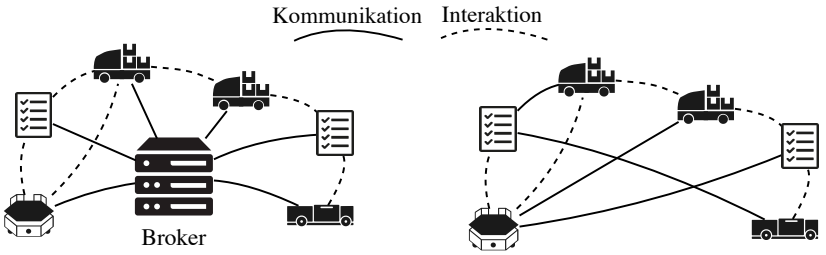
Der logische FTF-Agent bietet u. a. den Service *Transport* an. Damit andere Agenten diesen auch wahrnehmen können, wird eine offene und einheitliche Schnittstelle (logisches Informationsmodell) benötigt. Diese Schnittstellen können von anderen logischen Agenten (z. B. logischer Transportauftrags-Agent) anhand der CPPS-Kommunikation gefunden und genutzt werden. Es besteht auch die

Möglichkeit, dass die logischen Agenten auch noch weitere interne Komponenten benötigen. So kann der logische FTF-Agent eine Topologie oder ein Routing benötigen. In der hier dargestellten Abbildung werden diese Komponenten und ihre Funktionalitäten allerdings nicht von anderen Modulen und deren logischen Agenten benötigt. Daher sind diese nur über einen internen Kommunikationskanal erreichbar (Intra-Modul-Kommunikation). Ist es gewünscht, dass diese Funktionalität auch anderen Modulen bzw. logischen Agenten angeboten werden kann, so benötigen diese auch einen logischen Agenten für die Repräsentation. Um eine Flexibilität und Skalierbarkeit zu erhalten, ist es wichtig, dass sich ein neues FTF einfach an- und abmelden kann. Bei einer Anmeldung propagiert das FTF sein Informationsmodell, welches sich und seine Fähigkeiten widerspiegelt. Dieses Datenmodell kann auch verwendet werden, wenn ein logischer Transportauftrags-Agent nach möglichen Transportdiensten sucht. Während das FTF-Modul Dienste (hier: *Transport*) anbietet, ist der Konsument dieser Dienste das Materialfluss-Modul bzw. seine Agenten (logischer Materialfluss-Agent). Der logische Transportauftrags-Agent versucht sich eigenständig fertigzustellen, und sucht sich einen entsprechenden Transportdienst [NFH10, Pla16a, BLS+17]. Dabei bedarf es einer formalen Beschreibung der Transportaufgaben. An dieser Stelle erhält eine domänenspezifische Sprache ihren Einsatz (vgl. Abschn. 4.3).

Ein Transportauftrag kann eine einfache, unabhängige, augenblicklich auszuführende Aufgabe sein, für die lediglich nur eine Ressource (hier: FTF) benötigt wird. Allerdings kann ein Transportauftrag auch eine zeitliche Vorausplanung benötigen, bei der komplexe, zusammenhängende, mehrstufige Transportabfolgen erledigt werden sollen, für die diverse Ressourcen eines heterogenen Teams erforderlich sind. Die Beschreibung von (Transport-)Aufgaben kann dabei Folgendes umfassen [GM04]: Ein Roboter kann genau eine Aufgabe zu einem Zeitpunkt ausführen (engl.: *Single-Task Robots (ST)*) oder ein Roboter kann mehrere Aufgaben zu einem Zeitpunkt ausführen (engl.: *Multi-Task Robots (MT)*). Jede Aufgabe benötigt genau einen Roboter zur Ausführung (engl.: *Single-Robot Tasks (SR)*) oder eine Aufgabe benötigt mehrere Roboter, um diese durchzuführen (engl.: *Multi-Robot Tasks (MR)*). Die Aufgaben die erledigt werden sollen, sind unabhängig von einander und müssen unmittelbar ausgeführt werden (engl.: *Instantaneous Assignment (IA)*) oder die Aufgaben haben Abhängigkeiten untereinander und setzen eine entsprechende Planung voraus (engl.: *Time-Extended Assignment (TA)*). Kommen neue Teilnehmer nach der Zuweisung einer Aufgabe an einen Teilnehmer hinzu, so kann die Aufgabe sich mit den Teilnehmer auch über eine (Neu-)Zuweisung abstimmen [KSD13]. Hieraus resultiert, dass sich die Reihenfolge der Auftragsabarbeitung bei einem FTF ändert, und vermutlich die Start-, die End- sowie die Durchführungszeit für den Transport selbst. Auch diese Anpassungen bzw. Änderungen müssen zwischen den Teilnehmern ausgehandelt und akzeptiert werden. Diese Arbeit widmet sich auf die Beschreibung von Aufgaben der Klasse *SR-ST-TA*.

Die bisher beschriebene Selbstorganisation des Materialflusses ist getrieben durch die Dezentralisierung der Applikation und die Erhöhung der Autonomie einzelner Teilnehmer. Wie eingangs erwähnt, hat eine konsequente Dezentralisierung einen

erhöhten Kommunikationsbedarf zur Folge. Neben der steigenden Anzahl der Teilnehmer innerhalb eines CPPS bedarf es auch einer Selbstorganisation der Kommunikation. Bei einem CPPS mit einem zentralen Broker erfolgt die benötigte Kommunikation für die Interaktion ausschließlich über diesen (vgl. Abb. 2.7a). In einer dezentral organisierten Kommunikation hingegen existiert diese zentrale Instanz nicht. Hier ist die Kommunikation inhärent durch die Teilnehmer und ihre Kommunikationsfähigkeit realisiert (vgl. Abb. 2.7b). Die Teilnehmer können durch ihre Kommunikationsfähigkeit in Interaktion treten. Jeder Teilnehmer hat dabei nur eine (begrenzte) Anzahl an Verbindungen zu anderen Teilnehmern, welche zur Kommunikation verwendet werden. Durch die Aggregation der lokalen Informationen einzelner Teilnehmer entsteht so ein globales Wissen. Hier haben die Teilnehmer ein gemeinsames Verständnis bzgl. der Kommunikation und können somit nach anderen Teilnehmern und deren Fähigkeiten suchen und mit diesen in Interaktion treten.



(a) Jegliche Kommunikation erfolgt über den zentralen Broker

(b) Jeder Teilnehmer kennt nur eine Teilmenge an anderen Teilnehmern. Durch die Aggregation der Information aller Teilnehmer kann ein globales Wissen erzeugt werden

Abb. 2.7: Überführung der zentral organisierten Kommunikation in eine dezentrale, selbstorganisierende Kommunikation

Kapitel 3
Vernetzung in cyberphysischen Produktionssystemen

> "*A distributed system is a collection of independent computers that appears to its users as a single coherent system.*"
>
> — Andrew S. Tannenbaum & Marten Van Steen

In diesem Kapitel werden die für die vorliegende Arbeit notwendigen Grundlagen vorgestellt, notwendige Begriffe in Bezug auf das cyberphysische Produktionssystem definiert und Zusammenhänge zwischen diesen herausgearbeitet. Zunächst wird darauf eingegangen, was genau ein Teilnehmer im Kontext eines CPPS ist (vgl. Abschn. 3.1). Teilnehmer, auch Entitäten genannt, können entweder sich selbst repräsentieren oder eine Komposition verschiedenster Entitäten bilden. Neben der Definition eines Teilnehmers wird auch eine Quantifizierung durchgeführt.

Weitere wichtige Gegenstände dieses Kapitels sind die Kommunikation und die Möglichkeiten zur Vernetzung innerhalb des CPPS. Konkret handelt es sich um zentrale und dezentrale Strukturen sowie die damit verbundenen nicht-funktionalen Anforderungen. Dieses Kapitel schließt mit einer Diskussion darüber, ob ein CPPS zentral oder dezentral organisiert werden sollte und welchen Einfluss die physikalische Schicht auf die Wahl der Kommunikation hat.

3.1 Entitäten in einem cyberphysischen Produktionssystem

Bei der Transformation einer Fabrik in ein CPPS werden Technologien aus verschiedenen Disziplinen vereint [Mon14], um die Teilnehmer der realen Welt mit den Teilnehmern der informationsbasierten digitalen Welt zu vernetzen. In einer Fabrik kann ein Teilnehmer dabei u. a. einen Menschen, Maschinen, Sensoren, Aktoren

oder auch logische Komponenten wie eine Software repräsentieren. Innerhalb vom Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0) wird dies als *Asset* bezeichnet [HHH+19]. Ein *Asset* kann dabei eine physische oder logische Komponente sein. Jedoch ist die Differenzierung nicht trennscharf. *Assets* können auch als Teil des Unternehmens gesehen werden, vor allem dann, wenn diese *Assets* auch einen tatsächlichen Wert beitragen [Pla20]. Zamfirescu et al. [ZPG+14] sehen in ihrem anthropozentrischen cyberphysischen System [ZPS+13] den Menschen im Zentrum. Dabei besteht dieses System aus physikalischen, virtuellen oder menschlichen Komponenten [ZPG+14]. Hierauf basiert die von Berger et al. [BHH21] entworfene Taxonomie zur Klassifizierung von Entitäten innerhalb eines CPPS.

Ein Fokus dieser Arbeit ist innerhalb eines CPPS die Vernetzung von Teilnehmern und die Kommunikation untereinander. In einer Vielzahl unterschiedlicher Literaturquellen existieren verschiedene Definitionen für einen Teilnehmer innerhalb eines CPPS. Zur Vermeidung einer Verwirrung nimmt sich der Autor die Freiheit, eine Definition eines Teilnehmers in den eigenen Worten zu definieren. Diese Definition ist speziell für den Kontext dieser Arbeit zu verstehen.

Definition Entität
Eine Entität ist eine physische oder logische Komponente, welche in Interaktion mit anderen Entitäten innerhalb eines cyberphysischen Produktionssystems treten kann. Die Interaktion erfolgt durch den Austausch von Informationen mithilfe einer drahtlosen oder drahtgebundenen Kommunikation. Ferner kann eine Entität auch eine andere Entität oder eine Vielzahl an weiteren Entitäten repräsentieren.

Anhand der folgenden zwei Beispiele soll die Definition einer Entität verdeutlicht werden.

Eine physische Komponente kann zum Beispiel ein Stetigförderer sein, wie in Abb. 3.1 dargestellt. Dieser Stetigförderer hat die Form eines Förderbands mit einer Loopstruktur, mit zwei Einschleusbereichen (E_1 und E_2) und zwei Ausschleusbereichen (A_1 und A_2). Zum einen besteht die Möglichkeit, das Förderband als eine eigenständige Entität abzubilden. Andere Entitäten kommunizieren mit der Entität *Stetigförderer*, an welcher Stelle ein Gut ein- oder ausgeschleust werden soll. Zusätzlich besteht die Möglichkeit, dass sowohl die Einschleusbereiche E_1 und E_2 als auch die Ausschleusbereiche A_1 und A_2 als einzelne Entitäten in einem CPPS existieren. Hier kann das Gut, welches ebenfalls eine Entität ist, mit dem entsprechenden Ein- oder Ausschleusbereich kommunizieren und interagieren.

Als Beispiel für eine logische Komponente kann ein Produktionsauftrag gesehen werden. Dieser Produktionsauftrag kann in Form eines Vorranggraphs [WD06, S. 163] abgebildet werden [BLS+17]. Der Vorranggraph G besteht aus einer Menge an Teilaufgaben (T) und Abhängigkeiten (A). Formal lässt sich der Vorranggraph beschreiben als $G = (T,\ A)$. Eine Teilaufgabe repräsentiert einen Knoten im Graphen, während die Abhängigkeit der Reihenfolge der abzuarbeitenden Teilaufgaben als Kante repräsentiert wird. Während sich der Produktionsauftrag eigenständig fertigstellt, können Teilaufgaben an andere Entitäten ausgeschrieben werden. Dieser Produktionsauftrag hat nicht zwingend eine Verbindung zu einem physischen Gerät

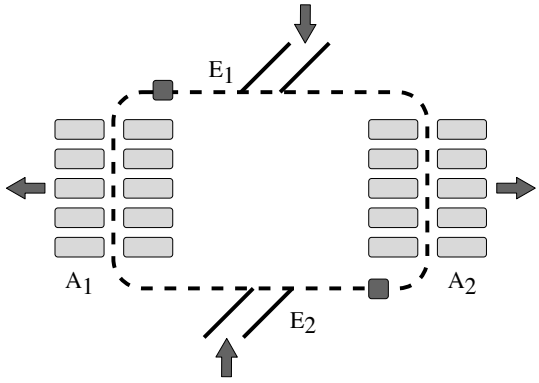


Abb. 3.1: Ein Förderband in einer Loop-Topologie mit zwei Einschleusebereichen E_1 und E_2 und zwei Ausschleusebereichen A_1 und A_2, basierend auf [HSD18, S. 169]

wie zu einer Maschine oder einem FTF. Allerdings kann dieser Produktionsauftrag ein Produkt oder einen Gegenstand repräsentieren.

3.1.1 Hierarchie und Emergenz

Das CPPS besteht aus einer Vielzahl an strukturellen und funktionellen Entitäten. Entitäten können selbst auch Unternehmensgrenzen überspannen, wenn ein Kunde den Zustand seines Produktionsauftrags abfragen will. Zusätzlich können sich Entitäten über Hierarchieebenen hinweg miteinander vernetzen, sodass eine Kommunikation zwischen allen Teilnehmern stattfinden kann [HHH+19, Mon14]. Die Verwendung von dezentral-organisierten Diensten hebt die Begrenzung von verschiedenen Stufen bzw. einer Hierarchie auf [Ver13]. Es sind lediglich diejenigen Teilnehmer miteinander verbunden, welche an einem Datenaustausch untereinander interessiert sind. Diese Verbindungen werden genutzt, um eine Entscheidung zu treffen. Unter Umständen kann es sein, dass diese Entscheidung für die entsprechende Komponente suboptimal ist, jedoch einem globalen Ziel dient [BFK+14, DRVD20]. Somit werden Entscheidungen nicht zentral getroffen, sondern dezentral durch eine Aushandlung unter den Teilnehmern [WH07, S. 7ff.].

Dennoch besteht die Option, dass durch die Selbstorganisation der Entitäten innerhalb eines CPPS ein explizites Level an Organisation oder Hierarchie gewollt wird oder entsteht. Zur Verdeutlichung wird zunächst ein Beispiel der Biologie gewählt. In der Biologie wird die Hierarchie wie folgt beschreiben: Ein Atom ist das kleinste chemische Element. Ein Wasserstoffatom ist die kleinste Einheit an Wasserstoff. Werden nun zwei Wasserstoffatome mit einem Sauerstoffatom verbunden, so entsteht ein Wassermolekül. Die Eigenschaften eines Wassermoleküls sind andere als die Eigenschaften eines einzelnen Sauerstoff- oder Wasserstoffatoms. Bilden sich neue Eigenschaften durch die Wechselwirkung der Teilnehmer,

so entsteht eine Emergenz. Dies lässt sich weiterführen. Eine Zelle wird geformt durch die Verbindung verschiedener Atome und Moleküle. Diverse Zellen wiederum formen ein Gewebe mit neuen Eigenschaften. Zusammengesetztes funktionales Gewebe wiederum formt ein Organ. Bei Lebewesen werden die biologischen Funktionen durch die Zusammensetzung von Gewebe und diversen Organen ausgeführt. Zusammen bilden diese ein organisches System mit neuen Eigenschaften, welches aus der Emergenz der Teilnehmer hervorgeht. [SMM+18, S. 6 f.]

Die Komplexität der niedrigsten Stufe (des Atoms) in der Hierarchie der Biologie ist gering. Mit der Integration verschiedener Stufen und der Wechselwirkung untereinander können neue, komplexere Stufen erschaffen werden [Lüt17, S. 143 f.]. Die Anzahl an Entitäten nimmt mit jeder neuen Stufe ab. Dies resultiert aus der evidenten Tatsache, dass die Anzahl der Atome die Anzahl an Lebewesen übersteigt. So existiert auch nur eine Biosphäre bzw. Erde, welche alle darunter liegenden Stufen mit ihrer Wechselwirkungen in sich vereint.

Dieses Prinzip der Emergenz bei der Teilnehmerbildung ist auch auf ein CPPS anwendbar. Das kleinste Element in einem CPPS ist das Produkt, welches eine geringe Komplexität aufweist. Bei dem Produkt kann es sich bspw. um eine Schraube, ein Sensorelement, eine Stromversorgung oder eine Leitung handeln. Gemeinsam könnten diese Elemente oder Produkte eine komplexere Entität bilden, z. B. ein Feldgerät mit neuen Eigenschaften. Verbunden über einen Feldbus kann dieses Feldgerät ein Teil eines Steuergeräts sein. Dieses Steuergerät ist selbst ein Teil einer Station, welches wiederum ein Teil einer Fertigungslinie oder einer Matrixproduktion ist. Alle Fertigungslinien bilden eine Firma. Firmenübergreifend wird eine vernetzte Welt erschaffen. Gemeinsam können alle Produktionssysteme der Welt auch das *Intra Planetary Production System* (IPPS) bilden, ein weltweit vernetztes Produktionssystem. Durch eine Vernetzung von Entitäten, unabhängig von ihrer Hierarchie, können Entitäten mit neuen Funktionen [Ver13, S. 4], Eigenschaften oder gar Strukturen entstehen.

In Abb. 3.2 ist die steigende Komplexität einer Entität in einem CPPS veranschaulicht. Analog zur Biologie überragt die Anzahl an Entitäten der niedrigen Stufen die der nächsthöheren Stufe. Dies kann deduktiv hergeleitet werden bei der Betrachtung einer Firma oder Fabrik: Innerhalb einer Firma existieren diverse Fertigungslinien. Jede Fertigungslinie besteht aus einer Vielzahl an Stationen. Der Grad der Dezentralisierung reduziert sich mit jeder Stufe. Anders betrachtet, steigt die Zentralisierung auf Systemebene mit jeder Stufe.

3.1.2 Quantifizierung der Entitäten in produzierenden Unternehmen

Die Anzahl der vernetzten IoT-Geräte wird in den kommenden Jahren stetig steigen [Sta16, Sta20]. In den Literaturquellen werden Entitäten mithilfe von Taxonomien [BHH21] oder ihren Aufgaben klassifiziert [HHH+19]. Eine Quantifizierung, wie viele Entitäten jedoch maximal in einem CPPS existieren, wurde noch nicht

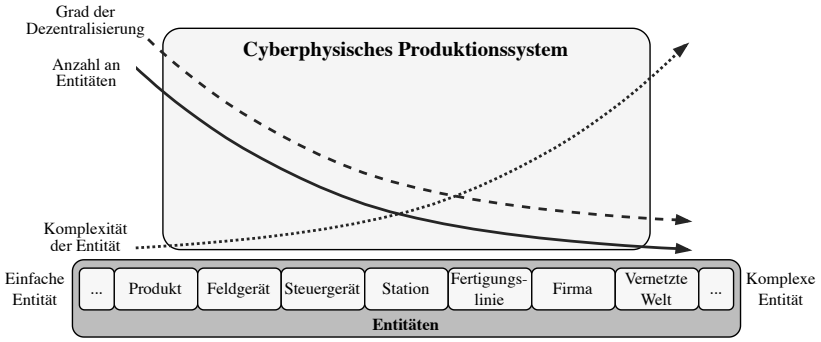
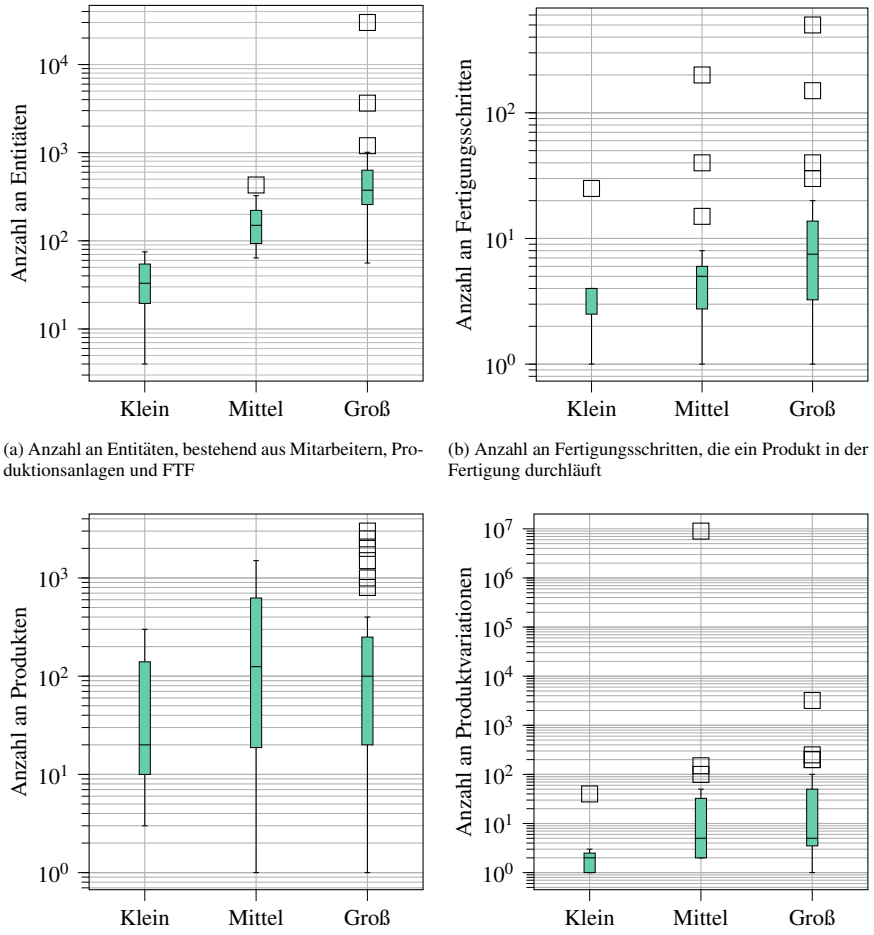


Abb. 3.2: Emergenz in einem cyberphysischen Produktionssystem, bei der durch das Zusammenspiel seiner Entitäten neue Entitäten erschaffen werden. Jede Entität kann mit jeder anderen Entität vernetzt sein und basierend darauf interagieren. Mit steigender Komplexität der Entität reduziert sich die Anzahl an Entitäten.

(ausgiebig) betrachtet. So wird geschätzt, dass weniger als 1.000 Geräte mit drahtlosen Technologien wie *5G* vernetzt sein werden [HWW+16]. Eine quantitative Erhebung bzgl. der Anzahl an existierenden Entitäten innerhalb eines CPPS wurde in Form einer Online-Befragung durchgeführt [Bus21]. Im Rahmen dieser Studie wurden in Summe 62 Datensätze von produzierenden Unternehmen innerhalb der EU erfasst. Die Datensätze sind kategorisiert nach Unternehmensgröße und Jahresumsatz. Es existieren für die Unternehmensgröße die drei Kategorien *Klein*, *Mittel* und *Groß*. Die Eingruppierung nach *Klein* und *Mittel* erfolgt gemäß der Definition von kleinen und mittleren Unternehmen (KMU) der EU-Kommission. Diese sieht vor, dass ein Unternehmen zu den KMU zählt, wenn dieses weniger als 250 Beschäftigte hat und einen Jahresumsatz von höchstens 50 Millionen Euro erwirtschaftet oder eine Bilanzsumme von maximal 43 Millionen Euro aufweist [Eur20]. Die genaue Aufschlüsselung der KMU-Definition ist in der Tabelle 3.1 dargestellt. Unternehmen, welche aufgrund der Anzahl an Mitarbeitern oder des Umsatzerlöses bzw. der Bilanzsumme in keine der beiden Kategorien passen, sind in der Kategorie *Groß* wiederzufinden.

Tabelle 3.1: Definition von kleinen und mittleren Unternehmen (KMU) gemäß der KMU-Definition der EU-Kommission, basierend auf [Eur20]

Kategorie	**Anzahl Mitarbeiter**	**Umsatzerlös in Mio. €**		**Bilanzsumme in Mio. €**
Kleinst	≤ 10	≤ 2	oder	≤ 2
Klein	≤ 50	≤ 10	oder	≤ 10
Mittel	≤ 250	≤ 50	oder	≤ 43
Groß	> 250	> 50	oder	> 43



(a) Anzahl an Entitäten, bestehend aus Mitarbeitern, Produktionsanlagen und FTF

(b) Anzahl an Fertigungsschritten, die ein Produkt in der Fertigung durchläuft

(c) Anzahl der verschiedenen gefertigten Produkte

(d) Anzahl der Variationen je Produkt im Durchschnitt

Abb. 3.3: Quantifizierung eines cyberphysischen Produktionssystem für kleine, mittlere und große Unternehmen, basierend auf [Bus21]

Nach der KMU-Definition der EU-Kommission muss bei einem Unternehmen, welches Teil einer größeren (Unternehmens-)Gruppe ist, je nach Höhe der Beteiligung die Mitarbeiterzahl und der Umsatz bzw. die Bilanzsumme der Gruppe mit berücksichtigt werden. Diese wurden in der Umfrage für die Quantifizierung der Entitäten innerhalb eines CPPSs nicht berücksichtigt, da diese lediglich für den Zugang zu Finanzmitteln und EU-Förderprogrammen, die speziell auf diese Unternehmen ausgerichtet sind, notwendig sind.

Basierend auf den Umfrageergebnissen aus [Bus21] wurden die Daten hinsichtlich der Anzahl an vorhandenen Entitäten, Fertigungsschritten, gefertigten Produkte und Variationen je Produkt untersucht (vgl. Abb. 3.3). Diese grafische Darstellung

umfasst den Median, die zwei Quartile (0,25 und 0,75), die Whisker (1,5-fache des Interquartilsabstands) und einige Ausreißer.

In Abb. 3.3a ist die Anzahl an Entitäten dargestellt, bestehend aus der Anzahl an Mitarbeitern, Produktionsanlagen und FTF (vgl. Anhang C, Tabelle B.1). Entitäten wie z. B. Sensoren werden hier nicht berücksichtigt. Verschiedene Sensoren können über einen Sensor-Agenten, der diese verwaltet, angebunden werden [RCJ09]. Dies wäre eine hierarchische, zentrale Struktur, sodass nur ein Sensor-Agent als weitere Entität hinzukommt. Im weiteren Verlauf dieser Arbeit verzichtet der Autor auf eine Quantifizierung der Sensoren. Neben den hier aufgeführten Entitäten können ein ERP, ein Warehouse Management System, ein Manufacturing Execution System oder eine bzw. mehrere Mensch-Maschine-Schnittstellen in einem CPPS existieren. Diese würden als eine Verschiebungskonstante zu der dargestellten Anzahl an Entitäten hinzukommen. Hierbei handelt es sich um statische Entitäten, welche unabhängig von der Produktion bzw. der Fertigung immer existieren.

Wie bereits in Abschnitt 3.1 eingeführt, kann ein Transport- oder ein Fertigungsauftrag ebenfalls eine Entität innerhalb eines CPPS sein [BLS+17, SGCS+21]. Hierbei handelt es sich um Entitäten, welche zur Laufzeit eines CPPS hinzukommen. Ein Transportauftrag kann zwischen zwei Fertigungsschritten oder bei der Anlieferung von Waren zu einer Produktionsanlage bzw. einer Ware ins Lager erfolgen. In Abb. 3.3b ist die Anzahl der benötigten Fertigungsschritte für ein Produkt dargestellt. Die tatsächlich ausgelösten Transportaufträge lassen sich hieraus nicht ableiten. Diese sind davon abhängig, wie viele Produkte am Tag und wie viele Produkte in einem Transportauftrag befördert werden. So kann bspw. bei einer Fertigung mit Losgröße 1 zwischen jedem Fertigungsschritt ein Transportauftrag notwendig sein. Bei einer Massenfertigung hingegen kann auch eine Vielzahl an Waren mithilfe eines Ladungsträgers zusammengefasst mit einem Transportauftrag befördert werden. Die Anzahl an Fertigungsaufträgen und die dazwischenliegenden notwendigen Transportaufträge variieren u. a. nach Bearbeitungsdauer, Anzahl an Produktionsanlagen und Aufbau der Fertigung.

Die durchschnittliche Anzahl an gefertigten Produkten je nach Unternehmensgröße ist in Abb. 3.3c dargestellt. In Abb. 3.3d sind die Variationen je Produkt in Abhängigkeit von der Unternehmensgröße dargestellt. Trotz der unterschiedlichen Anzahl an Mitarbeitern bei den mittleren und großen Unternehmen ist die Anzahl an Produkten bzw. an Variationen je Produkt vergleichbar. Kleine Unternehmen hingegen produzieren Produkte mit einer geringen Anzahl an Variationen und sind eher auf ein einzelnes Produkt fokussiert.

3.2 Kommunikation in verteilten Systemen

In der Biologie entsteht die deutlichste Form der *Kommunikation* genau dann, wenn ein Tier bei der Ausführung einer Handlung ein anderes Tier und dessen Verhalten beeinflusst. Dies kann durch den Austausch von (elektrischen, taktilen, optischen,

auditiven oder chemischen) Signalen erfolgen, welche von allen Teilnehmern gegenseitig anerkannt werden. [SMM+18, S. 1135]

In der Informationstheorie nach *Shannon* wird *Kommunikation* als Übertragung von Informationen zwischen Teilnehmern definiert. Diese besteht aus der Informationsquelle, dem Sender, dem Übertragungskanal, dem Empfänger und der Informationssenke. Die semantische Interpretation von Informationen ist hiervon ausgeschlossen [Sha48]. Die Interpretation und Reaktion durch die einzelnen Teilnehmer kann in diesem Kontext als *Interaktion* verstanden werden [Jos21].

Ein Teilnehmer (nachfolgend auch Komponente genannt) entspricht in diesem Abschnitt entweder einem physikalischen Gerät oder einem Softwareprozess [ST17, S. 2]. Wenn Sensoren, Aktoren oder andere mobile Geräte miteinander kommunizieren, so wird das auch als Machine-to-Machine (M2M) bezeichnet [WPS+11]. Teilnehmer tauschen Informationen in Form von Nachrichten (engl.: *messages*) aus. Ein Teilnehmer reagiert typischerweise auf eingehende Nachrichten. Diese werden bearbeitet und, sofern gewünscht, mit einer entsprechenden Nachricht beantwortet.

Bei der Betrachtung des OSI-Schichtenmodells [Zim80] in Abb. 3.4 erfolgt die Kommunikation über verschiedene Schichten hinweg. Auf Schicht 1 und 2 erstreckt sich der Zugang zum Netzwerk, dem Übertragungskanal bzw. dem Medium. Teilnehmer erhalten Zugang zum Netzwerk durch die Verwendung von Standards wie IEEE 802.15.4 [IEE22], IEEE 802.11 [IEE] oder anderen Kommunikationstechnologien wie 5G-Funknetze [SMS+17].

IEEE 802.15.4 ist u. a. die Grundlage für ZigBee, 6LoWPAN, WirelessHART und Thread. Dieser Standard soll bei der Verwendung einer niedrigen Datenrate, eine mehrmonatige bis mehrjährige Akkulaufzeit mit sehr geringer Applikationskomplexität unterstützen. IEEE 802.11 hingegen ist der Standard für lokale Funknetzwerke wie Wireless Local Area Network (WLAN), während *5G* die derzeit neueste Generation des Mobilfunkstandards ist. Um Teilnehmer aus unterschied-

#	ISO/OSI Schicht (Layer)
7	Anwendungsschicht (Application)
6	Darstellungsschicht (Presentation)
5	Sitzungsschicht (Session)
4	Transportschicht (Transport)
3	Vermittlungsschicht (Network)
2	Sicherungsschicht (Data-Link)
1	Bitübertragungsschicht (Physical)

Abb. 3.4: Das OSI-Schichtenmodell für die Vernetzung von offenen Systemen, basierend auf [Zim80]

lichen physikalischen Netzwerken miteinander kommunizieren zu lassen, können Gateways eingesetzt werden. Die Vermittlungsschicht ist für die Adressierung oder das Routing von Paketen zuständig. Als Transportschicht kann beispielsweise das verbindungsorientierte Transmission Control Protocol (TCP) [Int81b], das verbindungslose User Datagram Protocol (UDP) [Int81a] oder das UDP-basierte Verschlüsselungsprotokoll QUIC [Int20] verwendet werden.

Diese Arbeit setzt nachfolgend bei der betrachteten Kommunikation innerhalb eines CPPS eine IP-basierte Kommunikation voraus. Alle entwickelten Verfahren und Algorithmen beziehen sich somit auf die Transportschicht sowie die darüberliegenden Schichten. Sollte dies nicht zutreffen, wird dies explizit erwähnt. Eine IP-basierte Kommunikation wird auch innerhalb von RAMI 4.0 angenommen [Pla16b].

Eine Kommunikation zwischen den Teilnehmern ist notwendig, weil diese entweder physikalisch oder logisch voneinander getrennt sind. Eine physikalische Trennung bedeutet, dass mindestens zwei (physikalische) Teilnehmer miteinander kommunizieren wollen, diese sich aber an unterschiedlichen Orten befinden. Eine logische Trennung bedeutet, dass zwei Softwareprozesse, die auf einem Computer (oder in einem Netzwerk) ausgeführt werden, miteinander kommunizieren wollen. Die Teilnehmer sind in beiden Fällen verteilt und handeln jeweils autonom. Dem Benutzer erscheint die Sammlung von verteilten Teilnehmern von außen als ein kohärentes, einzelnes System. Dabei lässt sich ein verteiltes System auf unterschiedliche Weisen organisieren. [ST17]

3.2.1 Systemarchitekturen zur Organisation von verteilten Systemen

Ein CPPS besteht aus einer Vielzahl von Teilnehmern, die auf unterschiedlichen Weisen organisiert sein können. In der Kommunikation von verteilten Systemen wird zwischen zentralen, dezentralen und hybriden Systemarchitekturen unterschieden [ST17, S. 76 ff.]. Zentral und dezentral organisierte Systeme unterscheiden sich in der Verteilung der Informationen zwischen den Teilnehmern. Basierend auf diesen Systemarchitekturen können unterschiedliche logische Architekturen angewendet werden [ST17, S. 56 ff.]. Als Beispiele für logische Architekturen sind Schichtenarchitekturen oder Publish-Subscribe zu nennen.

Zentralisierte Organisation

Zentralisierte Systeme sind vermutlich die am weitesten verbreiteten Kommunikationsarchitekturen. Es existiert dabei eine zentrale Instanz (engl.: *server*), die mit

einer Vielzahl an Teilnehmern (engl.: *clients*) verbunden ist [BFO+98], wie in der Abb. 3.5 dargestellt: Ein Server bietet einen Dienst (engl.: *service*) an.

Möchte ein Client diesen Service nutzen, so stellt dieser eine Anfrage (engl.: *request*). Hat der Client den Server erreicht, so antwortet der Server entsprechend mit einer Antwort (engl.: *response*), unabhängig davon, ob die Nutzung des Services erfolgreich war oder nicht. Zwischen jedem Client und dem Server besteht eine Direktverbindung (engl.: *point-to-point*). Anfragen werden immer vom Client aus an den Server geschickt. Zentralisierte Systeme bringen Vorteile mit sich, die naheliegend sind: die einfache Entwicklung von Applikationen und die Verteilung dieser. Letzteres bedeutet, dass die Applikation bspw. nur auf dem zentralen Server aktualisiert werden muss. Ein Client kann sich diese von dem Server herunterladen. Aufgrund der zentralen Speicherung der Daten ist auch die Verwaltung der Daten einfach. Nachteilig wirkt sich jedoch aus, dass die Kommunikation zwischen den Teilnehmern nicht mehr möglich ist, sofern der Server ausfällt. Ist ein System von einer Komponente abhängig und diese fällt aus, so handelt es sich hierbei um einen *Single Point of Failure* (SPOF), einen einzelnen Ausfallpunkt.

Die Skalierung eines zentralen Servers ist nur vertikal möglich. Eine vertikale Skalierung (engl.: *scale up*) bedeutet, dass einem Server mehr Ressourcen (mehr CPU oder RAM) zugeteilt werden [MKL+]. Die horizontale Skalierung (engl.: *scale out*) dagegen sieht vor, dass weitere Rechner hinzugefügt werden. Dies widerspräche dem Paradigma der zentralen Instanz, da diese nicht mehr die einzige zentrale Instanz wäre. Sollen die Daten bei mehreren Servern exakte Kopien beinhalten, so können Konsistenz-Strategien gefahren werden [YV]. Existieren jedoch mehrere Server und die Daten werden an mehreren Stellen zur gleichen Zeit geändert, so müssen sich die Server auf einen gemeinsamen Wert mithilfe einer Konsensus-Strategie (engl.: *consensus*) einigen, welche Änderung gespeichert wird [XZL+20]. Ist ein Server an Informationen vom Client interessiert, so ändert sich auch ihr Verhältnis: Der Server wird zum Client und der Client wird zum Server.

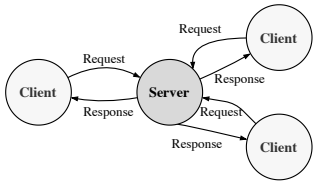


Abb. 3.5: Interaktion zwischen einem Client und einem Server. Der Client schickt eine Anfrage (engl.: *request*) an den Server, welcher diese entsprechend bearbeitet und beantwortet (engl.: *response*).

Dezentralisierte Organisation

Während bei der zentralisierten Organisation nur ein Server existiert und dieser lediglich vertikal skaliert werden kann (durch Hinzufügen weiterer Rechenleistung), so besteht bei der dezentralen Organisation die Möglichkeit der horizontalen Skalierung. Hierbei kann ein Client oder Server physisch so aufgeteilt werden, dass jede Teilkomponente einen ihm logisch zugeteilten Bereich eigenständig verwaltet [ST17, S. 80 ff.]. Zusätzlich können alle Teilkomponenten gleichberechtigt sein. Jeder Client ist ein Server und jeder Server ist ein Client.

Eine mögliche Form der dezentralen Organisation ist die Bildung eines Peer-to-Peer (P2P)-Systems durch die Teilnehmer (engl.: *peers*) (vgl. Abb. 3.6). Peers sind autonom hinsichtlich ihres Verhaltens, jedoch unterschiedlich bzgl. ihrer Fähigkeiten, Verbindungen (links zu anderen Peers) und Interessen [Gra10, S. 3]. Konträr zu

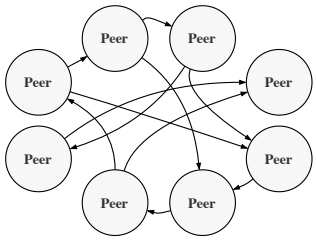


Abb. 3.6: Beispiel eines einfachen Peer-to-Peer-Netzwerks, bei dem diverse Peers untereinander verbunden sind. Jeder Peer bietet seine Ressourcen an, die von den anderen Peers genutzt werden können.

der zentralisierten Organisation, bei der nur ein Server seine Ressourcen bzw. Dienste anbietet, können alle beteiligten Peers ihre Ressourcen in dem Overlay-Netzwerk zur Verfügung stellen.

P2P-Netzwerke können in zwei Klassen aufgeteilt werden: *strukturierte* und *unstrukturierte* [ECP+05, AS04, KKH+13]. *Strukturierte* P2P-Netzwerke folgen bestimmten Regeln beim Aufbau des Netzwerks, bei der Vernetzung von Peers oder der Speicherung von Daten. Bei der Vernetzung kann eine maximale Anzahl Verbindungen zu anderen Peers als Obergrenze festgesetzt werden. Der Aufbau des Netzwerks kann dabei einem Algorithmus entsprechen, der eingehalten werden muss. *Strukturierte* Netzwerke verhalten sich idealerweise deterministisch. So kann eine Topologie – die einem Baum, einem Ring, einem Gitter oder Ähnlichem entspricht – aufgebaut werden. Die bekanntesten Vertreter der strukturierten P2P-Netzwerke sind CAN [RFH+], Chord [SMK+01], Kademilia [MM02] und Pastry [RD01]. Der Aufwand bei der Platzierung eines Peers oder der Daten kann daher höher sein als der bei einem *unstrukturierten* P2P-Netzwerk. *Unstrukturierte* P2P-Netzwerke folgen einfachen Regeln, ohne dass ein Peer die Topologie kennt oder eine bestimmte Topologie aufgebaut wird. Eine Überlastung eines einzelnen

Peers ist somit möglich. Bekannteste Vertreter für unstrukturierte P2P-Netzwerke sind Freenet [CSW+01], BitTorrent [Coh03] und Gnutella [Fra01, Rip01].

P2P-Netzwerke bringen einige Vorteile mit sich, u. a. die Skalierung des Netzwerks und die System-Fehlertoleranz. Jeder Peer hat nur eine beschränkte Sicht aufs Netzwerk durch seine Verbindungen zu den anderen Peers. Fällt ein Peer aus, so ist der Rest des gesamten Netzwerks noch immer verfügbar [ECP+05].

3.2.2 Publish-Subscribe für eine lose Kopplung

Zu dem Zeitpunkt, zu dem ein Client sich mit dem Server verbinden möchte, existiert eine zeitliche und eine räumliche Abhängigkeit [CLZ00]. Die zeitliche Abhängigkeit setzt voraus, dass zu dem Zeitpunkt der Kommunikation beide Teilnehmer erreichbar sind. Die räumliche Abhängigkeit setzt voraus, dass zu dem Zeitpunkt der Kommunikation beide Teilnehmer sich explizit kennen. Um die räumliche Abhängigkeit abzuschwächen, besteht die Möglichkeit der Verwendung des *Publish-Subscribe Patterns* [muh06]. Dieses Prinzip ist in Abb. 3.7 dargestellt. Die zu verteilenden Daten werden durch den Erzeuger (engl.: *producer*) an eine zentrale Instanz (engl.: *Broker*) gesendet. Zu dem Zeitpunkt des Sendens (engl.: *publish*) weiß der Producer jedoch nicht, wer seine Daten erhalten wird. Der Broker übernimmt die Verteilung der Daten an die jeweiligen Interessenten (engl.: *consumer*). Dafür existieren im Broker entsprechende Einträge (engl.: *subscriptions*), welche Konsumenten an welchen Daten interessiert sind (engl.: subscribe). Somit erhält ein Konsument nur die Daten, welche auch abonniert wurden. Mithilfe dieser logischen Architektur müssen die Konsumenten lediglich die Zieladresse des Brokers kennen.

Bei *Publish-Subscribe* wird u. a. zwischen *topic-based*, *content-based* und *type-based* unterschieden [EFG+03]: Das *Content-based*-Verfahren sieht vor, dass ein Subscriber spezifisch definieren kann, welche Daten für ihn relevant sind, ohne genau zu wissen, wie die Nachricht aussieht [BCM+99]. Im *Topic-based*-Verfahren ist vorgesehen, dass ein Subscriber nur an einem bestimmten Thema einer Nachricht interessiert sein kann [RKC+01], unabhängig davon, wie sich ggf. bestimmte Werte innerhalb der Nachricht verhalten. Ist die Struktur bzw. der Typ der Nachricht bei der Anmeldung relevant, so handelt es sich um das *Type-based*-Verfahren.

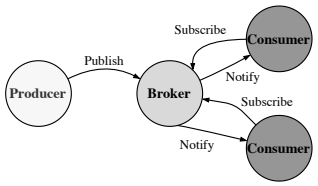


Abb. 3.7: Ein Publisher sendet seine Daten an einen Broker, welcher die Daten an die entsprechenden Consumer verteilt.

Publish-Subscribe Systeme müssen nicht zentralisiert organisiert sein. Auch dezentral organisierte Systemarchitekturen können ein Publish-Subscribe über P2P-Netzwerke realisieren. Dies wurde u. a. in [CMT+07, TBF+, AGD+06] gezeigt.

3.3 Kommunikation in heutigen cyberphysischen Produktionssystemen

Damit industrielle Anwendungen wie cyberphysische Produktionssysteme zum Erfolg werden, sind Referenzarchitekturen wünschenswert. Verschiedene Standardisierungsinitiativen, Industriekonsortien und Forschungsgruppen arbeiten an solchen Referenzarchitekturen [WE16]. Zwei bekannte Referenzarchitekturen sind das deutsche *Referenzarchitekturmodell Industrie 4.0* (RAMI 4.0) [HHH+19] und die amerikanische *Industrial Internet Reference Architecture* (IIRA) [SMD+19]. Ziel dieser Gremien, Standardisierungsinitiativen, Konsortien und Referenzarchitekturen ist es, entsprechende Anforderungen der Industrie 4.0 zu bedienen und Möglichkeiten zur Herangehensweise bei der Integration zu bieten. Allerdings wird von keiner der beiden Referenzarchitekturen die Verwendung einer zentralisierten oder dezentralisierten Systemorganisation bzgl. der Kommunikation empfohlen. Darüber hinaus empfehlen diese technologieunabhängige Referenzarchitekturen die Verwendung des Publish-Subscribe-Kommunikationsmodells. Daher werden nachfolgend in diesem Abschnitt zunächst die funktionalen und die nicht-funktionalen Anforderungen an die Kommunikation identifiziert. Im Anschluss an die erhobenen Anforderungen werden gängige Möglichkeiten vorgestellt, wie sich ein heutiges CPPS hinsichtlich der Organisation (vgl. Abschn. 3.2) aufbauen lässt.

3.3.1 Funktionale und nicht-funktionale Anforderungen

Funktionale Anforderungen legen fest, wie sich ein System verhalten soll. So können in einem (Kommunikations-)System einzelne Teilnehmer in Verbindung treten und dabei Informationen bzw. Daten miteinander austauschen. Nichtfunktionale Anforderungen hingegen gehen über die funktionalen Anforderungen hinaus und geben an, wie gut ein System eine Funktion oder eine Dienstleistung erfüllen soll. Daher werden die nichtfunktionalen Anforderungen auch als Randbedingungen oder Qualitätsmerkmale bezeichnet.

Funktionale Anforderungen

Datenzugriff und -transparenz

Die Datenerfassung und der Datenzugriff sind für die Gewinnung von Informationen relevant. So können z. B. Sensordaten der digitalisierten Maschinen genutzt werden, um Funktionen der (dezentralen) Steuerung und Analyse zu unterstützen [LFK+14]. Die vorhandenen Daten können aber auch für eine verbesserte Grundlage verwendet werden, um menschliche Entscheidungen zu treffen [HPO16], bspw. durch eine Visualisierung von Produktionsprozessen [GSL+14]. Voraussetzung ist, dass die Daten in transparenter Weise zugänglich sind und dass die Datensouveränität beim Betreiber des CPPS besteht. Daher soll dieser die vollständige Kontrolle über gespeicherte und verarbeitete Daten haben, neben der unabhängigen Entscheidung darüber, welche Entitäten auf die Daten zugreifen dürfen.

Datenverteilung und Ereignisbenachrichtigung

Alle Knoten sollten in der Lage sein, miteinander zu kommunizieren [HHH+19, SMD+19]. Am einfachsten ist die *One-to-One-Kommunikation* zwischen zwei Knoten. Weiterhin kann es vorkommen, dass ein Knoten bestimmte Informationen mit *n* Knoten teilen will bzw. von *n* erhalten soll. Es muss also eine (1:1)- oder (n:1)-Kommunikation möglich sein. Die Kombination aus beidem wird als *Many-to-Many-Kommunikation* bezeichnet. Eine Möglichkeit, diese Art der Datenverteilung und Ereignisbenachrichtigung zu ermöglichen, ist die Verwendung von *Publish-Subscribe* (vgl. Abschn. 3.2.2). Diese wird u. a. in den technologieunabhängigen Architekturen RAMI 4.0 [HHH+19] und in der IIRA [SMD+19] vorgeschlagen. Dabei dann die Datenverteilung entweder in einem Sprung bzw. mit einem einzigen Hop erfolgen oder mit mehreren in einem Multihop-Verfahren.

Service/ Resource Discovery

Alle verbunden Teilnehmer, unabhängig davon ob es sich um eine physische oder eine virtuelle Entität handelt müssen innerhalb des CPPS auffindbar sein [HRC+18]. Das automatische Auffinden von neuen Teilnehmern reduziert den Aufwand für eine manuelle Konfiguration. Weniger Konfiguration bietet eine schnellere Integration neuer Teilnehmer, aber auch eine einfachere Anpassung an unvorhergesehene Ereignisse [BFK+14]. Dies erhöht zusätzlich auch die Flexibilität, um sich an die Bedürfnisse eines Kunden hinsichtlich eines Produktes anzupassen [LFK+14]. Ein CPPS impliziert die Umsetzung einer solchen Anwendung und die Verwendung der unterschiedlichen Funktionalitäten jener Architektur [AIM10].

Nichtfunktionale Anforderungen

Skalierung

Ein System muss in der Lage sein, eine Vielzahl an Teilnehmern zu bewältigen [WE16, APP19, AIM10]. Dies bedeutet, dass es jederzeit möglich ist, dem System neue Teilnehmer hinzufügen zu können, ohne (merkliche) Leistungseinbußen. Die Skalierbarkeit kann auch auf die Verfügbarkeit der Services [APR+18] und auf die Kommunikation selbst angewendet werden. Diese betrifft neben der Anzahl der Nachrichten, welche durch das System geschickt werden, auch die eigentliche Größe der Nachricht [APR+18, HRC+18].

Um die realen (Hardware-)Kosten für eine Validierung der Skalierbarkeit gering zu halten, können solche Szenarien in Simulationen abgebildet werden [WMO+16, Erb17]. Neben der Skalierung der Anzahl an Teilnehmern kann auch der Datenverkehr auf verschiedenen physikalischen Schichten simuliert werden. Der simulierte Datenverkehr erlaubt Rückschlüsse, ob u. a. Pakete verloren gehen oder wie gut das System skalierbar ist. Idealerweise lässt sich auch die Rekonfiguration der Teilnehmer in der Simulation abbilden. [APR+18]

Quality of Service

Der *Quality of Service* (QoS) bezieht sich sowohl auf die Zuverlässigkeit als auch auf die Verfügbarkeit des Systems. Das System muss in einer bestimmten Zeit unter den zuvor definierten Bedingungen funktionieren [APR+18]. Ein einzelner Knoten sollte keinen *Single Point of Failure* (SPOF) erzeugen und zu einem Ausfall des gesamten Systems führen. Grund für den Ausfall könnte bspw. die Hardware oder ein Softwarefehler sein. Darüber hinaus ist eine zuverlässige Ende-zu-Ende-Nachrichtenzustellung [DPS+11] der gesendeten und der empfangenen Pakete ein weiterer wichtiger Aspekt [APR+18].

Im Kontext der funktionalen und nichtfunktionalen Anforderungen an die Kommunikation kann neben der Echtzeitfähigkeit auch die Sicherheit angebracht werden. Echtzeitfähigkeit wird auf der Feldebene benötigt [Mon14, Ver13], jedoch nicht bei der Übertragung von einem Teilnehmer zu einem anderen. Die Datenübertragung selbst muss zeitnah erfolgen, sodass bspw. ein Betreiber eines CPPS auf ein eingetroffenes Ereignis reagieren kann. Die Sicherheit bei der Datenübertragung umfasst neben der Verschlüsselung von Nachrichten auch die Authentizität der Teilnehmer [HPO16, AMN+19]. Dies beinhaltet auch, dass keine sensiblen Daten entwendet werden dürfen. Beides sind wichtige Aspekte, die jedoch in dieser Arbeit nicht weiter betrachtet werden.

Referenzarchitekturen wie RAMI 4.0 und IIRA benötigen Kommunikationstechnologien mit unterschiedlichen Funktionalitäten. Zu diesen Funktionalitäten zählen u. a. die zuverlässige Zustellung, die automatische Erkennung von Teilnehmern (engl.: *service discovery*) und die automatische Rekonfiguration bei Ausfällen des Netzwerks. Darüber hinaus können alle Knoten miteinander kommunizieren. Keine der beiden Referenzarchitekturen empfiehlt die Verwendung einer zentrali-

sierten oder dezentralisierten Systemorganisation bzw. schreibt zu verwendenden Technologien vor. Jedoch wird die Verwendung des Publish-Subscribe-Kommunikationsmodells vorgeschlagen.

3.3.2 Zentrale Organisation

Heutzutage sind CPPS hinsichtlich ihrer Kommunikation meist zentral organisiert. Solch ein Szenario ist in Abb. 3.8 dargestellt. Alle Teilnehmer, wie beispielsweise FTF, Sensoren oder Arbeitsstationen, sind über einen zentralen Server verbunden. Die gesamte Kommunikation wird über die zentrale Instanz, auch Broker genannt, verteilt.

Tabelle 3.2 zeigt eine Auswahl der wichtigsten und am weitesten verbreiteten Lösungen zur M2M-Kommunikation. Diese Auswahl basiert auf [DCJ+19, KD20, PTD+19, GCF+19, Nai17]. Das Message Queuing Telemetry Transport-Protokoll (MQTT) ist ein offenes, schlankes Publish-Subscribe-Protokoll für Geräte mit limitierter Bandbreite [MQT22]. MQTT benutzt TCP/IP für die Datenübertragung, wohingegen FIWARE's Orion eine HTTP Representational State Transfer (REST) API [Fie00] verwendet. TCP bietet als Transportprotokoll die garantierte Übertragung eines Pakets. So existiert eine Wechselwirkung bei der Wahl des Übertragungsprotokolls. Der Vorteil von HTTP liegt in der weiten Verbreitung und in seiner Interoperabilität; allerdings benötigt diese mehr Netzwerkbandbreite, was eine höhere Latenz bei der Übertragung zur Folge hat [Nai17].

Message Queuing Telemetry Transport (MQTT)

MQTT ist standardisiert von der *Organization for the Advancement of Structured Information Standards* (OASIS) und soll das Protokoll für das Internet der Dinge sein. MQTT deklariert sich dabei als ein extremst leichtgewichtiges Publish-Subscribe-

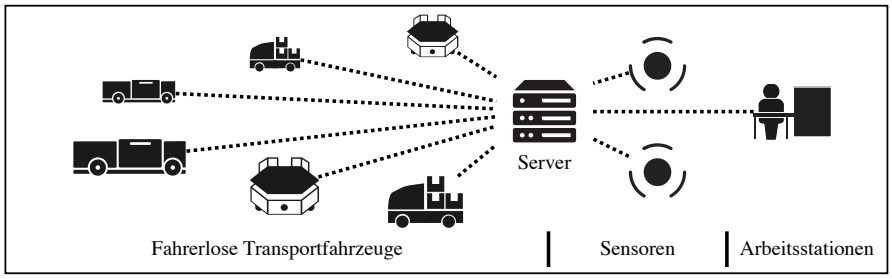


Abb. 3.8: In einem cyberphysischen Produktionssystem existieren unterschiedliche Teilnehmer wie Fahrerlose Transportfahrzeuge, Sensoren und Maschinen. Diese kommunizieren über einen zentralen Server

M2M-Protokoll. [OAS19] Als Referenzimplementierungen existieren u. a. die quelloffenen Broker *Eclispe Mosquitto*[1] und *RabbitMQ*[2].

Die Größe der Verwaltungsdate, die zu jedem Paket mit Nutzdaten mitgesendet werden, beträgt 2 Byte. Die maximale Nachrichtengröße kann 256 MB entsprechen [Nai17]. Es werden drei verschiedene Optionen unterstützt, um eine entsprecehende Dienstgüte bzw. QoS zu erreichen: *At most once*, *At least once* und *Exactly once*. Dieser Übertragungsmechanismus sagt u. a. aus, *ob* und *wie oft* eine Nachricht gesendet werden soll, bis diese vom Broker bestätigt wurde, und ob der Server die Daten zwischenspeichern soll. Es existiert auch keine Funktionalität, um Services oder Ressourcen aufzufinden. *Happ und Wolisz* zufolge hat MQTT zwei Nachteile [HW16]: Bei einem MQTT-basierten Broker weiß der Publisher nicht, wie viele Subscriber es für ein bestimmtes Topic gibt. Folglich weiß der Subscriber auch nicht, ob die Nachricht zu einem Topic alle Empfänger erreicht hat. Darüber hinaus fehlen bei MQTT auch Mechanismen zum Auffinden von Topics oder Publishern. Daher müssen alle Topics entweder vorher bekannt sein oder alternativ über einen anderen Kanal verhandelt werden.

Advanced Message Queuing Protocol (AMQP)

AMQP ist ein binäres, standardisiertes Nachrichtenprotokoll, das für die effektive Unterstützung einer breiten Palette von Nachrichtenanwendungen und Kommunikationsmustern entwickelt wurde [Kra09]. Für AMQP existieren verschiedene Referenzimplementierungen wie Apaches ActiveMQ[3] bzw. Apaches Qpid[4]. Die Verwendung eines binären Nachrichtensystems ermöglicht die Interoperabilität unter verschiedenen Anbietern. Ferner gibt es bei AMQP keine Begrenzung für die Größe einer Nachricht. Die Größe kann sowohl eine 4 GB- als auch 4 KB-sein. [LPB+15].

Tabelle 3.2: Übersicht über die zentralisierten Broker zur Machine-to-Machine-Kommunikation und einige ihrer Eigenschaften

Name	Quelle	Quality of Service (QoS)	Publish-Subscribe	Service Discovery	Logische Verbindungen	Internet-Protokoll
MQTT	[MQT22]	Diverse	✓	×	Unicast	TCP/IP
AMQP	[OAS12]	Diverse	✓	×	Unicast	TCP/IP
FIWARE Orion	[Tel21]	×	✓	×	Unicast	TCP/IP bzw. HTTP REST

[1] Eclipse Mosquitto: https://github.com/eclipse/mosquitto

[2] RabbitMQ Server: https://github.com/rabbitmq/rabbitmq-server

[3] Apaches ActiveMQ: http://activemq.apache.org

[4] Apache Qpid: https://qpid.apache.org

Die Kommunikation zwischen Client und Broker ist TCP-verbindungsorientiert. Fokus bei AMQP ist die Zuverlässigkeit, welche zwei QoS Mechanismen bieten.

FIWARE Orion Context Broker

FIWARE wurde ursprünglich im Jahr 2011 gestartet und finanziert durch eine Reihe von EU-Projekten [Eur16b, Eur17a, Eur16a]. Das Ziel von FIWARE ist es, eine standardisierte Plattform für die Entwicklung intelligenter (Stadt-)Anwendungen bereitzustellen. Der technische Kern besteht aus einer Reihe von Komponenten basierend auf einer HTTP RESTful-API [Fie00], die mit Referenzimplementierungen geliefert werden und alle quelloffen sind (vgl. Abb. 3.9). FIWARE wird u. a. von der FIWARE Foundation [FIW] beworben.

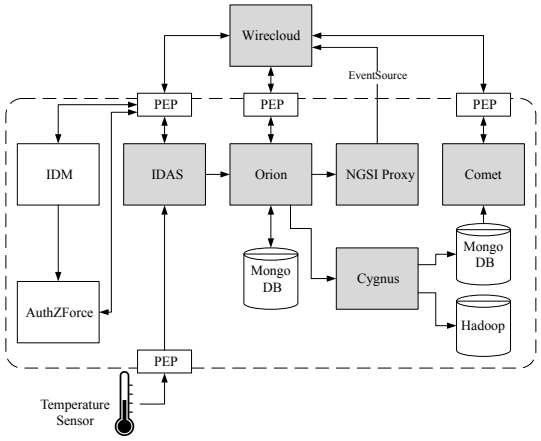


Abb. 3.9: FIWARE's Kernkomponenten (grau) und Sicherheitskomponenten (weiß), basierend auf [DS20]

Die wichtigste Komponente ist der Context Broker *Orion* [Tel21]. *Orion* ist die Open-Source-Referenzimplementierung des Context Brokers. Technisch betrachtet, handelt es sich hier um eine Mongo-NoSQL-Datenbank erweitert um eine HTTP-RESTful API. Daten, hier Entitäten, werden basierend auf dem *Next-Generation-Service-Interface*-(NGSI)-Protokoll NGSI-V2 [Ope12] und NGSI-LD [Eur19] ausgetauscht. Publisher einer Entität werden als *Context Producer* bezeichnet, Abonnenten einer Entität als *Context Consumers*.

Orion selbst besitzt keine QoS-Mechanismen, diese müssen durch die Applikation integriert werden [DS20]. Analog zu MQTT existiert bei *Orion* auch keine Möglichkeiten zur Service bzw. Ressource Discovery. In [DPF+19] wurde eine automatische An- und Abmeldung von Fahrerlosen Transportfahrzeugen für ein Materialflusssystem vorgestellt. Darüber schlagen die Autoren [DPF+19] vor, die Verwendung von Nutzdaten als Referenz für einen *Heartbeat* zu verwenden, um den

Ausfall eines Teilnehmers zu erkennen. Somit würden dedizierte Heartbeat-Pakete entfallen. Der Fehlerdetektor selbst wäre eine zusätzliche Komponente, welche parallel zu *Orion* ausgeführt wird.

3.3.3 Dezentrale Organisation

Ein dezentral organisiertes CPPS ist in Abb. 3.10 dargestellt. Konträr zu dem (zentralisierten) System aus Abb. 3.8 existiert keine zentrale Instanz, welche für die Kommunikation in dem P2P-basierten CPPS zuständig ist. Solange ein Knoten ein Teil des Netzwerks ist, wird dieser Knoten *Peer* genannt. Knoten können zu jedem Zeitpunkt dem P2P-Netzwerk beitreten (engl.: *join*) und dieses verlassen (engl.: *leave*). Nur eine Teilmenge ist an diesen Operationen beteiligt: Kein Peer hat die gesamte Übersicht über das gesamte P2P-Netzwerk. Darüber hinaus verwaltet jeder Peer nur einige wenige Verbindungen (engl.: *links*) zu anderen Knoten.

In Tabelle 3.3 ist eine Auswahl der am weitesteten verbreiteten brokerlosen Lösungen zur M2M-Kommunikation dargestellt. Hierzu zählen das populäre M2M-Kommunikationsprotokoll für industrielle Automation Open Platform Communications Unified Architecture (OPC UA) [MLD09], und der Data Distribution Service (DDS) [Obj15, Par03a].

OPC UA

OPC UA ist ein populäres M2M-Kommunikationsprotokoll für industrielle Automation ohne harte Echtzeit, welches erstmals 2006 veröffentlicht wurde [MLD09]. Die Internationale Elektrotechnische Kommission (engl.: *International Electrotechnical Commission (IEC)*) hat OPC UA auch als Normungsreihe 62541 veröffentlicht. Neben dem standardmäßigen Client-Server-Modell, welches OPC UA anbietet, beinhaltet es auch ein semantisches Informationsmodell. OPC UA wird

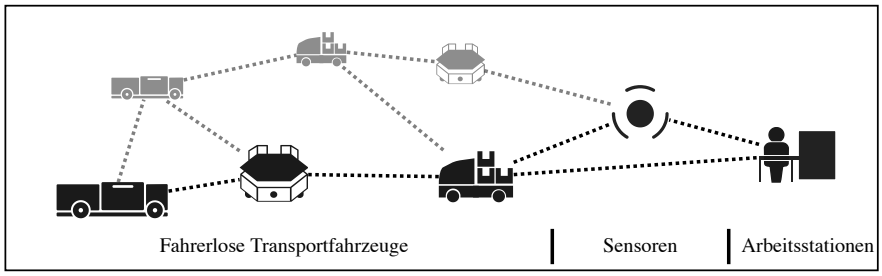


Abb. 3.10: In einem Peer-to-Peer-basierten cyberphysischen Produktionssystem kommunizieren die Peers über ein Overlay Netzwerk

primär durch die europäische Industrie vorangetrieben. Nach [UGG16] existieren in OPC UA auch keine QoS-Mechanismen.

Standardmäßig unterstützt OPC UA lediglich eine 1:1-Kommunikation. Um eine *m*:*n*-Kommunikation zu realisieren, kann OPC UA beispielsweise mit einem Broker wie MQTT kombiniert werden [PGP16]. Dies führt jedoch zu einem zentralisierten System (vgl. Abschn. 3.2.1). OPC UA hat einen kleinen Speicherbedarf und kann daher auch auf kleinen, ressourcenbeschränkten Geräten (engl.: *embedded system*) verwendet werden. Allerdings ist die vertikale Skalierung auf einem eingebetteten System schwierig, welche aus Ressourcenbeschränkung durch die Hardware resultiert.

Data Distribution Service

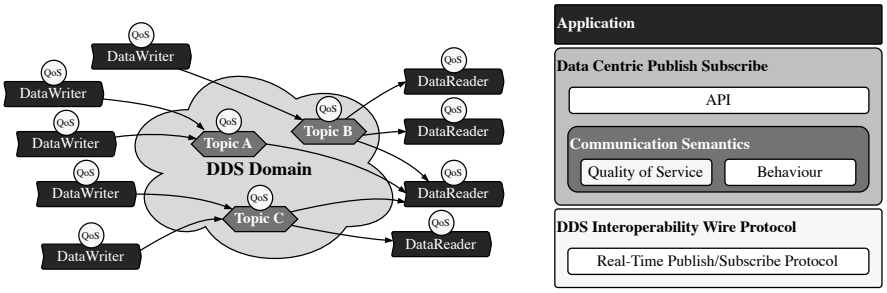
Der Data Distribution Service (DDS) ist eine datenzentrische (engl.: *data-centric*) Middleware, welche auf dem Publish-Subscribe-Modell basiert [Par03a]. Daten werden bei DDS innerhalb einer Domäne ausgetauscht (vgl. Abb. 3.11a). Eine DDS-Domäne ist ein logisches Netzwerk. Lediglich Anwendungen in der gleichen Domäne können miteinander kommunizieren. Diese werden als *DomainParticipant* bezeichnet. Das publishen von Daten zu einem Topic übernimmt bei DDS der *DataWriter*, wohingegen der *DataReader* für den Empfang von Daten und das Abonnement von Topics zuständig ist. DDS ist für drahtgebundene Systeme entwickelt worden, bei denen die Teilnehmer sich nicht die gleiche Kollisionsdomäne teilen. Ferner ist jeder *DataWriter* in einer *DDS Domain* mit jedem *DataReader* verbunden. Durch diese Verbindungen wird ein vollständiger Graph aufgebaut.

Wie in Abb. 3.11b dargestellt, besteht die DDS-Spezifikation aus zwei Komponenten, welche eine API und das erwartete Verhalten von DDS-Infrastrukturen beim Verteilen der Daten beschreiben:

- **Data Centric Publish-Subscribe** (DCPS) [Obj15]: definiert eine High-Level API für Programmiersprachen und Betriebssysteme sowie das Verteilen von Daten unabhängig von der Architektur. Zusätzlich können hier QoS-Parameter oder Kommunikationsmuster festgelegt werden.

Tabelle 3.3: Überblick über dezentral organisierte Frameworks zur Machine-to-Machine-Kommunikation und ihre Eigenschaften

Name	Quelle	Quality of Service (QoS)	Publish-Subscribe	Service Discovery	Logische Verbindungen	Internet-Protokoll
OPC UA	[MLD09]	✗	via MQTT	Ja	Unicast	TCP
DDS	[Par03a]	Diverse	✓	Ja	Unicast Multicast	TCP, UDP
0MQ	[Hin13]	At-most-once	✓	Ja	Unicast Multicast	TCP, UDP, Proprietär



(a) Teilnehmer im DDS können nur innerhalb einer Domäne kommunizieren. DataWriter senden ihre Daten zu einem bestimmten *Topic*, während die *DataReader* für den Empfang der Daten zuständig sind.

(b) Vereinfachte Darstellung der Architektur

Abb. 3.11: Data Distribution Service (DDS): Aufbau und Architektur, basierend auf [Par03a]

- **DDS Interoperability Protocol** (DDS-RTPS) [Obj21]: definiert ein *wire protocol* für eine interoperable Implementierung der DCPS-Schicht. Ferner beschreibt dieses Protokoll, wie Daten zwischen zwei Endpunkten ausgetauscht werden. Hier wird zum Beispiel auch die Zuverlässigkeit des Transports sichergestellt.

Es besteht auch die Möglichkeit, DDS-RTPS als eigenständiges Protokoll zu verwenden, ohne eine vollständige Verwendung der standardisierten DSPS-API. Die RTPS-Spezifikation definiert eine zustandsabhängige und eine zustandslose Implementierung [Obj21, S. 69]. Die zustandsabhängige Implementierung setzt voraus, dass jeder *DataWriter* den Zustand seiner *DataReader* kennt. Dies erhöht den Bedarf an Speicher beim *DataWriter* und reduziert die Skalierbarkeit. Hierdurch kann auch der zuvor gesetzte QoS eingehalten werden. Um eine große Skalierbarkeit zu erhalten, besteht die Möglichkeit zur Verwendung der zustandslosen Implementierung. Hier kennt ein *DataWriter* nicht den Zustand des *DataReaders*. Folglich wird auch weniger Speicher benötigt, aber im Zuge der Kommunikation kann ein erhöhter Bedarf an Bandbreite entstehen.

DDS sieht vor, dass ein *DataWriter* alle seine *DataReader* kennt und die Daten an die Subscriber sequenziell sendet, sofern Unicasts verwendet werden. Dieses Verfahren ist vergleichbar mit einem zentralen Ansatz, bei dem der Broker die Daten der Reihe nach an alle *Subscriber* schickt. Der DDS-Standard empfiehlt daher bei der zustandslosen Implementierung die Verwendung von UDP-Multicasts. UDP unterstützt nativ Multicast, was eine effiziente Verteilung einer einzelnen Nachricht an eine große Anzahl von Empfängern ermöglicht. Allerdings setzt DDS für die Paketgröße eine Größe von Ethernet-Paketen und eine drahtgebundene Infrastruktur voraus. Daher ist der Einsatz von DDS in drahtlosen (Sensor-)Netzwerken herausfordernd [BT12]. Eine weitere Herausforderung bzw. Limitierung können UDP-Multicasts in drahtlosen Netzwerken sein, da diese einen Einfluss auf den QoS haben [GGB+18]. Der Einfluss der physikalischen Schicht auf die Kommunikation wird in Abschnitt 3.4.2 thematisiert.

Anwendung findet DDS u. a. in der Robotik, insbesondere bei dem Programmierframework Robot Operating System 2 (ROS2). ROS2 ist eine Sammlung von Softwarebibliotheken und Werkzeugen zum Erstellen von Roboteranwendungen. Es werden Treiber, Algorithmen und Entwickler-Tools bereitgestellt, alles auf Open-Source-Basis. Für DDS existieren verschiedene Referenzimplementierungen, die theoretisch untereinander kompatibel sind. Zu nennen sind hier Fast-DDS[5], Cyclone DDS[6] und OpenDDS[7]. Der Vorgänger von ROS2, Robot Operating System (ROS), verwendet einen zentralen Broker für die Interaktionen zwischen den Knoten.

Ein reines P2P-basiertes Framework zur Realisierung eines CPPS ist das ActyxOS[8]. Wird ein Teilnehmer zu einem P2P-basierten CPPS hinzugefügt, so benötigt dies laut [TPS+19] erhöhten Programmierungs- und Integrationsaufwand. Dies kann allerdings durch die Verwendung von externen Bibliotheken oder entsprechenden Middlewares umgangen werden [AIM10]. ActyxOS basiert auf der Open-Source-Lösung libp2p[9]. Diese wiederum basiert auf Kademlia [MM02], einer Distributed Hash Table (DHT).

3.4 Diskussion der Systemorganisation eines CPPS hinsichtlich der Kommunikation

Ausgehend von den unterschiedlichen Anforderungen durch den Betreiber eines CPPS sind unterschiedliche Faktoren bei der Wahl der Kommunikation von Bedeutung. Sowohl zentralisierte als auch dezentrale Organisationsarchitekturen wurden vorgestellt (vgl. Abschn. 3.2.1). Allerdings diskutieren nur sehr wenige Publikationen die Verwendung einer dezentralen Kommunikationsstruktur im Kontext von CPPS [TPS+19]. Im Folgenden wird eine Diskussion beider Ansätze im Hinblick auf ein CPPS anhand der zuvor vorgestellten Anforderungen (vgl. Abschn. 3.3.1) geführt. Anschließend folgt eine Diskussion des Einflusses der Kommunikationstechnologie auf die Systemorganisation.

Es ist wichtig anzumerken, dass es sich bei der Kommunikation um Daten in Bewegung (engl.: *data-in-motion*) und nicht um die Speicherung von Daten (engl.: *data-at-rest*) handelt [SN15]. Für die Speicherung von Daten können z. B. dedizierte Datenbanken oder Datenlager verwendet werden.

[5] Eprosima Fast-DDS: https://github.com/eProsima/Fast-DDS

[6] Eclipse Cyclone DDS: https://github.com/eclipse-cyclonedds/cyclonedds

[7] Object Management Group OpenDDS: https://github.com/objectcomputing/OpenDDS

[8] Actyx: https://www.actyx.com

[9] libp2p: https://libp2p.io

3.4.1 Funktionale und nicht-funktionale Anforderungen

Funktionale Anforderungen

Datenzugriff und -transparenz

Ein Vorteil der Verwendung eines zentralisierten Servers ist der einfache Zugriff auf die Daten. Es gibt nur eine einzige Quelle der Wahrheit (engl.: *single source of truth*) für Daten. Folglich kann jeder, der Zugriff auf den Server hat, die Daten für seinen Zweck nutzen. Aus Sicht des Betreibers eines CPPS ist dies komfortabel und einfach. Eine Datenaggregation über mehrere (Daten-)Quellen ist nicht notwendig. Allerdings wirft die zentrale Datenspeicherung potenzielle Sicherheits- und Datenschutzprobleme auf.

In einem dezentralen System können die Daten nur dort gespeichert werden, wo diese auch benötigt werden. Konkret bedeutet dies, dass jeder Peer seine Daten lokal ohne Referenzen auf die Daten von anderen Peers speichert [RM06]. Existieren Referenzen zu Daten von anderen Peers, so ist dies ein verteilter Index (engl.: *distributed Index*) [RM06]. Als Beispiel lassen sich hier die semantisch freien Distributed Hash Tables (DHTs) nennen. Aber auch diese haben Herausforderungen zu bewältigen: Ändern sich die Indizes für die Daten, so müssen die Indizes umsortiert werden. Dies kann das gesamte Netzwerk betreffen [JOV05].

Die dezentrale Verteilung von Daten ist mit Blick auf die Privatsphäre von Vorteil, erschwert aber den Zugriff auf die Daten und deren Transparenz. Eine Aggregation der Daten ist notwendig, die Kommunikation und folglich Zeit benötigt. Da zeitkritische Operationen jedoch nur auf der Feldebene relevant sind [Ver13], lässt sich folglich bei der Aggregation von Daten eine (minimale) Latenz rechtfertigen und akzeptieren. Zur Vereinfachung der Entwicklung bzw. bei der Abstraktion der Kommunikation können Softwarebibliotheken oder Frameworks verwendet werden. So kann auch die entsprechende Bibliothek die Aggregation der Daten übernehmen [AIM10].

In einem verteilten System kann die Konsistenz von Daten über die Verifikation von Transaktionen wie in der Blockchain gelöst werden [Bas17, S. 30]. Blockchains sind allerdings nicht besonders geeignet zur Speicherung von großen Datenmengen [Bas17, S. 40]. Hierfür eignen sich die zuvor genannten DHTs besser.

Datenverteilung und Ergebnisbenachrichtigung

Sowohl im RAMI 4.0 [HHH+19] als auch in der IIRA [SMD+19] wird die Verwendung des Publish-Subscribe-Paradigmas innerhalb eines CPPS vorgeschlagen. Ziel des Publish-Subscribes ist die schnelle und zuverlässige Verteilung von Nachrichten an m Teilnehmer, welche das Topic abonniert haben. In [OKF10] vergleichen die Autoren das Publish-Subscribe-Paradigma mit Client-Servern und dem zyklischen Abfragen von Daten (engl.: *polling*). Basierend auf dem entworfenen Kostenmodell sind hier (zentralisierte) Publish-Subscribe-Systeme leistungsfähiger.

In zentralorganisierten CPPS ist der Broker (die zentrale Instanz) für die Verteilung von Events bzw. der Daten zuständig. Diese zentrale Instanz kann mit steigender Anzahl an Teilnehmern – und somit auch Verbindungen – der limitierende Faktor werden. Neben der Verteilung von Events ist der Broker auch für die Subscriptions zuständig. Die Teilnehmer selbst sind entweder Publisher und/oder Subscriber.

In einem dezentralen CPPS kann ein Teilnehmer noch eine weitere Funktion übernehmen, und zwar die des Brokers und damit der Verteilung der Events. In einem P2P-Netzwerk sind alle Teilnehmer implizit miteinander verbunden, sodass jeder Peer mit jedem anderen Peer kommunizieren kann. Sollte hier ein einzelner Teilnehmer ausfallen, fällt nicht das gesamte System aus. Eine Verteilung des Events kann innerhalb eines Netzwerks mithilfe von Gossip-Protokollen realisiert werden [DFS+13, LPR07]. Diese stellen sicher, dass jeder Peer die Nachricht idealerweise genau einmal erhält. Damit jedoch nur die Subscriber die Events erhalten, welche auch Interesse an diesen haben, kann z. B. ein zusätzliches Overlay erstellt werden. In diesem sogenannten Topic-Overlay befinden sich Publisher und Subscriber zu dem jeweiligen Topic. Dies reduziert zum einen den totalen Datenverkehr bei der Verteilung der Events. Nachteilig kann sich allerdings zum anderen die Verwaltung des zusätzlichen Overlays auswirken. Diese Topic-Overlays müssen zunächst gefunden werden, bspw. durch ein Rendezvous-Protokoll [XFD+07, S. 347] oder durch die Speicherung der Information in einer verteilten Datenstruktur wie einer DHT. Zusätzlich müssen weitere Verbindungen zu anderen Peers verwaltet werden für das entsprechende eigene Overlay.

In zentralisierten Ansätzen werden die Events i. d. R. mit einem Sprung (engl.: *single-hop*) bzw. vom Sender an den Empfänger an die entsprechenden Interessenten direkt verteilt. Dies resultiert aus der globalen Sicht des Brokers. Bei dezentralen Ansätzen hingegen, bei dem kein Knoten die globale Sicht hat, können u. U. mehrere Sprünge (engl.: *multi-hop*) notwendig sein, bis alle Knoten das Event erhalten haben. Aufgrund der Komplexität hinsichtlich der Verwaltung der einzelnen Overlays ist eine zentralisierte Lösung im Vorteil.

Service und Resource Discovery

In einem CPPS werden Aufgaben von Teilnehmern abgearbeitet. Die Zuweisung einer Aufgabe kann, wie bereits dargestellt, entweder zentral oder dezentral erfolgen. In einem dezentralen Ansatz können sich die Teilnehmer um einen Auftrag bewerben. Hierfür bietet sich das *Contract Net Protocol* an [Smi80]. Bei einer Zuweisung durch eine zentrale Instanz (z. B. ein Optimierungs- oder Orchestrator-Dienst) hat dieser die Übersicht über alle verfügbaren Teilnehmer, welche für eine Zuteilung relevant sind. Hierfür müssen die möglichen Teilnehmer aufgefunden und deren Zustand bekannt sein. Bei der zentralisierten Systemorganisation bzw. bei der Verwendung eines Brokers (z. B. MQTT oder Orion) existieren standardmäßig keine Verfahren zur Auffindung von Teilnehmern. Diese müssen eigenständig implementiert werden [DPF+19, HW16]. Bei einer großen Anzahl von Teilnehmern kann hierbei auch der zentrale Ansatz ein limitierender Faktor sein.

In einem dezentralen Ansatz muss ein Knoten erstmal ein Teilnehmer des P2P-Netzwerks werden, bevor dieser durch die anderen Teilnehmer aufgefunden werden kann. Hier muss sichergestellt werden, dass lediglich autorisierte Knoten ein Teil des P2P-Netzwerks werden können. Im Vergleich zu zentralisierten Lösungen kann dies komplexer werden [SA12]. Aber auch dezentrale Ansätze wie OPC UA verfolgen das Auffinden von Teilnehmern mithilfe einer zentralen Instanz. Hierfür wird u. a. ein Local Discovery Server (LDS) verwendet [MLD09, S. 276]. Sind diverse OPC UA Server über das Netzwerk verteilt, so kann eine hierarchische Suche initiiert werden. Hierfür existiert der Global Discovery Server (GDS) [MLD09, S. 276f.], welcher über Netzwerkgrenzen die Informationen der jeweiligen LDS vereint.

P2P-Netzwerke, wie beispielsweise Chord [SMK+01], CAN [RFH+] oder BATON [JOV05], bieten die Möglichkeit, Daten in einer DHT zu hinterlegen. Diese DHT kann verwendet werden, um ihre Fähigkeiten abzuspeichern. Der dezentrale Ansatz kann besser skalieren als ein zentralisierter Ansatz.

Neben den hier aufgeführten Punkten kann noch die Infrastruktur selbst betrachtet werden. Denn hier entstehen Kosten für die Anschaffung, die Betreibung der Infrastruktur zur Vernetzung der Server und die Energie. [MKI+16]

Nichtfunktionale Anforderungen

Skalierung

Zentralisierte Ansätze, wie z. B. eine Datenbank, liefern bei der Speicherung von Datei eine bessere Performanz als dezentralisierte Ansätze [CCK+18]. Jedoch ist die vertikale Skalierung (engl.: *scale up*) bei einem zentralisierten Ansatz limitiert (vgl. Abschn. 3.2.1). Hier können nur weitere Prozessoren oder zusätzlicher Speicher hinzugefügt werden. Bei der horizontalen Skalierung werden weitere Maschinen hinzufügt, was wiederum neue Herausforderungen impliziert [ST17]. Dazu gehört neben der Replikation auch die Verteilung der Daten, welche auch eine entsprechende Zeit benötigt. Als Beispiel lässt sich hier die Blockchain-Technologie nennen, welche eine Konsensus-Strategie benötigt, um sich auf einen gemeinsamen Wert zu einigen [CCK+18]. Wichtig ist hier zu erwähnen, dass nicht alle Daten in einer Blockchain gespeichert werden müssen. So können zum Beispiel Daten für die Abrechnung für die Blockchain relevant sein, jedoch nicht die Anfragen zur Durchführung eines Auftrags innerhalb eines CPPS [KAL+21]. Eine Blockchain wird nur benötigt, wenn Daten entweder verifiziert oder gespeichert werden, jedoch nicht, wenn sich die Daten in Bewegung befinden [WG18].

Sind die Teilnehmer innerhalb des CPPS mithilfe eines P2P-Netzwerks verbunden, so ist auch eine vertikale Skalierung des Netzwerks möglich. Diese kann sowohl nach oben als auch nach unten erfolgen. Der zusätzliche Aufwand für die Synchronisierung oder Koordinierung entfällt, der Aufwand zur Aggregation (und die Komplexität) von Daten steigt. In einem P2P-Netzwerk kann ein Knoten einen beliebigen Teilnehmer kontaktieren, um Teil des Netzwerks zu werden. Durch ent-

sprechende Strategien kann sichergestellt werden, dass zwei aufeinander folgende Knoten nicht denselben Knoten für einen Beitritt des Netzwerks kontaktieren. Bei einem zentralisierten Ansatz ist der Kontaktpunkt der zentrale Server, welcher der limitierende Faktor sein kann. Tabelle 3.4 stellt das Datenaufkommen für einen zentralisierten Ansatz dar, angelehnt an [Gra10, Abschnitt 4.3.4]. Hier existiert ein zentraler Server, der die Daten für eine entsprechende Anzahl an Clients (N) innerhalb eines Sendeintervalls (SI) und einer Paketgröße verarbeiten muss. In diesem Beispiel wurde eine steigende Anzahl und Heterogenität der Hardware vernachlässigt, hinsichtlich der Erstellung und der Verarbeitung der Daten.

Die Kommunikationsspitzen, die ein zentralisierter Server zu verarbeiten hat, können sich auch auf die Transfergeschwindigkeit der Daten auswirken. In einem P2P-basierten Netzwerk kann allerdings eine Lastenverteilung erschaffen werden. Dies kann durch die Nachahmung von physikalischen oder biologisch inspirierten

Tabelle 3.4: Exemplarisches Aufkommen an gesendeten und empfangenen Paketen bzw. Datenverkehr bei einer zentralisierten Kommunikation, in Abhängigkeit von der Anzahl der Teilnehmer (N), dem Sendeintervall (SI) und der Paketgröße

Anmerkung	**Zentraler Server**	**Clients**
Pakete gesendet je Teilnehmer	0	1
Pakete empfangen je Teilnehmer	N	0
Bestätigungen gesendet	N	0
Bestätigungen empfangen	0	1
Anzahl an gesendeten und empfangenen Paketen	2*N	2
Datenverkehr je Teilnehmer	2 * N * Paketgröße	2 * Paketgröße
$N = 10^2$; SI = 60 s; Paketgröße = 3 kB		
Anzahl Pakete je Teilnehmer	200	2
Datenverkehr je Teilnehmer je SI = 60 s	600 kB/min	6 kB/min
Datenverkehr je Sekunde	10 kB/s	0,1 kb/s
$N = 10^4$; SI = 60 s; Paketgröße = 3 kB		
Anzahl Pakete je Teilnehmer	20.000	2
Datenverkehr je Teilnehmer je SI = 60 s	6.000 kB/min	6 kB/min
Datenverkehr je Sekunde	100 kB/s	0,1 kB/min
$N = 10^6$; SI = 60 s; Paketgröße = 3 kB		
Anzahl Pakete je Teilnehmer	2.000.000	2
Datenverkehr je Teilnehmer je SI = 60 s	6.000.000 kB/min	6 kB/min
Datenverkehr je Sekunde	100.000 kB/s	0,1 kB/min

Methoden erreicht werden [LPR07]. Letzteres sieht eine epidemiologische Ausbreitung vor. Aber auch die Ausnutzung einer bestimmten Topologie, wie die eines Ringes [SMK+01] oder eines Baumes [JOV05], kann hierfür verwendet werden. Selbst in dezentralen Ansätzen wie DDS, OPC UA oder 0MQ existieren keine Topologien bzw. Strukturen.

Bei einer zentralen Lösung existiert immer ein Server. Dies gilt auch bei einer kleinen Anzahl an Teilnehmern im Netzwerk. Bei einer großen Anzahl an Teilnehmern werden sogar weitere Broker und Loadbalancer benötigt [KGR20, S. 352 ff.]. Bei einem dezentralen Ansatz, bei dem keine zentrale Instanz existiert, entfallen die Kosten für Hardware und Wartung. Ein dezentraler Ansatz funktioniert für eine kleine Anzahl an Teilnehmern genauso wie für eine große Anzahl und skaliert daher in beide Richtungen.

Quality of Service

In einem zentralisierten System sind Strategien in Bezug auf Redundanzen[10] unverzichtbar, um eine entsprechende Dienstgüte bzw. QoS sicherzustellen. Denn fällt der zentrale Server aus, ist somit auch die Kommunikation nicht mehr möglich und das gesamte CPPS erliegt dem Stillstand. Daher ist der zentralisierte Ansatz als kritisch anzusehen [DCJ+19]. Eine horizontale Skalierung zur Sicherstellung des QoS auf Systemebene ist somit unausweichlich.

Wird ein strukturiertes P2P-Netzwerk wie Chord [SMK+01] oder nBATON* [DGB21] verwendet, so kann sichergestellt werden, dass eine bestimmte Anzahl an Verbindungen zwischen den Peers existiert. Selbst wenn ein einzelner Teilnehmer ausfällt bzw. eine Anzahl an Teilnehmern ausfallen, so besteht noch immer die Möglichkeit mit anderen Peers zu kommunizieren und somit eine Güte aufrecht zu erhalten. Allerdings müssen diese zusätzlichen Verbindungen gespeichert und ggf. aufrechterhalten werden. Der SPOF ist beseitigt und die Skalierbarkeit und Verfügbarkeit des Gesamtsystems erhöht [MKL+]. Auch die Bereitstellung von Ressourcen eines Peers in einem P2P-Netzwerk kann zeitlich schwanken, was zum Vorteil genutzt werden kann. Viel beschäftigte Peers können sich auf ihre Operation konzentrieren, während Peers im Leerlauf (z. B. bei mobilen Systemen während des Ladevorgangs der Batterie) sich der Kommunikation widmen.

In P2P-basierten Netzwerken können Teilnehmer zu jedem Zeitpunkt dem Netzwerk beitreten oder dieses verlassen. Mit dem Verlassen eines Peers können auch seine Daten (oder die von anderen Teilnehmern) nicht mehr verfügbar sein. Um eine Konsistenz der Daten sicherzustellen, kann ein dedizierter Teilnehmer für eine historische Speicherung von Daten zuständig sein. Alternativ müssen die verbliebenen Peers die bisherigen Daten hinterlegen.

[10] Redundanzen können sich dabei auf weitere Instanzen zur Kommunikation beziehen, aber auch auf die Anzahl an redundanten Nachrichten, welche benötigt werden um bspw. möglichen Paketverlusten entgegen zu wirken

3.4.2 Einfluss der physikalischen Schicht auf die Kommunikationstechnologie

Dieser Abschnitt behandelt die Umsetzung des Adressierungsschemas eines IP-Pakets der Vermittlungsschicht (Schicht 3) im ISO/OSI-Modell auf die ersten beiden Schichten, abhängig vom physikalischen Medium. Nachfolgend wird sich beim physikalischen Medium auf drahtgebundene Netze und auf drahtlose Funknetztechnologien wie WLAN oder 5G beschränkt.

Das User Datagram Protocol (UDP) [Int81a] ist ein verbindungsloses, nichtzuverlässiges Protokoll auf der Transportschicht (Schicht 4) der ISO/OSI-Schichtenarchitektur (vgl. Abschn. 3.2, Abb. 3.4). Verbindungslos bedeutet, dass die versendeten Daten auf der Protokollebene der Transportschicht nicht bestätigt werden. Sowohl TCP als auch UDP erlauben Unicasts, bei dem eine Nachricht an einen einzelnen Empfänger versendet wird. Anders als TCP ermöglicht UDP auch Multicasts. Bei einem UDP-Multicast (nachfolgend: Multicast) hingegen ist die Übertragung einer Nachricht an eine Gruppe von Teilnehmern möglich. Um ein Multicast in einem lokalen Netzwerk empfangen zu können, muss eine Registrierung zunächst auf die entsprechende Adresse innerhalb der Multicast-Domäne (224.0.0.0-239.255.255.255) erfolgen. Die verbindungslosen UDP-Pakte werden zwar nicht auf der Protokollebene bestätigt, dafür aber auf den physikalischen Schichten 1 (Bitübertragungsschicht) und 2 (Sicherungsschicht). Das Versenden eines Multicasts wird wie folgt auf der physikalischen Schicht umgesetzt: In einem lokalen Netzwerk sind die Teilnehmer über einen drahtgebundenen Switch verbunden. Jeder Teilnehmer ist über seine eigene, dedizierte Netzwerkleitung mit dem Switch verbunden. Diese dedizierte Vernetzung bringt den Vorteil, dass jede Leitung einer eigenen Kollisionsdomäne und somit einem eigenen Medium entspricht. Dies ist auch der Grund, weshalb Multicasts in drahtgebundenen Netzwerken gut funktionieren. Der Multicast der Transportschicht wird in den unteren physikalischen Schichten 2 bzw. 1 in einen Unicast umgewandelt. Beim Zugriff aufs Medium werden die Telegramme bzw. die Signale bestätigt und ggf. wiederholt gesendet, bis das Bit bzw. das Frame bestätigt ist.

In drahtlosen Netzwerken hingegen, z. B. bei der Verwendung des IEEE 802.11-Standards, teilen sich die Teilnehmer das Medium Luft und somit die Kollisionsdomäne. Eine Garantie für eine erfolgreiche Übertragung eines Pakets in diesem Medium existiert nicht [Dor15]. Pakete können u. a. aufgrund von Kollisionen, Interferenzen oder einer schlechten Datenrate verloren gehen [HWW+16, ZKC+15]. Weitere Herausforderungen sind u. a. das *Hidden-Node-Problem* [RG06] oder das *Exposed-Node-Problem* [SCI03]. Um ein UDP-Multicast-Paket erfolgreich abzusenden, wird bewusst eine niedrige Datenrate gewählt [Dor15, MP17], auch, wenn eine niedrigere Datenrate zur Folge hat, dass größere Frames und somit längere Übertragungszeiten benötigt werden. Neben der Erhöhung des Energieverbrauchs beim Sender bzw. Empfänger [MP17] wird auch die Wahrscheinlichkeit erhöht, dass der am weitesten entfernte Empfänger das Paket empfangen kann. Allerdings impliziert eine längere Belegung des Mediums auch, dass die anderen

Teilnehmer weniger Zeit zum Senden erhalten. Ferner sind Erweiterungen auf der Medienzugriffssteuerung (engl.: *media access control (MAC)*) hinsichtlich Broad- und Multicasts begrenzt implementiert oder gar nicht verfügbar [PMS+, Dor15]. Zwar wird der 802.11-Standard incl. der Frequenzbändern durch die IEEE spezifiziert [IEE], allerdings können auch andere Funktechnologien und Standards dort verwendet [OLK+21, Dor15, HWW+16]. Als Beispiele sind hierfür ZigBee oder Bluetooth zu nennen, welche auf dem IEEE 802.15.4-Standard basieren, allerdings innerhalb des Frequenzbereichs des 802.11-Standards betrieben werden. Neuere WLAN-Standards wie 802.11n bieten ein besseres Signal-Rausch-Verhältnis und eine zuverlässigere, bessere Übertragung von Multicasts. Die Herausforderungen hinsichtlich des Multicasts bestehen weiterhin, sodass die Internet Engineering Task Force (IETF) eine Konvertierung von Multicasts in Unicasts (Directed Multicast Service) empfiehlt [MP17, Dor15].

Mit der Einführung neuer Mobilfunk-Kommunikationstechnologien wie 5G werden neue Möglichkeiten hinsichtlich der Vernetzung, der Infrastruktur oder auch der Anwendungen geschaffen [Bun20]. 5G wird in lizenzierten Frequenzbändern betrieben [ZKC+15, OLK+21]. Mit der Verwendung von 5G werden auch Anwendungen mit (ultra-)kurzen Latenzzeiten im Bereich von 20 ms ermöglicht [Bun20, S. 19], [HWW+16]. Bei der Infrastruktur besteht die Möglichkeit der Verwendung eines eigenen 5G-Campusnetzes, eines geografisch begrenzten, lokalen 5G-Mobilfunknetzes. Anwendung findet dieses, wenn besondere Anforderungen z. B. an die Zuverlässigkeit oder Verfügbarkeit bei der (industriellen) Kommunikation existieren.

Bei die Verwendung eines 5G-Campusnetzes existieren unterschiedliche Modelle hinsichtlich des Betriebs [Bun20, S. 22]. Eine Möglichkeit ist, dass ein eigenes lokales, privates 5G-Netz betrieben wird – ein maßgefertigtes Mobilfunknetz auf dem eigenen Betriebsgelände. Hier kann der Aufbau bzw. der Betrieb eigenständig oder durch Drittanbieter erfolgen. Das erschaffene 5G-Campusnetz ist ein Netz mit eigener Netz-ID und einer Abgrenzung vom öffentlichen Mobilfunknetz. Der Betreiber hat vollen Zugriff auf die Verwaltungsfunktionen wie bspw. die Dienstgüte. Um Zugriff auf ein privates 5G-Campusnetz zu erhalten, werden Geräte über eine entsprechende SIM-Karte autorisiert. Funk-Koexistenzen mit anderen Funktechnologien oder -geräten, welche innerhalb des 5G-Frequenzspektrums betrieben werden, können somit reduziert oder gänzlich vermieden werden. Auch innerhalb von 5G besteht die Möglichkeit zur Versendung von UDP-Unicasts und -Multicasts. Analog zum IEEE 802.11-Standard wird das Paket auf der physikalischen bzw. der Sicherungsschicht durch eine Point-to-Point-Verbindung umgesetzt [GFC+20, CMF+18]. Derzeit existieren Bestrebungen, um in Zukunft Multicasts bzw. Broadcasts durch Point-to-Multicast-Pakete zu unterstützen [Eur17c, S. 32][CMF+18, TS15]. Diese werden u. a. durch das *3rd Generation Partnership Project* (3GPP) vorangetrieben.

Fahrerlose Transportfahrzeuge benötigen eine drahtlose Übertragungstechnologie, unabhängig davon, ob es sich um 5G oder WLAN handelt, um mit anderen Teilnehmern zu kommunizieren bzw. zu interagieren. Im Folgenden wird daher der

Empfehlung der IETF gefolgt und eine Unicast-Kommunikation zwischen den Teilnehmern umgesetzt.

Kapitel 4
Das Fahrerlose Transportfahrzeug als Betriebsmittel und die Digitalisierung des Transports

"The limits of my language means the limits of my world."

— Ludwig Wittgenstein

Die Produktivität und Effizienz derzeitiger Fertigungssysteme ist i. d. R. durch die Leistungsfähigkeit der Einzelprozesse und Fertigungseinrichtungen begrenzt und kann häufig nur durch Parallelisierung gesteigert werden [KPT+10]. Bei dem immer stärker werdenden Trend nach Individualisierung der Produkte (vgl. Abb. 4.1) ist jedoch eine Erhöhung der Effizienz nicht mehr durch eine Steigerung der Prozessgeschwindigkeit bestimmt [KHM18]. Vielmehr bedarf es für eine hohe Produktionsvariabilität (flexible Fertigungsfunktionalitäten und -wege) folgende Voraussetzung: eine schnelle Wandelbarkeit (Umrüstung und Inbetriebnahme), eine Vernetzung der Komponenten (Re-Konfigurierbarkeit), eine transparente Produktion (durchgehende Weitergabe von Informationen) und eine Skalierbarkeit des Systems [LFK+14, LBK15]. Mobilen Robotern wird hierbei ein erhebliches Potenzial beim innerbetrieblichen Transport beigemessen [Weh20, S. 825 ff.]. Durch das dy-

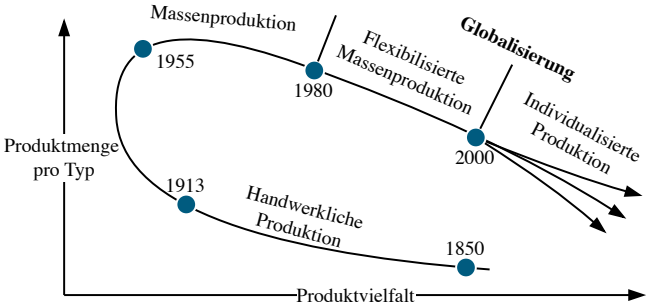


Abb. 4.1: Die Paradigmen in der Fertigungstechnik abhängig von der Produktvielfalt und der Produktmenge pro Produkt, eigene Darstellung basierend auf [Kor10]

namische Hinzufügen und Entfernen von FTF zur Laufzeit kann eine Skalierbarkeit in beide Richtungen (scale up/ down) erreicht werden. FTF können neben einem reinen Transport, bei dem Waren bzw. Ladehilfsmittel z. B. von einer Quelle zu einer Senke bewegt werden, auch bei der Kommissionierung (Ware-zur-Person) [tSB11a, S. 66] eingesetzt werden. Gefragt ist daher ein Ansatz für den Transport, der sich agil auf unterschiedliche Anforderungen anpassen lässt. Dies bedeutet, dass sich ein Transport nach einer Änderung oder Umrüstung der Prozesslinie selbst kalibriert und die Transporteinheiten (hier: FTF) sich selbstständig durch Kommunikationskanäle über die zu leistenden Transportfolgen abstimmen.

Wesentliches Ziel dieses Kapitels ist daher die Entwicklung eines ganzheitlichen Informationsmodells, welches für eine Selbstorganisation des Transports benötigt wird. Begonnen wird hierbei mit der Erschaffung eines Informationsmodells für ein FTF, mit dem sich seine Fähigkeiten und angebotenen Dienste beschreiben lassen (vgl. Abschn. 4.1). Als Bindeglied zwischen einem FTF und einem Transportauftrag wird der Transport aus Sicht des Fahrerlosen Transportfahrzeugs formalisiert (vgl. Abschn. 4.2), bevor die Anforderungen an eine formale Sprache erhoben werden, welche notwendig sind für eine Beschreibung des Transportprozesses. Anschließend erfolgt die Einführung der digitalen Repräsentation eines Transportauftrags und die Umsetzung dieser (vgl. Abschn. 4.3 – 4.5). Die digitale Repräsentation des Transports kennt dadurch die Anforderungen an sich selbst, um die anstehende Transportaufgabe durchzuführen und sich selbst fertigstellen zu können.

Werden die Selbstbeschreibungen des Betriebsmittels und des Prozesses zusammengeführt, entsteht ein ganzheitliches Informationsmodell, welches die Grundlage für einen selbstorganisierenden Materialfluss ist. Jeder Transport ist dabei eine eigenständige Entität (vgl. Abschn. 3.1) mit dem Ziel, sich selbstständig fertigzustellen. Dabei kann diese digitale Repräsentation mehrere Betriebsmittel kontaktieren und individuell über eine Ausführung verhandeln.

4.1 Ein Informationsmodell für das Betriebsmittel Fahrerloses Transportfahrzeug

Fahrerlose Transportfahrzeuge können zentral oder dezentral organisiert werden (vgl. Abschn. 2.2). Bei der zentralen Organisation kennt die Leitsteuerung – auch Flottenmanagementsystem (FMS) genannt – die Fähigkeiten und die Zustände aller FTF, um das bestmögliche FTF für eine Transportaufgabe zu finden. Dabei sind die von Unternehmen benötigten FMS-Operationen oft vertraulich [BVF+16]. Dies impliziert eine Herstellerabhängigkeit und erzeugt somit einen Bedarf für einen (offenen) Standard, um schnell neue Fahrzeuge in eine Leitsteuerung integrieren zu können. Die VDA5050, eine Standardisierungsinitiative des Verbands der Automobilindustrie e.V. (VDA), versucht dies zu ändern [Ver20]. Der Standard *VDA5050* beschreibt den Austausch von Auftrags- und Statusdaten zwischen einer zentralen Leitsteuerung und einem FTF für intralogistische Transporte [Ver20]. Das FMS

setzt dabei voraus, dass alle Fahrzeuge bereits im System zur Ausführung bekannt sind.

Ein Eckpfeiler der Industrie 4.0 ist die dezentrale Entscheidungsfindung, um Komplexität zu bewältigen. Wird dies nun auf das FMS angewandt, so bedeutet dies, dass jedes FTF sich selbst repräsentiert und die zentrale Koordinierungseinheit entfällt. Zusätzlich organisiert jeder Transportauftrag als eigene Entität sich selbst. Das bedeutet, dass er den Auftrag dem FTF zuweist, welches für den Transport aus seiner lokalen Sicht am geeignetsten ist [BLS+17]. Damit ein FTF einen Transport annehmen kann, muss dieses sich seiner Fähigkeiten und der Anforderungen des Transports bewusst sein.

Zur Vermeidung einer überflüssigen Kommunikation kann in beiden Fällen eine Vorauswahl (bzw. Filterung) basierend auf dem Informationsmodell des FTF erfolgen. Es wird lediglich mit den Fahrzeugen kommuniziert, welche für einen Auftrag relevant sind.

Innerhalb des Forschungsprojektes *Logistics for Manufacturing SMEs* (L4MS)[1] wurde die IoT-Plattform *Open Platform for Innovations in Logistics* (OPIL) entwickelt [SPP+19]. OPIL ist eine auf die Intralogistik zugeschnittene zentral organisierte Plattform mit dem Ziel, eine schnelle und einfache Integration von FTF innerhalb einer Produktion zu ermöglichen. Innerhalb dieser Plattform wurde eine ROS-basierte Auftragssteuerung beschrieben, die jedoch nicht auf intralogistische Prozesse ausgelegt ist, sondern auf Bewegungen und Aktionen [QNF20].

Darüber hinaus kann ein Informationsmodell zur Integration von neuen Fahrzeugen zur Ausführungszeit verwendet werden: Ein Fahrzeug propagiert automatisch seine Existenz im Netzwerk mithilfe seines Informationsmodells. Interessierte Entitäten erhalten das FTF-Informationsmodell und können dies für ihre Zwecke verwenden. Eine manuelle Konfiguration entfällt, sodass eine flexible und schnelle Skalierung an Fahrzeugen zur Laufzeit möglich ist. Ein standardisiertes Informationsmodell gewährleistet auch eine Interoperabilität und somit die Aufhebung einer Herstellerabhängigkeit. [DPF+19]

In der folgenden Abb. 4.2 ist das Datenmodell mithilfe eines UML-Klassendiagramms für ein FTF dargestellt, basierend auf den Vorarbeiten des Autors aus [DPF+19]. Das hier aufgeführte Datenmodell besteht aus den drei vereinfachten Komponenten *DeviceDescription*, *DeviceProperties* und den Beschreibungen der *Services* selbst. Ein *Service* kann aufgerufen werden, indem *addOrder(Service)* verwendet wird. Mithilfe des Services *estimateOrder()* erfolgt eine Schätzung für einen Auftrag, ohne dass dieser vom FTF ausgeführt wird. Bei der *OrderResponse* handelt es sich um ein Objekt, welches beim Aufruf eines *Services* zurückgegeben wird. Im Folgenden werden die drei vereinfachten Komponenten *DeviceDescription*, *DeviceProperties* und die Beschreibungen der *Services* erläutert.

[1] EU-Forschungsprojekt Logistics for Manufacturing SMEs (L4MS): http://www.l4ms.eu
Laufzeit: Okt. 2017 - März 2021, Grant Agreement 767642: https://cordis.europa.eu/project/id/767642

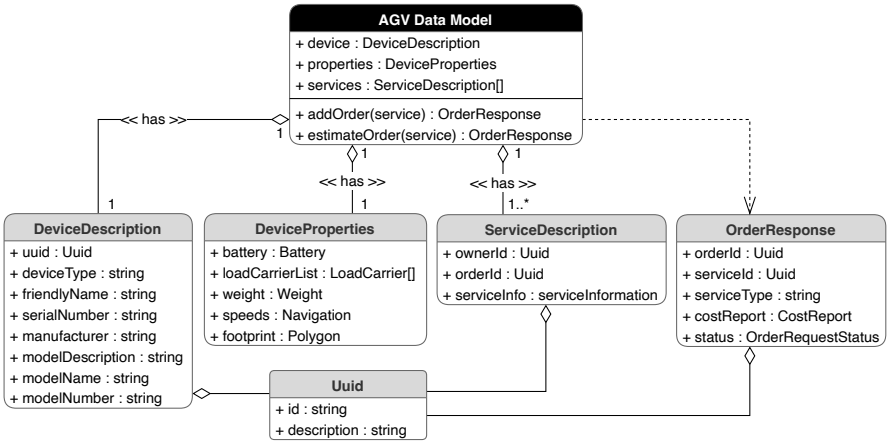


Abb. 4.2: Vereinfachtes Datenmodell eines Fahrerlosen Transportfahrzeugs (engl.: *automated guided vehicle (AGV)*) bestehend aus den drei Kategorien *DeviceDescription*, *DeviceProperties* und *ServiceDescription*

DeviceDescription

Die *DeviceDescription* repräsentiert die Gerätebeschreibung, welche vom Hersteller verfasst wird, und enthält herstellerspezifische Informationen und Definitionen. Die erforderlichen Attribute in der *DeviceDescription* sind die *uuid* und der *deviceType*: Bei der *uuid* handelt es sich um eine eindeutige Kennung, welche durch den Hersteller gesetzt wird. Diese kann als global eindeutig angenommen werden, als eine eigene eindeutige Kennung für ein FTF. Mithilfe vom *deviceType* kann beschrieben werden, ob es sich um eine (physische oder logische) Einheit handelt, welche die entsprechenden Dienste anbietet. In diesem Beispiel ist es ein *FTF*. Optionale, herstellerabhängige Attribute sind eine Modellbeschreibung des Fahrzeugs (*modelDescription*, *modelName* und *modelNumber*) oder die Angabe des Herstellers (*manufacturer*).

DeviceProperties

Die Geräteeigenschaften *DeviceProperties* sind spezifisch für die physische Einheit des FTF, wie in Abb. 4.3 dargestellt. Diese umfasst u. a. die Eigenschaften der Batterie *Battery*, wie die maximale Reichweite, die ein FTF mit einer vollen Ladung erreichen kann. Darüber hinaus hat jedes FTF mindestens einen oder mehrere Ladungsträger *LoadCarrier*, die transportiert werden können. Für jeden *LoadCarrier* kann ein einzelner Transportservice aufgerufen werden. Der Typ des Ladungsträgers *LoadCarrierType* beschreibt, ob es sich um eine Palette, einen Kleinladungsträger *SmallLoadCarrier*, einen Großladungsträger *BigLoadCarrier* usw. handelt. Eine automatische bzw. die manuelle Beladung ist innerhalb des *LoadHandling* definiert.

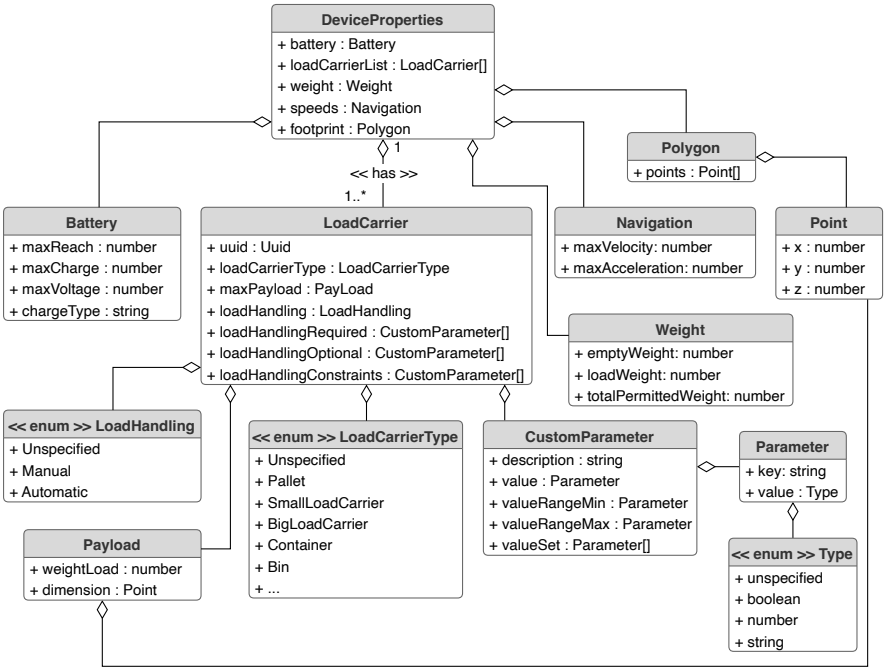


Abb. 4.3: Modellierung der physikalischen Eigenschaften eines Fahrerlosen Transportfahrzeugs wie z. B. der Batterie, der Handhabung von Ladungsträgern, der physikalischen Abmessungen oder des Gewichts

Die (physische Dimension) Stellfläche *Payload* beinhaltet die Höhe, die Breite, die Tiefe und das zu tragende Gewicht. Als Beispiele für FTF mit mehreren Ladungsträgern lassen sich hier Magazinos[2] *Toru* bzw. *Soto* und Hairobotics[3] *HAIPICK* nennen. Einschränkungen bzw. Restriktionen, wie die maximale Ladehöhe bei einem Gabelstapler, können innerhalb der *loadHandlingConstraints* definiert werden. Dies kann als ein *CustomParameter* mit dem Namen *standardLiftingHeight* definiert sein. In den *loadHandlingRequired* sind die Parameter beschrieben, die bei dem Aufruf eines Transport-Services erforderlich sind.

ServiceDescription

Services sind Dienste, die von einem (physikalischen) Gerät oder einer Software angeboten werden, wie in der Abb. 4.4 dargestellt. Die Beschreibung eines Services definiert Aktionen, deren Argumente sowie deren Datentypen und Wertebereiche. Verlässt das Gerät bzw. die Software das Netzwerk, so ist dieser Service

[2] Magazino: https://www.magazino.eu

[3] Hairobotics: https://www.hairobotics.com

auch nicht mehr verfügbar. Jeder Service hat eine eindeutige Kennung (*serviceId*), einen Namen (*serviceType*) und eine Liste an Events, die zu dem Service erstellt werden (*events*). Diese drei Attribute sind in den *ServiceInformationen* zusammengefasst und werden nicht benötigt, wenn ein Auftrag ausgeführt soll. Die folgenden beschriebenen Attribute sind für den Aufruf eines Auftrags relevant: Für einen Serviceaufruf wird eine eindeutige Auftragsnummer (*orderId*) und eine eindeutige Kennung für den Initiator (*ownerId*) benötigt. Die vier vorgestellten Services (*TransportOrderService*, *TransportOrderStepService*, *MoveOrderService* und *ActionOrderService*) sind als elementare Services in einem Logistikzentrum bzw. in einem CPPS anzusehen [Sch18a, S. 147].

Jede Aktion (*Action*) bezieht sich auf einen *LoadCarrier* und dessen *Uuid* (*loadCarrierUuid*). Zusätzlich müssen die erforderlichen Parameter (*LoadHandlingParamter*) aus den *DeviceProperties* für den jeweiligen *LoadCarrier* angegeben sein. Der *ActionType* beschreibt, ob es sich um ein Be- (*Load*) bzw. Entladen (*Unload*) handelt, während die *Payload* sich auf das aktuelle Gewicht und die Abmessungen des Ladungsträgers (vgl. Abb. 4.3) bezieht. Bei dem Transport (*TransportOrderServices*) eines Ladungsträgers von einer Quelle zu einer Senke muss die *loadCarrierUuid* übereinstimmen.

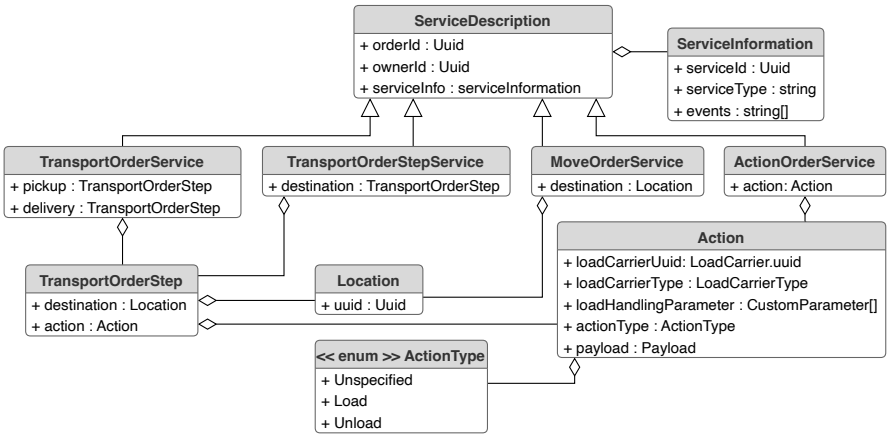


Abb. 4.4: Angebotene Services eines Fahrerlosen Transportfahrzeugs

Zustandsänderungen eines FTF

Services bzw. FTF können ihre Zustände und die damit verbundenen Änderungen an interessierte Entitäten senden. Zu diesem Zeitpunkt wurde bereits ein FTF von anderen Entitäten gefunden und der angebotene Service genutzt. Ereignismeldungen enthalten die Namen von einer oder mehreren Zustandsvariablen und den aktuellen Wert dieser Variablen (vgl. Abb. 4.5). Diese Ereignismeldungen für eine

Order beziehen sich auf den Zustand der ausgeführten Services (*TransportOrderStatus*, *TransportOrderStepStatus*, *MoveOrderStatus* oder *ActionOrderStatus*). Die *OrderQueue* gibt eine Übersicht über die kommenden Aufträge. Hier wird neben dem aktuellen Auftrag (*ongoing*) und den als nächstes anstehenden Aufträgen (*upcomingOrderList*) auch die Belegzeit (*occupiedUntil*) und die Endposition des letzten Auftrags (*finalLocation*) mitgeteilt. Im *AGVStatus* werden u. a. allgemeine Informationen hinsichtlich der Position (*Location*), der Ausrichtung (*Pose*) oder der verbleibenden Kapazität der Batterie (*battery*) offengelegt. Sowohl die *OrderQueue* als auch der *AGVStatus* können verwendet werden, um kommende Aufträge hinsichtlich ihrer Durchlauf- und Ausführungszeit zu optimieren. Einige Zustandsvariablen können sich auch mit einer hohen Aktualisierungsrate ändern. Diese Zustandsänderungen können das Kommunikationsnetzwerk belasten, sofern diese mit der gleichen Frequenz an andere Entitäten geschickt werden. Abhilfe kann hier eine Filterung bzw. eine Änderung der Mitteilungsfrequenz schaffen. Auch Zustandsvariablen mit einer großen Datenmenge sollten hier möglichst vermieden werden.

Basierend auf den Informationen des Zustandes *QueueOrder* und dem *AGVStatus* eines Fahrzeugs kann eine stetige Optimierung zur Laufzeit erfolgen: Ein bereits zugewiesener Auftrag kann von einem anderen Fahrzeug übernommen werden. Dafür wird ein weiterer Service für die Übernahme von Transportaufträgen benötigt. Im Rahmen der Selbstorganisation verwaltet ein Fahrzeug sich und seinen Zustand selbst. Sobald die Batterie eine gewisse Kapazität unterschritten hat, fährt es autonom zu einer Ladestation, um sich aufzuladen [WSF+17].

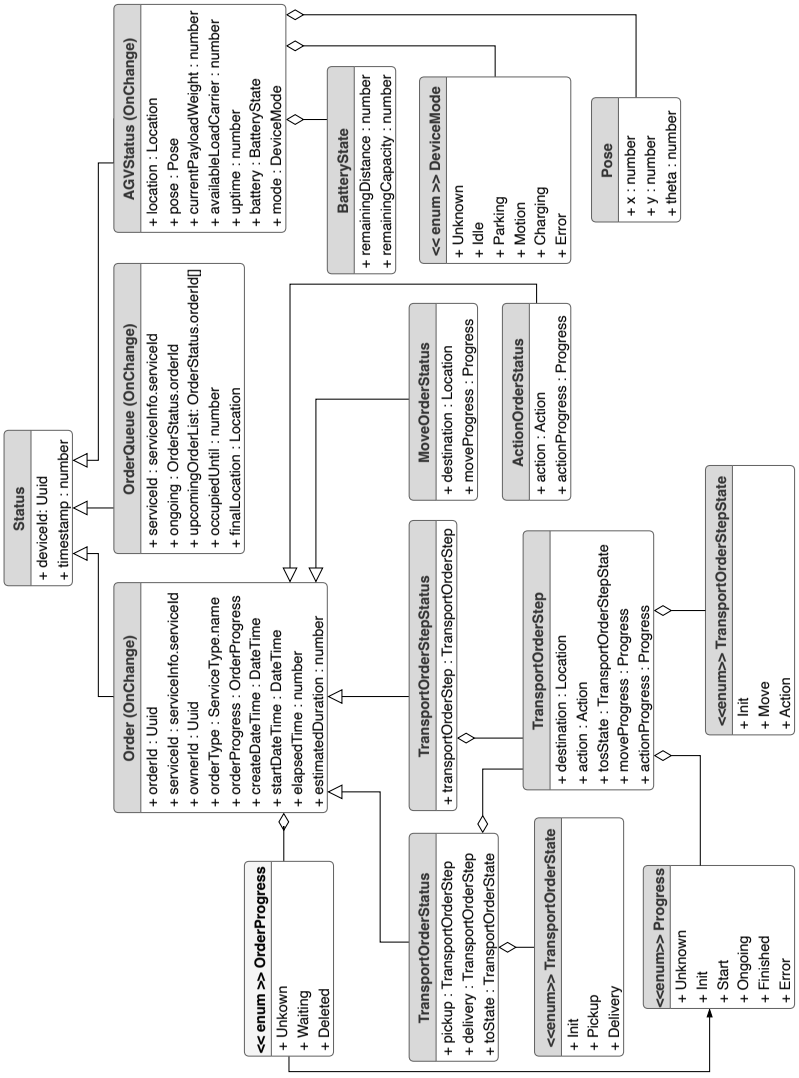


Abb. 4.5: Übersicht der angebotenen Statusänderungen eines Fahrerlosen Transportfahrzeugs (FTF), welche andere Entitäten bzw. Teilnehmer abonnieren können

4.2 Formale Aufgabenbeschreibung für Fahrerlose Transportfahrzeuge

Im vorherigen Abschnitt wurde das Informationsmodell des intralogistischen Betriebsmittels FTF hinsichtlich seiner Fähigkeiten, Funktionen und seines Zustandes beschrieben. In diesem Abschnitt werden der formale Transportauftrag und die Kriterien für eine formale Sprache zur Beschreibung des Transportprozesses definiert. Entlang des formalen Transportauftrags und der hier definierten Kriterien wird im nächsten Abschn. 4.3 eine deklarative Sprache entwickelt.

Der Transportauftrag für ein FTF wird entweder manuelle oder automatisch generiert [HSD18, S. 218]: Die manuelle Auftragsgenerierung erfolgt durch einen Menschen. Bei der automatischen Auftragsgenerierung wird der Transportauftrag durch einen anderen Teilnehmer innerhalb des CPPSs, wie zum Beispiel einem ERP oder ein Warehouse Management System, generiert. Die Möglichkeit, dass eine Arbeitsstation eine bedarfsgerechte Anforderung von Material initiiert und somit einen Transportauftrag erzeugt, besteht ebenfalls [BLS+17].

4.2.1 Der formale Transportauftrag

Die Auftragsbeschreibung eines Transportes, auch Beförderungsauftrag genannt, „[...] gibt vor, zu welcher Abholzeit Z_{abi} eine Ladungsmenge oder Fracht mit M_{LE} Ladeeinheiten oder Beförderungseinheiten an welchem Abholort S_i zu übernehmen ist und bis zu welcher Anlieferzeit $Z_{an\ j}$ die Ladung an welchem Zielort S_j abzuliefern ist.“ [Gud10, S. 774 f.]. Ein Transportauftrag (TA) (*TransportOrder (TO)*) ist dafür vorgesehen, Güter von einer bestimmten Quelle zu einem bestimmten Ziel zu einem bestimmten Zeitpunkt zu bewegen [HH11]. Eine Möglichkeit der Abbildung eines Transportauftrags als eine atomare Operation kann daher wie folgt sein:

$$TransportOrder := 2 \times TransportOrderStep$$

In diesem Kontext bedeutet atomar, dass ein begonnener Transportauftrag nicht durch einen anderen Transportauftrag unterbrochen werden kann.[4] Dies impliziert, dass ein Transportauftrag beendet werden muss, bevor ein weiterer Transportauftrag ausgeführt werden kann. Der Transportauftrag selbst kann dabei als Komposition aus zwei Transportauftragsschritten (*TransportOrderStep (TOS)*) aufgefasst werden:

$$TransportOrderStep := MoveOrder \vee ActionOrder$$

Der TOS ist dabei ein Fragment, welches aus einer Bewegungsanweisung (*MoveOrder (MO)*) zu einer Position oder aus einer Aktionsanweisung (*ActionOrder (AO)*) bestehen kann. Das Ziel einer MO ist entweder als Abholungs- oder dem Lieferungsort zu verstehen. Bei der AO ist das Laden (bei

[4] Dies hat keinen Einfluss auf die Reihenfolge der nachfolgenden Transportaufträge.

der Abholung) und das Entladen (bei der Lieferung) gemeint. Hieraus resultiert die Definition für eine MO wie folgt:

$$MoveOrder := GoToPosition$$

Die MO entspricht somit der Bewegung von der aktuellen Position zu einer Zielposition. Eine MO muss jedoch nicht zwangsläufig ausgeführt werden, wenn zum Beispiel die Abholung des nächsten Transportauftrags der Position der letzten Lieferung entspricht. Schließlich wird noch die AO, abhängig von der aktuellen Position, benötigt:

$$ActionOrder := Load \veebar Unload$$

Bei der Abholung wird das FTF beladen, bei einer Lieferung wird das FTF entladen.

In Abb. 4.6 ist ein exemplarischer Aufbau einer Fabrikhalle dargestellt. In dieser Fabrikhalle existieren zwei Bereiche (*Produktion* und *Warenlager*) und ein FTF. Innerhalb dieser Bereiche gibt es Positionen, denen logische Namen zugeordnet werden können. Die Position *Lieferung* ist im Bereich der Produktion, während die Position *Abholung* innerhalb des Warenlagers liegt. Wenn eine Palette aus dem Warenlager von der Position *Abholung* abgeholt werden soll, so muss das FTF zunächst zu dieser Position fahren. Sobald das FTF angekommen ist, wird mit der Beladung begonnen. Nach dem Beladen kann das FTF zum Lieferort *Lieferung* fahren. Die Entladung erfolgt nach Ankunft am Lieferort. In diesem Beispiel ist die Navigation des FTF zwischen den beiden Positionen vernachlässigt.

Dieses Beispiel lässt sich beim Transport sowohl von Stückgutfracht in diskreten Ladeeinheiten bzw. Gebinden als auch auf Einzelgut anwenden. Diese Arbeit fokussiert sich auf den Transport von diskreten Ladeeinheiten wie beispielsweise Paletten oder Kleinladungsträgern.

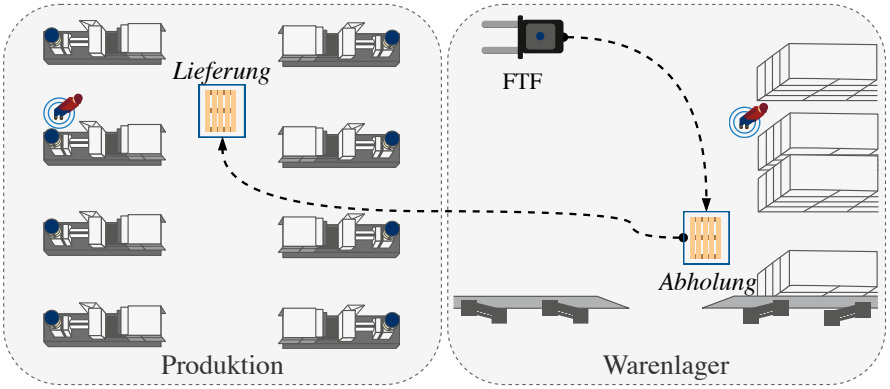


Abb. 4.6: Eine Fabrikhalle aufgeteilt in zwei Bereiche (*Produktion* und *Warenlager*). Innerhalb dieser Bereiche gibt es Positionen (*Abholung* und *Lieferung*).

4.2.2 Kriterien für eine formale Sprache in der Intralogistik

Ein Transportauftrag kann mithilfe einer Programmiersprache abstrahiert oder generiert werden. In der Programmierung existieren verschiedene Paradigmen, wobei zwei davon konträr zueinander stehen: die *imperative* und die *deklarative* Programmierung [VRH04, S. 406 f.]. Bei der *imperativen* Programmierung steht das *Wie* im Fokus, also eine Folge an Anweisungen. Bei der *deklarativen* Programmierung wird hingegen beschrieben, *was* gemacht werden soll, während eine zusätzliche Komponente diese Aufgabe löst. Dies geht einher mit Kowalskis Gleichung $Algorithm = Logic + Control$, welche die Logik (engl.: *logic*) losgelöst von der Steuerung (engl.: *control*) definiert [Kow79]. *Deklarative* Programmiersprachen können nach [VRH04, S. 114] auch auf mathematischer Logik beruhen. Erwähnenswerte Lösungen sind hier die *Temporale Logik* [Pnu77] oder die *Hierarchical Task Networks* (HTN) [NAI+03]. *Simple Temporal Networks* (STN) [VB] oder *Linear Temporal Logic* (LTL), zwei Ausführungen von temporaler Logik, setzen ein Expertenwissen voraus, um Mehrdeutigkeiten und somit Fehler zu vermeiden [KFP07]. Eine Benutzerfreundlichkeit ist hierbei nicht gegeben, welche aber ein wichtiger Bestandteil für die Kollaboration und Interaktion mit dem Menschen ist [KJU+18]. Darüber hinaus haben Logik-basierte Sprachen den Nachteil, dass diese die Erhebung und Darstellung von domänenspezifischem Wissen nicht unterstützen, deine domänenspezifische Sprache (engl.: *domain-specific language (DSL)*) hingegen schon [Dec98]. Die Entwicklung einer neuen Programmiersprache resultiert meist aus dem Bedarf, ein bestimmtes Problem einfach lösen zu wollen. Unabhängig davon, ob es sich um eine General-Purpose Language (GPL), eine DSL oder eine anderweitige Programmiersprache handelt, ist diese an bestimmte Randbedingungen und Anforderungen gebunden. Häufig existiert hierbei ein Wechselspiel zwischen Anforderungen und Randbedingungen. Im Folgenden werden sowohl die wichtigsten formalen Anforderungen an eine Programmiersprache, die sich über die Zeit in der Entwicklung von Software manifestiert haben [Mar08, GHJ+94, Lap17], als auch die Anforderungen zur Beschreibung des Materialflusses, welche zur Laufzeit bzw. für die Ausführung notwendig sind, beschrieben.

Formale Anforderungen

Die folgenden Kriterien wirken sich auf die softwaretechnischen Aspekte der zu entwickelnden DSL aus. Darüber hinaus kann es vorkommen, dass entsprechende Funktionen erst in der Ausführungseinheit bzw. erst zur Ausführungszeit der entwickelten Sprache implementiert werden können.

Modularisierung

Die Modularisierung korreliert mit der Trennung von Zuständigkeiten (engl.: *separation of concerns principle*) (SoC) [Lap17, S. 85]. Ein Modul erledigt genau eine Aufgabe. Daher hängt die Modularisierung auch eng mit dem *Single Responsibility Principle* zusammen, welches einem Objekt, einer Klasse oder einer Methode genau eine Aufgabe zuweist [Mar08, S. 181].

Komposition

Bei der Komposition werden einfache Bausteine verwendet, um durch die geeignete Kombination dieser Objekte mit einer höheren Komplexität zu definieren [Par03b]. So besteht zum Beispiel ein Transportschritt aus einer Bewegung zu einem Ort oder einer Aktion, die an dieser Position ausgeführt werden soll. Kompositionen ermöglichen auch eine Wiederverwendung von bereits definierten Bausteinen. Zusätzlich lassen sich durch Kompositionen komplexere Objekte bzw. Bausteine erstellen.

Erweiterbarkeit und Anpassung

Während der Entwicklung einer Sprache ist es nahezu unmöglich, alle Elemente zu bestimmen, die eine Sprache zu ihrer Laufzeit benötigen wird. Hierfür existiert das *Open-Closed-Principle* [Mey97, S. 57 ff.]. Dieses Prinzip sieht vor, dass die Software offen für Erweiterungen ist, jedoch geschlossen für Veränderungen. Erweiterungen betreffen in diesem Zusammenhang Datenstrukturen oder Operationen, während die bereits definierte Funktionsweise nicht verändert wird.

Negierung

Eine Negierung ermöglicht die Beschreibung eines Zustandes, welcher nicht *wahr* in einem bestimmten Kontext sein darf. So darf eine Aktion (hier: Transport) erst ausgeführt werden, wenn eine bestimmte Bedingung nicht eingetreten ist.

Eindeutigkeit der Ausdrucksfähigkeit

Die Grammatik einer Programmiersprache muss eindeutig und präzise sein. Eindeutigkeit einer Programmiersprache bedeutet, dass sowohl die Syntax als auch die Semantik keine weitere Deutung erlauben und somit nicht zu Missverständnissen führen. Die Syntax wird durch das Vokabular und die Grammatik beschrieben. Die Semantik hingegen spiegelt die Bedeutung wider. Sowohl der Mensch als auch ein Computer (*Interpreters* oder *Compilers*) brauchen eine eindeutige Sprache.

Logistische Anforderungen zur Ausführungszeit

Die Hauptaufgabe der Logistik fassen [HSD18, S. 340] in einer *6r*-Regel zusammen: Die *richtige* Ware wird zur *richtigen* Zeit, am *richtigen* Ort, in der *richtigen* Menge, in der *richtigen* Qualität und zu den *richtigen* Kosten zur Verfügung gestellt. Eine Erweiterung um eine weitere Regel dieser Definition, ist die Darstellung der *richtigen* Information [MFH01]. Die Logistik selbst ist vollständig deterministisch und algorithmierbar [tH20, S. 5]. Diametral zu der Logistik steht ihre Umgebung, z. B. in Form der Verkehrsbedingungen. Auch der Zeitpunkt, wann ein Transport initiiert wird, ist unter bestimmten Bedingungen unvorhersehbar.

Wird eine Massenproduktion betrachtet, so beinhaltet diese ein hohes Produktionsvolumen pro Variante [Kor10]. Produktionspläne existieren für einen gewissen Zeitraum in der nahen Zukunft. Folglich ist auch die Logistik, unter Berücksichtigung einer fest vorgegebenen Prozessabfolge und der Durchlaufzeit, hinsichtlich der *6r*-Regel planbar. Bei der individualisierten Produktion mit einer Losgröße 1 ist aus logistischer Sicht das optimale Betriebskonzept der *One-Piece-Flow* [ABB$^+$11, S. 29]. Hier existieren keine Liegezeiten, sodass die Ware bzw. das Werkstück nach Beendigung der Verarbeitung bzw. Bearbeitung an der entsprechenden Station abgeholt wird. Dies hat eine Auswirkung auf die Komplexität des Prozesses hinsichtlich seiner Durchlaufzeit und der Planung [Kor10].

Bei der individualisierten Produktion hat ein Kunde die Möglichkeit, sein Produkt nach seinem Bedarf zu konfigurieren (z. B. in einem Webshop). Auch hier ist die Logistik deterministisch hinsichtlich des Transports eines Guts von der Quelle zur Senke. Die Initiierung kann zu einem beliebigen Zeitpunkt erfolgen. Bei der Herstellung eines individuellen Produkts müssen verschiedenste Prozesse und Randbedingungen berücksichtigt werden. Die Kombinationen verschiedenster Prozesse unter Berücksichtigung bestimmter Bedingungen ist ein multikriterielles Optimierungsproblem [Sch18b], welches nicht Teil dieser Arbeit ist. Nachfolgend werden die logistischen Anforderungen, welche zur Ausführung notwendig sind, beschrieben.

LA1: Unabhängigkeit vom Fördermittel

Wie bereits im Abschnitt 2.2 erläutert, wird in der Logistik zwischen verschiedenen Fördermitteln unterschieden. Die explizite Zuweisung eines Fördermittels kann zur Beschreibung des Transports erfolgen oder zur Laufzeit, wenn der Transport ausgeführt werden soll. Eine Zuweisung eines Fördermittels während der Beschreibung würde eine imperative Eigenschaft bedeuten. Bei einer Selbstorganisation eines CPPS ist nicht relevant, welches Fördermittel (oder welcher Service) den Transport erfüllt. Wichtig ist, dass die Aufgabe erfüllt wird und der Transport erfolgt. Vielmehr würden auch Informationen, die zu dem Zeitpunkt der Zuweisung noch nicht vorhanden sind, zu einer schlechteren Lösung führen. Erfolgt die Zuweisung allerdings erst zur Laufzeit (der Transport ist eine eigene Entität innerhalb des CPPS) durch eine FTF-Leitsteuerung, einen Optimierungsdienst oder durch sich

selbst (Selbstorganisation) [BLS+17], so können alle verfügbaren Informationen zu diesem Zeitpunkt verwendet werden.

LA2: Unabhängigkeit vom Ladehilfsmittel

Ladehilfsmittel wie Paletten, Gitterboxen oder Kartonagen ermöglichen es, eine Vielzahl an Objekten gleichzeitig zu transportieren. Allerdings existiert eine Wechselbeziehung zwischen den Kosten für den Transport und den Kosten für die Bildung der Ladeeinheit mithilfe des Ladehilfsmittels [Fel18, S. 13 ff.]. Befinden sich mehrere Objekte auf einem Ladehilfsmittel, so reduzieren sich die Kosten pro Objekt in jedem Transport. Konträr hierzu sind die Kosten zur Verwaltung der Ladehilfsmittel, da diese zunächst vorbereitet und ggf. nach Beendigung des Transports zurück transportiert werden müssen. Fahrerlose Transportfahrzeuge können neben einem Ladehilfsmittel auch nur eine Ladefläche besitzen, sodass diese auch beschrieben werden muss.

LA3: Transportketten

Wird bei einem Transport das Arbeitsmittel gewechselt, so wird dies auch als *Umschlagen* bezeichnet [HSD18, S. 308]. Ein Umschlag kann notwendig sein, wenn der Transport zwischen zwei räumlich getrennten Orten zu realisieren ist. Dies ist der Fall, wenn ein FTF nur für einen bestimmten Bereich zugelassen ist, während ein anderes FTF dies nicht ist. Durch eine bestimmte Reihenfolge bei der Verkettung von Transporten können unnötige Schritte vermieden werden [TWB+10, S. 177].

Eine andere Form der Verkettung ist die Pickliste bei der Kommissionierung. Im Rahmen der Kommissionierung werden dabei Waren aus einem bereitgestellten Sortiment anhand von vorgegebenen Aufträgen zusammengestellt [Gud10, S. 685]. Bei einer Pickliste handelt es sich um eine auftragsgerechte Zusammenstellung von Teilmengen aus einem gegebenen Artikelsortiment von ggf. unterschiedlichen Orten, die zu einem gemeinsamen Zielort transportiert werden sollen.

LA4: On-demand und Vorausplanung

On-demand-Transporte sind Transporte, die eine zeitnahe Ausführung haben. Diese werden bei Bedarf ausgeführt, sodass eine Anlieferung weder zu früh noch zu spät erfolgt. Aber auch die Vorausplanung ist von Relevanz. Wenn an einer Arbeitsstation die Durchführung einer Bearbeitung eine gewisse Zeit t benötigt, so kann bereits ein Transport vorausgeplant und in Zukunft berücksichtigt werden. Mit einer Vorausplanung wird auch die Liegezeit reduziert, ein wichtiges Kriterium beim *One-Piece-Flow*.

LA5: Automatische und manuelle Lastaufnahme

Die vollständige Automatisierung und die menschenleere Fabrik sind aus technologischen und ökonomischen Gründen kein realistisches Szenario [Hir15]. In diesem Kontext wird von hybriden Fertigungszellen gesprochen, in denen Mensch und Maschine flexibel und sicher zugleich kollaborieren [GVB16]. Dies kann beim Umschlag der Ladung besonders wichtig sein. Beim Umschlag der Ladung, also dem Be- oder Entladen, kann eine Sicherung bzw. Entsicherung der Ladung notwendig sein, welche durch den Menschen durchgeführt werden muss.

LA6: Definition von Randbedingungen und Anforderungen

In einem Transportauftrag können zusätzliche Anforderungen durch den Transport definiert werden. Beispielhaft lassen sich hierfür Verderblichkeit, Empfindlichkeit oder Explosionsgefahr nennen [Gud10, S. 775]. Darüber hinaus können auch die maximalen Kosten, zu denen der Transport durchgeführt werden soll, von Bedeutung sein. Weitere Randbedingungen sind die maximale vorgegebene Transportzeit, der früheste Startpunkt und der späteste Fertigstellungszeitpunkt (engl.: *deadline*) [DRVD20].

4.3 Eine deklarative Programmiersprache für den intralogistischen Transport

In diesem Abschnitt wird die *Material Flow Description Language (MFDL)* vorgestellt, eine high-level-deklarative domänenspezifische Sprache, mit der die Materialflussoperation eines Transports hinsichtlich seiner Ausführung, Flusskontrolle, Verkettung oder Einschränkungen und Restriktionen abgebildet werden kann. Die *MFDL* basiert sowohl auf der *Logistic TAsk LANguage (LoTLan)* [DKJ19] als auch der *Production Flow Description Language (PFDL)* [Hör22, DEH+22], den Vorarbeiten des Autors.

Begonnen wird dieser Abschnitt mit einer Einführung in domänenspezifische Sprachen und ihre Eigenschaften. Aufbauend darauf werden die bisher existierenden Lösungen im Bereich der Logistik und Robotik vorgestellt und untersucht, wie sich diese mit den logistischen Anforderungen aus dem vorherigen Abschnitt 4.2.2 abbilden lassen. Im Anschluss folgt die Einführung einer formalen Grammatik von *MFDL*, welches die Grundlage für den Abschnitt 4.4 ist, in dem die Terminologie und einige rudimentäre Beispiele vorgestellt werden.

4.3.1 Domänenspezifische Sprachen

Eine generische Programmiersprache (engl.: *General-Purpose Language (GPL)*) bietet die Möglichkeit, ein möglichst breites Spektrum von Problemen für diverse Domänen zu lösen, mit einer begrenzten Notation und Abstraktion [vKV00]. Bekannte GPLs sind zum Beispiel *Python*, *Java*, *C/ C++* oder *C#*. Programmiersprachen wie *C* und *C++* bieten Möglichkeiten, auf einen Speicherbereich durch Zeiger zuzugreifen. Ein Speicherzugriff bringt jedoch keinen Vorteil bei der Beschreibung eines Transportauftrags in der Logistik. Ältere Programmiersprachen wie zum Beispiel Cobol, Fortran oder Lisp wurden ursprünglich für ein bestimmtes Problem bzw. eine bestimmte Domäne entwickelt [vKV00]. Schrittweise wurden diese um weitere Funktionalitäten erweitert, sodass diese nicht mehr ausschließlich Anwendung in ihrer ursprünglichen Domäne gefunden haben, sondern generisch in unterschiedlichen Domänen eingesetzt werden konnten.

Wird eine Sprache genau für eine bestimmte Domäne entwickelt, so handelt es sich um eine DSL [MHS05]. Eine DSL kann ein Problem bzw. mehrere Probleme einer Domäne lösen [Voe13, S. 29]. Diese Eigenschaft, ein *spezifisches* Problem einer Domäne zu lösen, steht entgegengesetzt zu der Eigenschaft der *generischen* Programmiersprache. Abb. 4.7 veranschaulicht, wie eine GPL auf verschiedene Domänen angewendet werden kann; eine DSL hingegen findet Anwendung nur explizit für eine Domäne. Kann ein Problem innerhalb einer Domäne nicht mit einer GPL oder DSL abgebildet werden, so besteht der Bedarf an einer Erweiterung der GPL bzw. der DSL. Alternativ besteht auch die Option zur Schaffung einer neuen

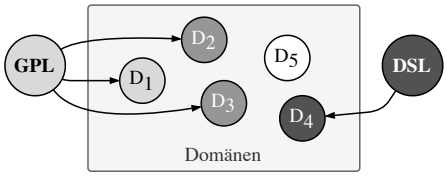


Abb. 4.7: Anwendung einer General-Purpose Language (GPL) und einer domänenspezifischen Sprache (DSL) auf verschiedene Domänen. Die GPL ist anwendbar auf mehrere Domänen ($D_1 \ldots D_3$); die DSL nur auf eine einzige Domäne (D_4). Für die Domäne D_5 existiert weder eine GPL noch eine DSL.

Programmiersprache. Innerhalb einer DSL oder einer GPL besteht die Möglichkeit, ein Problem auf unterschiedliche Weisen zu lösen [Voe13, S. 58].

DSLs, auch *microscopic* oder *little languages* genannt [Ben86], bieten verschiedene Risiken und Möglichkeiten. So kann laut *Fowler* neben der Steigerung der Entwicklungsproduktivität unter anderem auch die Kommunikation mit den Experten der Domäne verbessert werden. Nachteilig sieht *Fowler* beispielsweise die falsche Verwendung der Sprache und den initialen Aufwand sowohl bei der Definition als auch bei der Implementierung. [FP11, S. 33 ff.]

Die Definition einer einheitlichen Sprache schärft nicht nur die Gedanken, sondern begünstigt auch die Reduktion von Wiederholungen [VC11]. DSL zeichnen sich unter anderem auch durch den begrenzten Satz von Vokabeln aus. Die Mächtigkeit einer DSL für eine Domäne steht konträr zu den Eigenschaften der Abstraktion und des Generischen der GPL. Die bekanntesten DSLs sind vermutlich Hypertext Markup Language (HTML) und die Structured Query Language (SQL) für Datenbanken [MHS05].

Um die Relevanz und Aktualität des Forschungsbereichs zu DSLs zu beurteilen, wird eine quantitative Analyse des bisherigen Forschungsstandes zu DSLs durchgeführt. Es wurden nur Publikationen der größten Herausgeber (IEEE, Springer, Elsevier/Sciencedirect und ACM) berücksichtigt. Bei der ersten Iteration wurden für die Volltextsuche folgende Begriffe zur DSL verwendet: '*domain-specific language(s)*', '*specification language(s)*', '*description language(s)*' und '*dsl(s)*'. In einer zweiten Iteration wurde die Volltextsuche zur DSL um die Logistik mit den Begriffen '*logistic(s)*' und '*intra-logistic(s)*' verfeinert. In einer weiteren Iteration wurde nochmals um den Begriff '*robotic*' verfeinert. Abb. 4.8 stellt eine quantitative Analyse zu den Themenbereichen für den Zeitraum von 2000–2022 dar.

Trotz der Vielzahl an bereits erfolgter Veröffentlichungen stellen DSLs immer noch ein relevantes Forschungsfeld dar. Dies gilt insbesondere für die Logistik in Verbindung mit der Robotik, aufgrund der steigenden Anzahl an Veröffentlichungen.

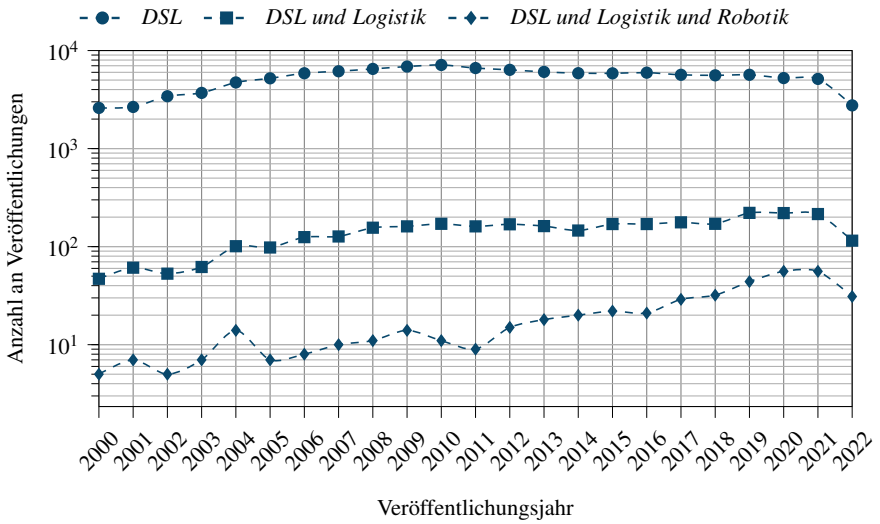


Abb. 4.8: Zeitliche Verteilung der Veröffentlichungen zu den Themen *'domain-specific language (DSL)„ 'DSL und Logistik,* und *'DSL und Logistik und Robotik,* im Zeitraum von 2000–2022 (Stand: Oktober 2022)

4.3.2 DSLs für die Logistik und Robotik

Nachfolgend werden existierende Arbeiten anhand der logistischen Anforderungen aus Abschn. 4.2.2 bewertet. Ziel dieser Evaluation ist es, den Schwerpunkt der bisherigen Arbeiten zu ermitteln und offene Fragen bei der Gestaltung von DSLs im Rahmen der High-Level-Aufgabenbeschreibung für FTF in der Intralogistik zu identifizieren.

DSLs werden als ein unverzichtbares Werkzeug für den Einsatz bei Roboterschwärmen gesehen [PB16]. Neben Roboterschwärmen haben DSLs auch Anwendung in anderen Bereichen wie der Automatisierung und Fertigung gefunden. So wurde für die Domäne *Schweißen* eine DSL entworfen und implementiert [MPP14]. In [NHW14, NHW+16][5] wurde eine annotierte Übersicht über die derzeit verfügbaren DSLs in der Robotik erstellt und diese nach den folgenden zwei Dimensionen klassifiziert:

- **Die funktionale Dimension**: Diese Dimension basiert auf den grundlegenden Prinzipien und Methoden, die für die Entwicklung eines Roboters oder eines Robotersystems notwendig sind [SK16]. Diese besteht aus den neun Teildomänen *Kinematik*, *Dynamik*, *Mechanik und Aktoren*, *Sensoren und Schätzung*, *Bewegungsplanung*, *Bewegungssteuerung*, *Kraftsteuerung*, *Architekturen und*

[5] Eine aktuelle Übersicht aller verfügbaren DSLs in der Robotik ist hier zu finden: https://corlab.github.io/dslzoo/index.html – letzter Zugriff: September 2022

Programmierung sowie *Wissensrepräsentation und Methoden* [SK16, Part A, Kapitel 1-4,7-9,14].

- **Die Entwicklungsprozess-Dimension**: Diese Dimension soll einem Entwickler bei der Integration eines Roboters bzw. eines Robotersystems helfen. Der holistische Entwicklungsprozess *Robot Application Development Process* (RAP) wurde in dem EU-Forschungsprojekt *Best Practice in Robotics* (BRICS)[6] entwickelt und ist in acht Phasen (*Scenario Building*, *Functional Design*, *Platform Building*, *Capability Building*, *System Deployment*, *System Benchmarking*, *Product Deployment* und *Product Maintenance*) unterteilt [KSP+10]. Dieser Prozess basiert auf bekannten Prinzipien des Software Engineerings, agiler Softwareentwicklung, modellbasiertem Engineering und System Engineering [NHW+16].

Diese Arbeit fokussiert sich auf die Verwendung eines (mobilen) Roboters (hier: FTF) bzw. eines Robotersystems, um einen selbstorganisierenden Materialfluss zu realisieren, und nicht darauf, einen eigenen Roboter zu entwickeln. Daher sind für diese Arbeit die *funktionale Dimension*, die Teildomäne *Architektur und Programmierung* und die Phasen *Scenario Building* und *Functional Design* der Dimension *Entwicklungsprozess* relevant. Die Teildomäne *Architektur und Programmierung* beschäftigt sich mit den High-Level-Spezifikationen von Robotersystemen auf Software-Ebene. Dabei kann zwischen der Struktur und dem Design differenziert werden, wie einzelne Teilnehmer miteinander interagieren oder kommunizieren. Die Phase *Scenario Building* umfasst die Definition von Merkmalen, Einschränkungen und Eigenschaften der Umgebung, welche für das Szenario (hier: der Materialfluss) relevant sind. Zusätzlich werden hier die Aufgaben des Roboters definiert. Die Phase *Functional Design* ist unter anderem notwendig, um die High-Level-Funktionalität eines Systems zu definieren. Diese High-Level-Funktionalitäten werden dann zerlegt und die Abhängigkeiten zwischen diesen identifiziert.

In Abb. 4.9 ist ein zeitlicher Verlauf der Veröffentlichungen zu DSLs in der Robotik seit dem Jahr 1984 dargestellt. Die zeitliche Verteilung berücksichtigt die Teildomäne *Architektur und Programmierung* der *funktionalen Dimension* nach *Siciliano et al.* und die Phasen *Scenario Building* und *Functional Design* nach dem *BRICS RAP* und einige Filterkategorien (fehlende Online-Verfügbarkeit, fehlendes Meta-Modell, keine Anwendung für die Domäne Robotik). Diese Übersicht bezieht sich auf die Art und Weise wie ein Robotersystem auf der Software-Ebene entworfen wird. Dies umfasst u. a., wie Aufgaben definiert werden oder eine Interaktion zwischen Robotern stattfindet. Die Tabelle 4.1 enthält eine weitere, verfeinerte Auswahl der wichtigsten existierenden DSLs im Vergleich. Die hier gelisteten DSLs sind chronologisch nach ihrer ersten Publikation sortiert. Die logistischen Anforderungen (LA) 1 bis 6 wurden in Abschn. 4.2.2 definiert. Weitere wesentliche Eigenschaften sind das Programmierparadigma (imperativ oder deklarativ) oder die Domäne, in der die DSL ihren Einsatz findet. Bei der imperativen Programmierung steht das *Wie* im Vordergrund, während bei der deklarativen Programmierung das *Was* von Bedeutung ist.

[6] Vgl. BRICS – Best Practice in Robotics: https://cordis.europa.eu/project/id/231940

Mit der *vTSL* wird eine DSL vorgeschlagen, bei der der Benutzer imperativ das Verhalten eines Roboters mithilfe von *TaskTrees* definiert [HL18]. Diese *TaskTrees* basieren auf *Actions* und *Behaviours*. Eine *Action* ist zum Beispiel das Finden einer Ladevorrichtung, während das *Behaviour* beschreibt, wie dies umgesetzt wird. Dabei werden die vom Skill-Layer bereitgestellten Fähigkeiten, beispielsweise durch ROS [QGC+], durch die im Behaviour beschriebenen Methoden orchestriert. Nach der Definition wird das *vTSL*-Modell für eine formale Verifikation in ein Promela-Modell [Hol03] umgewandelt. *vTSL* hilft dabei, ein autonomes Verhalten eines Fahrerlosen Transportfahrzeugs zu beschreiben, welches sicherstellt, dass eine Anweisung weder die Sicherheit noch die Hardware des Fahrzeugs gefährdet.

Die DSL *Dolphin* ermöglicht durch ihren imperativen Ansatz Aufgaben für eine bestimmte Anzahl an Vehikeln zu definieren [LMP+18]. Diese Aufgaben können konkurrierend, sequenziell oder eventbasiert abgearbeitet werden. Ein zentraler Orchestrator setzt hierbei die globale Spezifikation der Aufgaben um. Die Domäne dieser DSL ist allerdings die maritime Umgebung. *NVL* [MRP+15] und *Dolphin* [LMP+18] sind beide Turing-vollständige Sprachen. Der Benutzer muss allerdings bei beiden Sprachen Scheduling inkl. des Festlegens aller Parameter durchführen.

Der Ansatz von *Karma* umfasst eine Architektur zur Orchestration für Mikro-Aerial-Vehikles-Netzwerke [DKW+11]. Der zentrale Orchestrator (*Hive*) verteilt die Aufgaben an die Drohnen mit der Randbedingung, dass diese während der Ausführung einer Aktion untereinander kommunizieren können. Jedes Verhalten ist unabhängig definiert, mit entsprechenden Eigenschaften und Fortschrittsfunktionen.

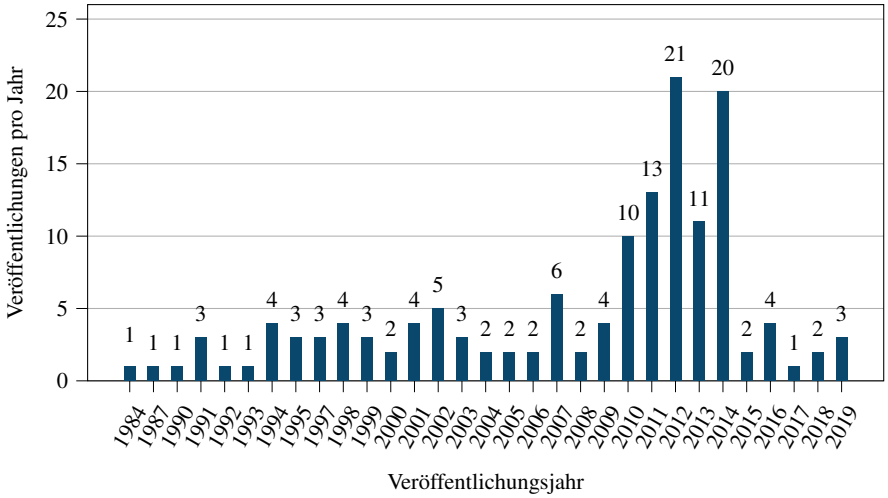


Abb. 4.9: Zeitliche Verteilung der Veröffentlichungen, welche sich mit DSLs in der Robotik im Zeitraum von 1984–2019 beschäftigt haben (Summe: 142). Die hier aufgeführten DSLs decken die Phasen *Scenario Building* und *Functional Design* nach [KSP+10] oder die Teildomäne *Architekturen und Programmierung* nach [SK16] ab.

Tabelle 4.1: Überblick über die wichtigsten domänenspezifischen Sprachen in der Robotik unter Berücksichtigung ihres Programmierparadigmas und der Erfüllung der logistischen Anforderungen LA 1 bis LA 6 (vgl. Abschn. 4.2.2)

Name	**Quelle**	**Jahr**	**Domäne**	**Programmier-paradigma**	**LA 1**	**LA 2**	**LA 3**	**LA 4**	**LA 5**	**LA 6**
TDL	[SA98]	1998	Robotik	Imperativ	✓	✓	✗	✓	✗	✓
Meld	[ALG+09]	2009	Robotik	Deklarativ/ Logik	✗	✗	✓	✓	✗	✓
Karma	[DKW+11]	2011	Robotik/ unbemannte Luftfahrt	Logik	✗	✗	✗	✓	✗	✓
NVL	[MRP+15]	2015	Robotik	Imperativ	✓	✓	✗	✓	✓	✓
BUZZ	[PB16]	2016	Robotik	Imperativ	✗	✗	✗	✓	✗	✓
vTSL	[HL18]	2018	Robotik	Imperativ	✓	✓	✗	✓	✓	✓
Dolphin	[LMP+18]	2018	Schifffahrt	Imperativ	✓	✓	✓	✓	✓	✓
CP^{3}L	[ASP+15]	2015	Produktion	Deklarativ	✓	✓	✗	✓	✓	✓
PROMISE	[GPM+19]	2019	Robotik	Deklarativ	✓	✓	✓	✓	✓	✓
RoboLang	[SM19]	2019	Robotik	Deklarativ	✓	✓	✗	✓	✗	✓

Gesammelte Daten werden an den *Hive* übermittelt, sobald die Mission abgeschlossen ist.

Auf die Beschreibung des Verhaltens für einen heterogenen Roboterschwarm fokussiert sich die DSL *Buzz* [PB16]. Hierbei kann das Verhalten von einem einzelnen Roboter oder einem ganzen Schwarm durch imperative Befehle beschrieben werden. Informationen werden durch Nachbarschafts-Anfragen in der unmittelbaren Nähe mittels Nachrichten ausgetauscht. Daher muss bei *Buzz* auch das Verhalten für eingehende Nachrichten definiert werden. *Buzz* unterstützt die Formation eines Schwarms durch eine *virtuelle Stigmergie*, die indirekte Koordinierung durch die Beeinflussung der Umgebung, durch die Einigung auf ein Set von Variablen aufgrund der nachrichtenbasierten Kommunikation. Die Heterogenität ist gegeben durch eine Erweiterungssprache, welche oberhalb von ROS ausgeführt wird. Um *Buzz* direkt in ROS auszuführen, kann die ROS-basierte Erweiterung *ROSBuzz* verwendet werden [SVL+17].

Die deklarative DSL *PROMISE* ermöglicht sowohl eine textuelle als auch eine graphische Programmierung [GPM+19]. Hierbei wird das Ziel einer Mission beschrieben, jedoch nicht die einzelnen Aktionen, die zur Durchführung der Mission benötigt werden. Die globale Mission wird dann in eine Zwischensprache (*Inter-*

mediate Language) überführt, für eine Entkopplung des Ziels von der auszuführenden Roboterplattform. Diese Zwischensprache, welche auf Linearer Temporaler Logik (LTL) basiert, muss von einem Interpreter in Aktionen für die jeweilige Roboter-Plattform umgesetzt werden.

Meld, eine deklarative DSL, folgt einem Top-down Ansatz [ALG+09, S. 265 ff.]. Der Benutzer führt eine High-Level-Logische Beschreibung durch, was der Schwarm als Ganzes erreichen/erzielen/erledigen soll. Die Kommunikation zwischen den einzelnen Schwarmteilnehmern liegt in der Verantwortung des Entwicklers. *Meld* basiert auf Fakten (*Facts*) und elf Regeln (*Rules*). Ein Fakt ist eine Information, welche das System zu einem bestimmten Zeitpunkt und in Abhängigkeit von der Roboterfähigkeit als *wahr* ansehen kann. Anhand der Regeln werden Fakten erstellt. Der *Meld*-Compiler generiert im Anschluss Code für leichtgewichtige, eingebettete Betriebssysteme wie TinyOS[7].

In der Intralogistik und somit in CPPS ist die Auslastung an FTF zur Laufzeit nicht bekannt. Wird zur Definition der Aufgabe bereits formuliert, wie viele Fahrzeuge benötigt werden und welche Fahrzeuge diese Aufgabe ausführen sollen, kann dies eine Einschränkung sein. So besteht die Möglichkeit, dass ein anderes Fahrzeug eine bessere Option zur Ausführung der Aufgabe wäre als es zu einem vorherigen Zeitpunkt festgelegt wurde. Eine DSL ist das Bindeglied zwischen einem ERP und einem FTF oder einer FTF-Leitsteuerung.

Die zuvor vorgestellten Lösungen unterscheiden sich in ihrem Programmierparadigma, ihrem Einsatzgebiet und der Abdeckung der logistischen Anforderungen. Insgesamt kann festgehalten werden, dass sich unter den recherchierten Lösungen kein Ansatz befindet, welcher die Domäne Logistik und den damit verbundenen intralogistischen Anforderungen an einen dezentral organisierten Transportprozess berücksichtigt. Ansätze wie *Dolphin* oder *Karma* setzen einen zentralen Koordinator (Orchestrator) voraus, welcher vergleichbar mit einer zentralen Leitsteuerung ist. Zusätzlich verfolgen die meisten einen *Bottom-up*-Ansatz, bei dem mithilfe von bestimmten primitiven Grundoperationen komplexere Funktionen realisiert werden. Dies ist auch bei der *vTSL* der Fall, bei der der Benutzer imperativ die Aktionen aus Sicht des Roboters beschreiben muss. In dem Lösungsansatz *Promise* wird die DSL in die Zwischensprache Lineare Temporale Logik (LTL) konvertiert. LTLs können in einer nichtdeterministisch polynomiellen Komplexitätsklasse (NP-Schwere) [BSS+07] liegen und auch anfällig für Fehler während der Definition sein, auch für Experten [AGL+15, Hol02]. In der Forschung findet LTL eine weite Verbreitung, jedoch aufgrund der mangelnden Benutzerfreundlichkeit kaum Anwendung im industriellen Umfeld. Darüber hinaus besteht bei LTL die Herausforderung, eine eindeutige Beschreibung eines Problems zu schaffen.

In einem CPPS, bei dem ein Transport ein eigenständiger Teilnehmer ist, hat jeder Teilnehmer nur eine auf sich selbst beschränkte lokale Sicht. Daher kann eine Konvertierung in einen Petri-Netz-Plan oder einen Graphen ausreichen, wie gezeigt in [DKJ19].

---

[7] TinyOS: https://github.com/tinyos/tinyos-main

4.3.3 Formale Grammatik der MFDL

Die Basis der *Material Flow Description Language (MFDL)* ist die abgebildete Grammatik in der nachfolgenden Abb. 4.10. Zur Vereinfachung wird eine vereinfachte Grammatik verwendet, in Anlehnung an [Hos15]. Ein senkrechter Strich (|) stellt einen optionalen Gebrauch vor; ein Überstrich impliziert eine Wiederholung die keinmal, einmal oder mehrmals vorkommen kann. Im Anhang C.1 befindet sich die vollständige ANTLR-Grammatik [Par13] der MFDL.

$p \in$ Program ::= $\overline{\textbf{Import}\ m_{id}}\ \overline{s \mid i \mid r \mid m \mid t \mid tos \mid mos \mid aos}$ PROGRAMM

$i \in$ Instance ::= $s_{id}\ i_{id}\ \overline{aa}^{+}$ **End** INSTANZEN

$r \in$ Rule ::= **Rule** $rc\ \overline{expr}^{+}$ **End** REGELN

$rc \in$ RuleCall ::= r_{id} **(** $(rp\ (\overline{\textbf{,}\ rp}))$ **)**

$rp \in$ RuleParameter ::= p (**=** v)

$m \in$ Module ::= **Module** $m_{id}\ \overline{s \mid i \mid r \mid t \mid tos}$ **End** MODULE

$t \in$ Task ::= **Task** $t_{id}\ \overline{ts}^{+}$ **End** AUFTRÄGE

$ts \in$ TaskStatement ::= $os \mid ops \mid rs \mid cs$

$os \in$ OrderStatement ::= $tp \mid mv \mid ac$

$rs \in$ RepeatStatement ::= **Repeat :** *Integer*

$cs \in$ ConstraintStatement ::= **Constraint :** *expr*

$tp \in$ Transport ::= **Transport** TRANSPORTE
From tos_{id}
To tos_{id}

$mv \in$ Move ::= **Move To** mos_{id} BEWEGUNGEN

$ac \in$ Action ::= **Action Do** aos_{id} AKTIONEN

$tos \in$ TransportOrderStep ::= **TransportOrderStep** $\text{tos}_{id}\ \overline{tos\_s}^{+}$ **End** TRANSPORTSCHRITTE

$tos\_s \in$ TosStatement ::= **Location :** $i_{id} \mid$ **Parameters :** $v \mid ops$

$ops \in$ OptionalStatement ::= **StartedBy :** $expr \mid$ **FinishedBy :** $expr$
$\mid$ **OnDone :** $\overline{t_{id}}$ **,** t_{id}

$mos \in$ MoveOrderStep ::= **MoveOrderStep** $mos_{id}\ \overline{tos\_s}^{+}$ **End**

$aos \in$ ActionOrderStep ::= **ActionOrderStep** aos_{id} **End**

$aos_s \in$ AosStatement ::= **Parameters :** $v \mid ops$

$ad \in$ Attribute Def. ::= a_{id} **:** dt ATTRIBUTDEFINITIONEN

$aa \in$ Attribute Ass. ::= $a_{id}\ \overline{\textbf{.}\ a_{id}}$ **:** v ATTRIBUTZUWEISUNGEN

$expr \in$ Expression ::= ($expr$) $\mid$! $expr$ BOOL. AUSDRÜCKE
$\mid expr\ bo\ expr \mid v \mid rc$

$bo \in$ Binary Operation ::= $< \mid \leq \mid > \mid \geq \mid + \mid - \mid * \mid /$ BIN. OPERATOREN
$\mid == \mid != \mid$ **And** $\mid$ **Or**

$dt \in$ Data Type ::= **number** $\mid$ **string** $\mid$ **boolean** $\mid s_{id}$ DATENTYPEN

$v \in$ Value ::= **true** $\mid$ **false** $\mid$ *Number* $\mid$ *String* $\mid$ *id* $\mid$ *jo* WERTE

$p \in$ Parameter ::= $p_{id} \mid v$ PARAMETER

$s_{id}, i_{id}, r_{id}, m_{id}, t_{id} \in$ *Identifizierer der Primitiven*

$tos_{id}, mos_{id}, aos_{id} \in$ *Identifizierer der Orderschritte*

$v_{id}, a_{id} \in$ *Variable- oder AttributeName*

Abb. 4.10: Formale Grammatik der Material Flow Description Language (MFDL)

4.4 Interpretation der MFDL

In diesem Abschnitt folgt die Vorstellung der *Material Flow Description Language (MFDL)* anhand von verschiedenen Beispielen und den logistischen Anforderungen (vgl. Abschn. 4.2.2). Mithilfe von *MFDL* wird ein Transport deklarativ definiert. Dabei wird nicht definiert, welches FTF den Transport durchführt, sondern lediglich, wie ein Transport von der Quelle zur Senke aussieht. Ein *MFDL*-Programm besteht aus Modulen (*Module*), Strukturen (*Structs*), Regeln (*Rule*) und Aufgaben (*Task*). Module und Strukturen helfen, den Programmcode zu strukturieren, während die Aufgaben und Regeln auch Berechnungen bzw. Operationen erlauben. Alle zuvor definierten Regeln und Strukturen können dann in den logistischen Teilschritten, dem Transportauftrag (*TransportOrder*), dem Transportschritt (*TransportOrderStep*), einer Aktion (*ActionOrder*) oder einer Bewegung (*MoveOrder*) angewendet werden. Mithilfe dieser drei Teilschritte können dann Transportaufgaben in den Aufgaben (*Tasks*) definiert werden. Innerhalb der *Tasks* können Transporte sequenziell, parallel oder ereignisgesteuert beschrieben werden.

4.4.1 Strukturen und Module

In jedem Programm werden Informationen verarbeitet. In objektorientierten Programmiersprachen wie C++, C-Sharp oder Java werden diese innerhalb von Objekten gespeichert. *MFDL* verwendet hierfür Strukturen mit Attribut-Werte-Paaren. In Auflistung 4.1 ist dargestellt, wie eine Struktur innerhalb der *MFDL* definiert wird. Eine Struktur beinhaltet einen Attributnamen gefolgt von einem Datentyp. Schlüsselwörter beginnen mit einem Großbuchstaben, während elementare Datentypen mit einem Kleinbuchstaben beginnen. Folgende elementare Datentypen werden unterstützt: Zahlen (*number*), Zeichenketten (*string*), logische Werte (*boolean*) oder Strukturen (*structs*).

Auflistung 4.1: Definition einer Struktur in *MFDL*.

```
1 Struct
2   id : string   # eindeutige Identifikation der Struktur
3   time : number # Zeitstempel der letzten Aenderung eines Attributes
4 End
```

Eine Struktur besitzt immer zwei Attribute, welche nicht durch den Benutzer definiert werden müssen: Bei dem Attribut *id* handelt es sich um eine eindeutige Kennung. Wird diese nicht gesetzt, so wird ein Wert für die ID verwendet, der als global eindeutig angenommen werden kann. Das Attribut *time* ist per Definition immer gesetzt und muss in keiner Struktur definiert werden. Der Wert für das Attribut *time* kann entweder durch eine Eingabequelle (bspw. ein Event) gesetzt werden oder automatisch durch den *MFDL*-Interpreter. Alle entworfenen Strukturen besitzen diese beiden Attribute per Definition. Zusätzlich sind innerhalb von *MFDL* die folgenden Basisstrukturen (vgl. Auflistung 4.2) mit entsprechenden Attributen definiert. Die Attribute *id* und *time* aus der Auflistung 4.1 werden hier nicht erneut aufgeführt.

Auflistung 4.2: Beispiel der vorhandenen Basisstrukturen in der *MFDL*.

```
1 Module Intralogistic
2   Struct Location
3     type : string # Repraesentiert den Ladungstraeger
4   End
5   Struct Event
6     value : {number|string|array|object|boolean}
7   End
8   Struct Time
9     timing : string # Unix crontab Format
10  End
11  Struct Constraint
12    type : {number|string|array|object|boolean}
13  End
14 End
```

Die vier Strukturen *Location*, *Event*, *Time* und *Constraint* sind fester Bestandteil in *MFDL* und müssen nicht jedes mal definiert werden.

- ***Location***: Eine *Location* (Z. 2–4) beschreibt einen Ort innerhalb eines CPPS. Der Ort selbst wird im Attribut *id* beschrieben. Dies kann bspw. eine logische, eindeutige ID sein. Das Attribut *type* steht für den Typen des Ladehilfmittels, der dem Ort zugeordnet ist.
- ***Event***: Für eine ereignisgesteuerte Transportkette kann die Basisstruktur (*Event*) (Z. 5–7) verwendet. So kann bspw. ein manuelles Beladen durch den Menschen durch das Drücken eines Sensors bestätigt werden. Ein Event besteht aus einer eindeutigen Identifikation (*id*) und dem Wert (*value*) des Ereignisses. Die *id* kann ein logischer Name des Sensors sein.
- ***Time***: Der Struktur *Time* (Z. 8–10) wird als Zeichenkette im Format eines Unix *crontab*[8] verwendet. Somit kann ein Transport zu einer bestimmten Uhrzeit ausgeführt werden.
- ***Constraint***: Ein *Constraint* (Z. 11–13) beschreibt Randbedingungen, Einschränkungen oder sonstige Anforderungen.

Diese Basisstrukturen sind eng mit der Definition eines Transportauftrags verknüpft (vgl. Abschn. 4.2.1): Ein Transportauftrag ist eine Bewegung eines Guts (in einem Behälter) von einem Ort (hier: Quelle) zu einem anderen Ort (hier: Senke).

Ein Modul (*Module*) wird verwendet, um eine Funktionalität zu kapseln bzw. um diese zu gruppieren. Hierdurch kann eine Wiederverwendbarkeit geschaffen werden. Module sind vergleichbar mit Strukturen, können allerdings nicht instanziiert werden. Daher werden Module initial zu Beginn eines Programms global mit dem Schlüsselwort *Import* eingefügt (vgl. Auflistung 4.3).

Auflistung 4.3: Importierung eines externen Moduls in der *MFDL*

```
1 Import myModule
2 ...
```

[8] crontab(5) - Linux man page: https://linux.die.net/man/5/crontab

Komposition und Vererbung

MFDL unterstützt zwei Arten bei der Definition von neuen, komplexeren Strukturen: Hierbei handelt es sich um eine Komposition von Strukturen oder die Vererbung von Attributen und Eigenschaften. Diese Mechanismen sind auf alle Strukturen innerhalb von *MFDL* anwendbar. Zur beispielhaften Darstellung werden diese Mechanismen nun auf die Basisstruktur *Location* angewendet (vgl. Auflistung 4.4 bzw. vgl. Auflistung 4.5).

Auflistung 4.4: Komposition von Strukturen innerhalb von *MFDL*.

```
1 Struct MyStructComposition
2   weight : string
3   myLocation : Location
4 End
```

Somit beinhaltet die Datenstruktur *MyStructComposition* ein Attribut *weight* (Z. 2) und ein Attribut *myLocation* (Z. 3) von der Basisstruktur *Location*. Bei der Vererbung von Strukturen werden alle Attribute in die neue Struktur übernommen. Die Struktur *MyLocationInheritence* besitzt ein Attribut *weight*, neben den beiden Attributen *id* und *type* der Basisstruktur *Location*.

Auflistung 4.5: Vererbung von Attributen existierender Strukturen.

```
1 Struct MyLocationInheritence : Location
2     weight : string
3 End
```

Alternativ zur Vererbung bei der Struktur *MyLocationInheritence* kann auch eine Struktur mit den entsprechenden Parametern definiert werden.

Instanziierung von Strukturen

Bei der Instanziierung von Strukturen werden den zuvor definierten Attributen konkrete Ausprägungen bzw. Werte zugewiesen. Jede Instanz hat lediglich die innerhalb der Struktur definierten Attribute. Nachfolgend werden die verschiedenen Basisstrukturen aus dem vorherigen Abschnitt vorgestellt.

Location

Die folgende Auflistung 4.6 setzt die Strukturen aus den Auflistungen 4.4 und 4.5 voraus. Die *Location* in Z. 1–4 initialisiert eine Basisstruktur. In Z. 5–8 wird die erweiterte *Location* aus Auflistung 4.4 initialisiert. Strukturen können entweder Attribut für Attribut initialisiert werden (Z. 5–9) oder über eine *JSON*-angelehnte *inline*-Initialisierung (Z. 13).

Auflistung 4.6: Instanziierung der Basisstruktur *Location*.

```
1  Location abholung
2    id : "0x4711"
3    type : "pallet"
4  End
5  MyStructComposition einWeitererOrt
6    weight : "50 kg"
7    myLocation.id : "0x4711"
8    myLocation.type :"pallet"
9  End
10 # alternativ kann eine JSON Initialisierung einer Struktur erfolgen
11 MyStructComposition einWeitererOrt
12   weight : "50 kg"
13   myLocation : {"id" : "0x4711", "type":"pallet"}
14 End
15 # t.b.d.
16 MyLocationInheritence lieferungOrt
17   weight : "50 kg"
18   id : "0x4242"
19   type : "pallet"
20 End
```

Event

In der Auflistung 4.7 ist ein Beispiel dargestellt, in dem ein *Event* initiiert wird.

Auflistung 4.7: Instanziierung der Basisstruktur *Event*.

```
1 Event manuelleBestaetigungFTFistBeladen
2     id : "ButtonFtfBeladen"
3     value : False
4 End
```

In Zeile 2 wird dem Event ein logischer Name zugewiesen, welcher das Event auslöst, während in Zeile 3 zunächst der initiale Wert zugewiesen wird. Das Attribut *time* wird entweder explizit gesetzt oder automatisch bei einer internen Änderung aktualisiert.

Time

In der Auflistung 4.8 wird ein temporaler Aspekt dargestellt. Der Aufbau der Basisstruktur orientiert sich an dem Unix *crontab*.

Auflistung 4.8: Aufbau der Basisstruktur *Time* basierend auf dem Unix *crontab*.

```
1 # ---------- Minute (0 - 59)
2 # | -------- Stunde (0 - 23)
3 # | | ------ Tag des Monats (1 - 31)
4 # | | | ---- Monat (1 - 12)
5 # | | | | -- Tag der Woche (0 - 6) (Sonntag bis Samstag;
6 # | | | | |   7 ist in manchen Systemen der Sonntag)
7 # | | | | |
8 # | | | | |
9 # * * * * *
```

In der nachfolgenden Auflistung 4.9 wird dieser auf einen kommenden Tag um 08:30 Uhr morgens gestellt.

Auflistung 4.9: Definition eines zeitgesteuerten Events für den kommenden Tag um 08:30 Uhr morgens.

```
1 Time morningBreak
2     timing : "30 08 * * *"  # Crontab Format
3 End
```

Constraint

Einschränkungen (*Constraints*) können z. B. in der Terminplanung oder bei der Aushandlung eines FTFs verwendet werden. Diese Einschränkungen können folglich in die Optimierung mit einfließen. Beispiele für Einschränkungen sind die Transportkosten oder eine Zeitspanne, in der ein Transport durchgeführt werden soll (vgl. Auflistung 4.10).

Auflistung 4.10: Definition einer Einschränkung bzw. einer Randbedingung.

```
1 Struct Constraint
2   type : string
3 End
4 Constraint transportStart
5   value : "30 08 * * *"
6 End
```

4.4.2 Regeln

Mithilfe einer Regel (*Rule*) können die Werte von Strukturen evaluiert und eine ereignisgesteuerte Programmsteuerung zu realisiert werden. Innerhalb von Regeln können die logischen Operatoren für Vergleiche ($< \mid \leq \mid == \mid != \mid \geq \mid >$) und Verknüpfungen ($AND|OR|NOT$) sowie mathematische Grundrechenarten ($+, -, *, /$) verwendet werden. Abhängigkeiten, Reihenfolgen oder auch temporale Eigenschaften können durch die Verwendung von Regeln abgebildet werden. Innerhalb von Regeln werden die Ausdrücke lediglich evaluiert. Eine Manipulation der Attribute erfolgt nicht.

Wenn ein Ausdruck innerhalb einer Regel eintrifft, so wird der Wert *True* zurückgegeben, andernfalls ist diese Regel ungültig (*False*). Darüber hinaus können in einer Regel auch andere Regeln aufgerufen werden. In der Auflistung 4.11 wird die zeitliche Reihenfolge von Events basierend auf dem Standardattribute *time* in der Regel *before* ausgewertet.

Auflistung 4.11: Eine zeitliche Regel basierend auf dem Attribut *time*.

```
1 Rule before(event1, event2, offset = 0)
2   (event1.time + offset < event2.time) And (event1.value == 5)
3 End
```

Einer Regel können Parameter übergeben werden oder Default-Werte. Dieses Beispiel hat drei Parameter, *event1*, *event2* und *offset*. Die ersten beiden Parameter wer-

den zur Laufzeit übergeben und der dritte, sofern dieser keinen Wert erhält, wird mit einem Default-Wert belegt.

In Tabelle 4.2 sind einige temporale Operatoren abgebildet [All83], die in *MFDL* als Regeln verwendet werden können. Diese Regeln sind auf die Basisstrukturen bzw. auf alle Strukturen anwendbar. Es besteht auch die Möglichkeit, eine Vielzahl

Tabelle 4.2: Temporale Operationen auf Strukturen

Operator	**Definition**	**Regel in *MFDL***
Equal	$struct1.time = struct2.time$	equalT(event1, event2)
Before	$struct1.time < struct2.time$	beforeT(event1, event2)
After	$struct1.time > struct2.time$	afterT(event1, event2)
Contains	$struct2.time < struct1.time$	containsT(event1, event2)

an Regeln in einer Regel abzuarbeiten. Stehen mehrere Regeln in den Zeilen untereinander, so wird implizit angenommen, dass alle Ausdrücke der Regel über die Zeilen hinweg mit einem *and* verknüpft sind (vgl. Z. 2–4 bzw. Z. 8 aus Auflistung 4.12).

Auflistung 4.12: Verkettung von Regeln basierend auf dem Attribut *time*.

```
1 Rule orderOfEvents(event1, event2)
2     before(event1, event2, 50) # implizites und
3     before(event2, event3)  # implizites und
4     event2.value ==  100
5 End
6 # gleichwertig  zu
7 Rule orderOfEvents
8     before(event1, event2, 50) And before(event2, event3) And event2.value
       ==  100
9 End
```

In Z. 2 bzw. Z. 3 wird die zuvor erstellte Regel *before* aus Auflistung 4.11 verwendet. Sollten die Regeln aus Z. 2 bzw. Z. 3 mit einer Oder-Verknüpfung ausgewertet werden, so hätte am Ende der Zeile entweder ein *or* stehen müssen oder das *and* aus Z. 8 durch ein *or* ersetzt werden müssen.

4.4.3 Tasks

Die dezentrale Entscheidungsfindung ist in der Industrie 4.0 eine wichtige Eigenschaft [HPO16]. Ein Transportauftrag kann dabei eine eigene Entität innerhalb eines CPPS sein, welche versucht sich zur Laufzeit eigenständig fertigzustellen. Dabei sucht sich der Transportauftrag die entsprechende Entität selbst aus, welche

den Transport durchführen soll. Hierfür existieren verschiedene Optionen: Entweder kennt der Transportauftrag die Zustände aller relevanten Entitäten. Basierend auf diesem Wissen kann der Transportauftrag einem FTF den Auftrag zuweisen, welches die Kriterien für den Transportauftrag am besten erfüllt. Oder es besteht die Option, dass der Transportauftrag ausgeschrieben wird und die entsprechenden Entitäten sich um den Auftrag bewerben [BLS+17]. Ein klassisches Bieter-Verfahren, welches sich hier anbietet ist *Contract-Net* [Smi80]. Diese dezentrale Entscheidungsfindung kann allerdings zu lokalen Optima führen [DRVD20]. Da jeder Transportauftrag für sich selbst verantwortlich ist, versucht dieser auch, aus seiner Sicht das Optimum zu erhalten. Unter Umständen wird ein Transportauftrag nie durchgeführt, z. B. wenn andere Transportaufträge höher priorisiert werden und der Transportauftrag ohne Priorität sich immer am Ende der Auftragswarteschlange befindet.

Basierend auf der formalen Definition eines Transportauftrags aus Abschnitt 4.2.1 wird in der Auflistung 4.13 eine Aufgabe (*Task*) definiert (Z. 2–4).

Auflistung 4.13: Definition und Aufbau eines Transports mit notwendigen und optionalen Attributen.

```
1 Task {name}
2   Transport
3   From {transportOrderStep_pickup1, ...} # Abholung von Ort 1, ...
4   To {transportOrderStep_destination} # Lieferung an Ort
5
6    # optional:
7    # Event-getrieben Flusskontrolle
8   StartedBy : Rule({Instance_Event|Instance_Time})
9   FinishedBy : Rule({Instance_Event|Instance_Time})
10
11  OnDone : {none|Task} # Anschlussaufgabe
12
13  Repeat : {none = once|1, ..., n|0 = forever}
14  Constraints : Rule({Instance_Constraint}) # Randbedingung
15 End
```

Für einen einfachen Transport wird ein *TransportOrderStep* (*TOS*) benötigt, welcher die Abholung explizit beschreibt. Eine genauere Beschreibung, wie ein *TOS* aussieht, ist in der Auflistung 4.16 dargestellt. Bei einem *1:1*-Transport, bei dem etwas von einer Quelle zu einer Senke transportiert wird, muss lediglich ein *TOS* angegeben werden. Bei einem *N:1*-Transport, bei dem etwas von verschiedenen Positionen abgeholt wird, um es an einen Zielort zu transportieren, können auch mehrere *TOSs* angegeben werden. So kann eine Pickliste bzw. eine Kommissionierliste [HSD18, S. 287] realisiert werden. Die Reihenfolge der *Pickups* (Tour), in welcher die Waren oder Behälter abgeholt werden, kann ebenfalls optimiert werden. Dies ist ein klassisches Beispiel für das *Travelling Salesman Problem* [LM90]. Allerdings wird hier noch die Komplexität erhöht: Zum einen ist der Platz innerhalb einer Fabrik begrenzt, in dem sich FTF bewegen können. Zum anderen kann es mit anderen aktiven Transportaufträgen – die sich in der Optimierung befinden – zu einer Ressourcenallokation von FTF oder Überschneidung von Fahrtwegen kommen.

Mithilfe der optionalen Schlüsselwörter *StartedBy* (Z. 8) und *FinishedBy* (Z. 9) kann eine Ereignissteuerung erfolgen. Zur Auswertung einer *Event*-Struktur (vgl. Abschn. 4.4.1) können Regeln (vgl. Abschn. 4.4.2) verwendet werden. *StartedBy*

beschreibt eine Vorbedingung, wann genau der Transport (Z. 2–4) gestartet werden soll. Konträr beschreibt *FinishedBy* eine nachgelagerte Bedingung, wann der Transport abgeschlossen ist. Eine Verkettung von Transporten wird mit dem Schlüsselwort *OnDone* (Z. 13) beschrieben, durch das Setzen einer bzw. mehrerer *Tasks*. Mithilfe der *Constraints* können Randbedingungen definiert werden, wie z. B. die Kosten für einen Transport, die maximale Transportzeit oder bis wann ein Transport abgeschlossen sein soll. In der Abb. 4.11 ist der Aufbau der hier beschriebenen *Task* dargestellt. Die gestrichelten Linien repräsentieren optionale Eigenschaften.

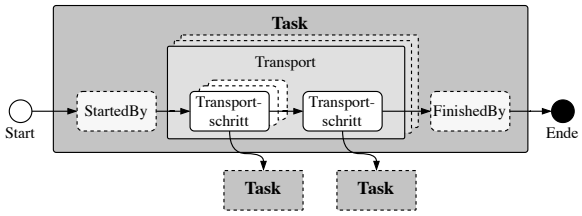


Abb. 4.11: Die Struktur einer *Task* in *MFDL*. Gestrichelte Linien indizieren optionale Eigenschaften.

Transporte innerhalb einer *Task* werden sequenziell abgearbeitet. In der Auflistung 4.14 wird zuerst der Transport aus Z. 2–4 abgearbeitet, bevor der Transport aus Z. 6–8 durchgeführt wird.

Auflistung 4.14: Sequenzielle Abarbeitung von Transporten innerhalb einer *Task*.

```
1 Task {name}
2   Transport
3   From {transportOrderStep_pickup1, ...} # Abholung von Ort 1, ...
4   To {transportOrderStep_destination1} # Lieferung an Ort
5
6   Transport
7   From {transportOrderStep_pickup2, ...} # Abholung von Ort 1, ...
8   To {transportOrderStep_destination2} # Lieferung an Ort
9 End
```

Sollen Transporte parallel ausgeführt werden, so können mehrere Tasks parallel definiert sein (vgl. Auflistung 4.15).

Auflistung 4.15: Parallele Abarbeitung von Transporten.

```
1 Task {name1}
2   Transport
3   From {transportOrderStep_pickup1, ...} # Abholung von Ort 1, ...
4   To {transportOrderStep_destination1} # Lieferung an Ort
5 End
6 Task {name2}
7   Transport
8   From {transportOrderStep_pickup2, ...} # Abholung von Ort 1, ...
9   To {transportOrderStep_destination2} # Lieferung an Ort
10 End
```

Welches FTF den Transport ausführt, wird erst zur Ausführungszeit bzw. zur Laufzeit entschieden. Hierbei kann es dazu kommen, dass der zweite Transport vor dem

ersten Transport ausgeführt werden kann. Dies hängt von der Anzahl der vorhandenen Fahrzeuge und der Zuweisungsstrategie ab.

Ein Transport ist eine Komposition aus zwei Transportschritten, den TransportOrderStep (TOS) (vgl. Abschn. 4.2.1). Ein TOS ist beschrieben durch einen Ort und die Aktion, welche an diesem ausgeführt werden soll (vgl. Auflistung 4.16). Dies kann entweder eine Beladung an einer Quelle oder die Entladung an einer Senke sein. Teilschritte werden innerhalb einer Aufgabe (*Task*) definiert.

Auflistung 4.16: Ein Transportschritt (*TransportOrderStep*) im Detail mit seinen notwendigen und optionalen Attributen.

```
1  TransportOrderStep {name}
2    # required
3   Location : {Instance_Location}
4    # optional
5   Parameters : {Instance_Parameter}
6    # event flow control
7   StartedBy : Rule(Instance_Event, ...)
8   FinishedBy : Rule(Instance_Event, ...)
9
10  OnDone : {none|TOS|Task} # follow up tasks
11  Constraints : Rule(Instance_Constraint, ...)
12 End
```

In Z. 3 wird die *Location* für die Abholung definiert. Mit der Definition *Parameter* kann eine Aktion an dem jeweiligen Ort (Z. 3) beschrieben werden, um etwas zu be- oder entladen. Die weiteren optionalen Parameter aus Z. 7–11 sind äquivalent zu denen einer *Task*. Für die zeitliche Flusskontrolle (wann dieser Teilauftrag beginnt bzw. abgeschlossen ist) werden mithilfe der Schlüsselwörter *StartedBy* bzw. *FinishedBy* beschrieben. Ob eine weitere Aufgabe im Anschluss durchgeführt werden soll, wird mithilfe von (*OnDone*) beschrieben. Die Randbedingungen bzw. Kosten zur Durchführung dieses *TOS* sind durch (*Constraints*) beschrieben.

Basierend auf dieser formalen Definition werden für einen Transport die Schritte bzw. Teilaufträge *MoveOrderStep* und *ActionOrderStep* benötigt (vgl. 4.2.1). Diese werden analog zum *TransportOrderStep* definiert.

4.5 Anwendung, Eigenschaften und Fähigkeiten von MFDL

In diesem Abschnitt werden einige Beispiele zur Anwendung von *MFDL* vorgestellt. Darüberhinaus werden die logistischen Anforderungen (vgl. Abschn. 4.2.2) berücksichtigt. Voraussetzung für die kommenden Beispiele ist der Aufbau einer Fabrikhalle aus Abb. 4.12. Es existieren zwei Bereiche (*Produktion* und *Warenlager*). Innerhalb dieser existieren jeweils zwei weitere Positionen (*prod1*, *prod2*, *lager1* und *lager2*). Zur Vereinfachung wird davon ausgegangen, dass innerhalb der Fabrik lediglich Paletten transportiert werden. Basierend auf diesen Positionen sind in der folgenden Auflistung 4.17 die entsprechenden *Locations* und *TransportOrderSteps* definiert.

Auflistung 4.17: Definitionen der Positionen und der dazugehörigen Transportteilschritte für Abb. 4.12.

```
1  # produktion
2 Location produktion1
3   id : "prod1"
4   type : "pallet"
5 End
6 Location produktion2
7   id : "prod2"
8   type : "pallet"
9 End
10  # warenlager
11 Location warenlager1
12   id : "lager1"
13   type : "pallet"
14 End
15 Location warenlager2
16   id : "lager2"
17   type : "pallet"
18 End
19
20 TransportOrderStep tos_warenlager1
21   Location : warenlager1
22 End
23
24 TransportOrderStep tos_warenlager2
25   Location : warenlager2
26 End
27
28 TransportOrderStep tos_produktion1
29   Location : produktion1
30 End
31
32 TransportOrderStep tos_produktion2
33   Location : produktion2
34 End
```

Die Unabhängigkeit des Fördermittels (LA1) ist durch die Tatsache gegeben, dass kein FTF angegeben wird. Dies erfolgt erst bei der Zuweisung des Transports. Die Unabhängigkeit des Ladehilfsmittels (LA2) ist durch die Flexibilität der *Locations* gegeben. Hier kann der Typ des Ladehilfsmittels frei definiert werden.

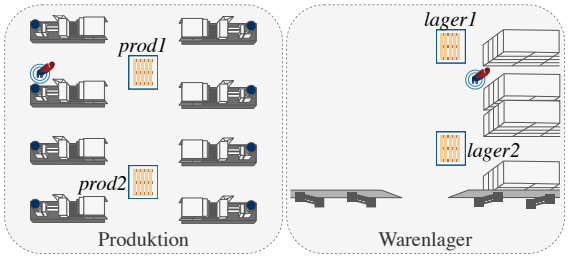


Abb. 4.12: Eine Fabrikhalle, aufgeteilt in zwei Bereiche (*Produktion* und *Warenlager*). Innerhalb dieser Bereiche gibt es vier Positionen (*prod1*, *prod2*, *lager1* und *lager2*), zwischen denen ein FTF Paletten transportiert.

Hello World Transport

Ein einfacher Transport, ohne Berücksichtigung eines Beladungs- oder Entladungsvorgangs, ist in der folgenden Auflistung 4.18 dargestellt. Dabei wird eine Palette vom Ort *lager1* aus dem Lagerbereich in den Produktionsbereich zum Ort *prod1* transportiert.

Auflistung 4.18: Ein einfacher Transport von der Position *Warenlager Abholung1* in die *Produktion* zur *Lieferung1*.

```
1 Task hello_world_transport
2   Transport
3   From tos_warenlager1
4   To tos_produktion1
5 End
```

Dies ist die einfachste Art, eine *Task* und somit einen Transport zu definieren.

Manuelle Beladung und automatische Entladung

Es folgt nun ein komplexeres Beispiel (vgl. Auflistung 4.19), bei dem an der Quelle eine manuelle Beladung erfolgt und das Fahrzeug an der Senke automatisch entlädt. Dies entspricht der logistischen Anforderung LA5. Für eine manuelle Beladung wird ein Sensor in Form eines Tasters benötigt, um die Beladung zu bestätigen. Ein Taster hat einen Default-Zustand, in den dieser übergeht, wenn dieser nicht gedrückt wird. Dieser ist in Z. 3 als *False* initialisiert. Ist das Fahrzeug am Beladungsort beladen worden, so kann durch das Drücken des Tasters *lager_agv_loaded* die Beladung bestätigt werden. Die Auswertung geschieht in Z. 8 mithilfe des *FinishedBy*-Ausdrucks. Alternativ ist auch hier die Anwendung einer *Rule* (vgl. Abschn. 4.4.2) denkbar.

Auflistung 4.19: Manueller Beladung und automatische Entladung.

```
1  Event agvLoaded
2    name : "lager_agv_loaded"
3    value : False
4  End
5
6  TransportOrderStep tos_warenlager1
7    Location : warenlager1
8    FinishedBy : agvLoaded == True
9  End
10
11 TransportOrderStep tos_produktion1
12   Location : produktion1
13    # JSON inline Initialisierung
14   Parameters : {"loadDirection" : "unload" , "liftHeight": "0"}
15 End
16
17 Task hello_world
18   Transport
19   From tos_warenlager1
20   To tos_produktion1
21 End
```

Die Entladung wird durch eine *JSON*-Inline-Initialisierung in Z. 14 beschreiben. Dabei wird davon ausgegangen, dass das FTF einen Parameter *loadDirection* besitzt, der entweder das Laden (*Load*) oder das Entladen (*Unload*) beschreibt.

Sequenzielle Transporte und parallele Tasks

Innerhalb einer *Task* können diverse Transportaufgaben beschrieben werden. Hierfür gilt die Auflistung 4.20 als Beispiel. In Z. 2–4 ist ein einfacher Transport definiert. Nach Abschluss des Transports wird der nächste Transport (Z. 5–8) ausgeführt. Innerhalb einer *Task* können so beliebig viele Transporte nacheinander ausgeführt werden.

Auflistung 4.20: Eine einfache Verkettung von zwei Transporten. Sobald der erste Transport (Z. 2–4) abgeschlossen ist, soll ein zweiter Transport (Z. 5–8) ausgeführt werden.

```
1 Task hello_world
2   Transport
3   From tos_warenlager1
4   To tos_produktion1
5   Transport
6   From tos_produktion2
7   To tos_warenlager2
8 End
```

Wichtig zu erwähnen ist, dass bei jedem Transport auch individuelle *TransportOrderSteps* verwendet werden können.

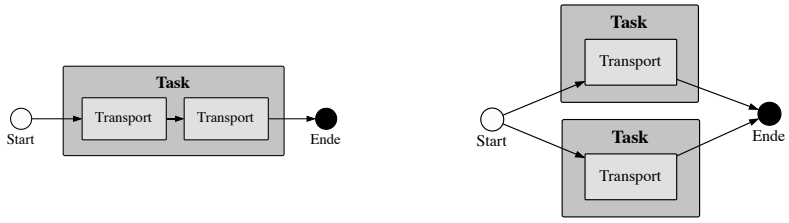
Werden zwei *Tasks* unabhängig voneinander innerhalb von *MFDL* definiert, so werden diese parallel abgearbeitet. In dem Beispiel in Auflistung 4.21 existieren zwei definierte Tasks (Z. 1-5 und Z. 6-10).

Auflistung 4.21: Parallele Ausführung von *Tasks*.

```
1 Task hello_world
2   Transport
3   From tos_warenlager1
4   To tos_produktion1
5 End
6 Task task2
7   Transport
8   From tos_produktion2
9   To tos_warenlager2
10 End
```

Die Zuweisung, welches FTF den jeweiligen Transport durchführt, wird erst zur Laufzeit ausgehandelt. In Abb. 4.13 sind die serielle Abarbeitung von Transporten (vgl. Abb. 4.13a) und die parallele Abarbeitung von *Tasks* (vgl. Abb. 4.13b) dargestellt. Generell werden alle *Tasks* auf Anforderung ausgeführt, sofern diese einmal definiert sind (vgl. LA4). Dies trifft allerdings nicht zu, wenn Abhängigkeiten unterhalb von *Tasks* oder von Transporten bestehen. Bei der sequenziellen Abarbeitung der Transporte ist die Reihenfolge relevant. Der zweite Transport soll erst ausgeführt werden, wenn der erste abgeschlossen ist. Da die Dauer des ersten Transports von den physikalischen Eigenschaften des Fahrerlosen Transportfahr-

zeugs und den vorherigen Aufträgen abhängt, ist eine Vorausplanung kompliziert. Hier könnten zwei aufeinanderfolgende Aufträge einem FTF zugewiesen werden.



(a) Serielle Verkettung von Transporten (b) Parallele Abarbeitung von Tasks

Abb. 4.13: Sequenzielle Verkettung von Transporten innerhalb einer Task und die parallele Ausführung von Tasks

Verkettung von Tasks

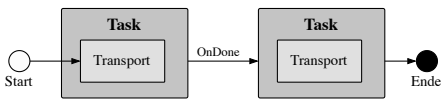
Die Verkettung von *Tasks* setzt voraus, dass eine Tasks abgeschlossen ist, bevor eine weitere gestartet wird. So wird, wie in Auflistung 4.22 dargestellt, erst die *Task hello_world* ausgeführt (Z. 1–6). Durch die Verwendung des *OnDone*-Ausdrucks in Z. 5 wird *Task hello_world* mit der *Task task2* (Z. 8–12) verkettet.

Auflistung 4.22: Eine einfache Verkettung von zwei *Tasks*. Sobald die erste *Task hello_world* abgearbeitet ist, wird die zweite *Task task2* ausgeführt.

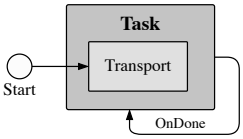
```
1  Task hello_world
2    Transport
3    From tos_warenlager1
4    To tos_produktio1
5    OnDone : task2
6  End
7
8  Task task2
9    Transport
10   From tos_produktion2
11   To tos_warenlager2
12 End
```

Es besteht auch die Möglichkeit, eine Endlosschleife zu realisieren. Dafür muss lediglich in der Bedingung *OnDone* ein Verweis auf die eigene Task erfolgen (*OnDone hello_world*), wie in Abb. 4.14 dargestellt. Abb. 4.14a ist ein Beispiel für eine Verkettung von zwei *Tasks* mithilfe des Schlüsselworts *OnDone*.

Mithilfe der Verkettung von *Tasks* und der seriellen Ausführung von Transporten ist die LA3 abgedeckt. Selbst bei einer Endlosschleife kann bei jeder Ausführung ein neues Fördermittel bzw. ein FTF ausgewählt werden.



(a) Eine serielle Verkettung von *Tasks* durch Verwendung der *OnDone*-Bedingung



(b) Eine *Task* die sich immer wieder selbst ausführt.

Abb. 4.14: Möglichkeiten zur Verkettung von *Tasks*

Die Pickliste und Kindsaufgaben

Mithilfe einer Pickliste kann eine Abholung der Ware an mehreren Orten erfolgen. Daher entspricht diese einem *N:1*-Transport, der Abholung von Waren von *N* verschiedenen Orten, welche zu einer einzelnen Senke transportiert werden. Dabei kann die Reihenfolge der Orte für die Abholung individuell optimiert werden. In der Auflistung 4.23 sind in Z. 3 zwei *TransportOrderSteps* angegeben. Jeder *TransportOrderStep* beschreibt die Abholung individuell.

Auflistung 4.23: Realisierung einer Pickliste: Waren werden von zwei verschiedenen Orten (*tos_warenlager1* und *tos_warenlager2*) zu einem gemeinsamen Ort (*tos_produktion1*) transportiert.

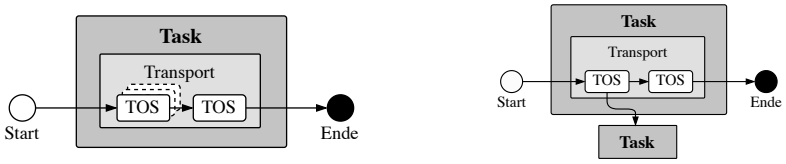
```
1 Task hello_world
2   Transport
3   From tos_warenlager1, tos_warenlager2
4   To tos_produktion1
5 End
```

Ein weiteres Szenario, welches mit *MFDL* realisiert werden kann, ist die Kindsaufgabe. Innerhalb einer Fabrikhalle soll eine Palette von *A* nach *B* transportiert werden. Sobald die Abholung an Ort *A* erfolgt ist, soll eine neue Palette an diesen Ort transportiert werden. Hierfür wird innerhalb der Task *task1* eine Kindsaufgabe gestartet. Somit ist eine Kindsaufgabe eine Aufgabe, welche innerhalb einer Aufgabe entsteht. Dieses Szenario ist in der Auflistung 4.24 realisiert: Sobald eine Abholung an dem Ort *produktion1* des *TransportOrderSteps* (Z. 1-4) erfolgt ist, wird eine neue Task durch die Verwendung der *OnDone*-Bedingung (Z. 3) gestartet.

Auflistung 4.24: Eine *Task* erstellt eine Kindsaufgabe, nachdem eine Abholung stattgefunden hat.

```
1  TransportOrderStep tos_produktion_refill
2    Location : produktion1
3    OnDone : task2
4  End
5
6  Task task1
7    Transport
8    From tos_produktion_refill
9    To tos_produktion2
10 End
11
12 Task task2
13   Transport
14   From tos_produktion1
15   To tos_warenlager2
16 End
```

Die Kindsaufgabe wird nebenläufig zu der *Task* ausgeführt, welche die Kindsaufgabe erstellt. In der nachfolgenden Abb. 4.15 sind die Möglichkeiten zur Pickliste bzw. zur Erstellung einer Kindsaufgabe dargestellt.



(a) Eine Pickliste besteht aus mehreren Abholorten, für den jeweils ein eigener *TransportOrderSteps* angegeben ist

(b) Eine Task erstellt eine Kindsaufgabe, nachdem eine Abholung erfolgt ist.

Abb. 4.15: Realisierung einer Pickliste und einer Kindsaufgabe

Randbedingungen

Transporte, die bestimmte Randbedingungen erfüllen sollen, können mithilfe des Schlüsselwortes *Constraint* definiert werden. In der folgenden Auflistung 4.25 ist in Z. 5 eine Zeitspanne mithilfe der *JSON*-Inline-Syntax definiert. Dabei soll der Transport im Zeitraum von 14:05 Uhr bis 14:45 Uhr stattfinden. Diese Restriktion wird bei der Auftragszuweisung berücksichtigt, entweder vom Prozess selbst, oder von den FTF, welche sich um den Auftrag mit der Restriktion bewerben.

Auflistung 4.25: Durchführung eines Transport, der innerhalb einer Zeitspanne durchgeführt werden soll.

```
1 Task hello_word
2   Transport
3   From tos_warenlager1
4   To tos_produktion1
5   Constraint : {"TransportStart" : "05 14 * * *", "TransportFinished" : "45
        14 * * *"}
6 End
```

Aber auch die maximalen Kosten oder Orte, die das Fördermittel z. B. nicht besuchen darf, könnten hier definiert werden.

4.6 Architektur und Integration von MFDL

Im Folgenden werden die Architektur und die Integration der *MFDL* innerhalb eines CPPS vorgestellt. In der Abb. 4.16 sind die wichtigsten Komponenten der *MFDL* und die Feldkomponenten eines CPPS (z. B. FTF, Sensoren, ...) dargestellt.

Die *Infrastruktur*-Komponente dient zur Anbindung an die Feldkomponenten innerhalb eines CPPS. Das *Eventing* dient zur Verteilung der erhaltenen Daten des CPPS und der Einhaltung von bestimmten Events wie beispielsweise einer *Time*-Struktur. Mithilfe der *Core*-Komponente werden die erhaltenen Daten intern verifiziert und validiert, sodass nur Daten verarbeitet werden, die auch ein entsprechendes Format einhalten. In der *Execution Engine (EE)* wird das eigentliche *MFDL*-Programm ausgeführt.

4.6.1 Infrastruktur

In der *Infrastruktur*-Komponente werden Daten von den CPPS-Feldkomponenten wie FTF oder Sensoren empfangen und in eine interne Darstellung umgewandelt. So können verschiedene Repräsentationen über verschiedene Kommunikationsframeworks (z. B. via MQTT, FIWARE, ...) versendet werden, die unterschiedlichen Protokolle (TCP, HTTP, Websockets, ...) verwenden. Klassische Repräsentationen für den Datentransfer können *JSON* [Cro06], *CSV* [Int05], *XML* [BPS+08] oder auch proprietäre Darstellungen sein. Diese wiederum können in einem bestimmten Informationsmodell wie NGSI-LD [ETS20] eingebunden sein.

Mit der Integration eines *Adapters* werden die o. g. Darstellungen auf eine einheitliche Form transformiert. Hierbei existiert eine bidirektionale Kommunikation zwischen *MFDL* und den Feldkomponenten: Zum einen müssen neue Sensordaten oder Zustände von den FTF empfangen, zum anderen auch Steuerbefehle an ein FTF gesendet werden können. Diese Interaktion ist auch in der Abb. 4.17 dargestellt.

Ein *MFDL*-Programm interagiert mit einem Sensor über einen MQTT-Broker, während es für die Interaktion mit dem FTF mit einem proprietären Kommunikationskanal kommuniziert. Als Vereinfachung sind der MQTT-Broker und die dedizierte Kommunikation nicht dargestellt, sondern lediglich die logische Kommunikation. Zunächst meldet sich das MFDL-Programm für die Statusänderungen des

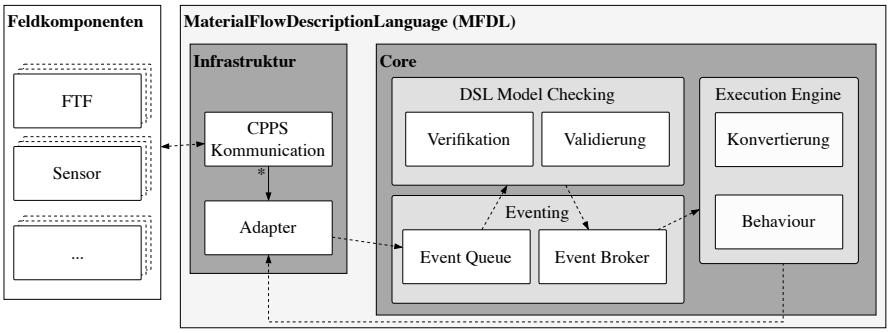


Abb. 4.16: Architektur und Integration der *MFDL* in einem CPPS zur Interaktion mit Feldkomponenten.

Sensors an. Die Anmeldung wird synchron direkt bestätigt bzw. abgelehnt. Dies gilt auch bei der Anmeldung für Statusänderungen des FTF. Im Anschluss weist die EE dem FTF eine Transportaufgabe zu. Diese wird ebenfalls synchron vom Fahrzeug bestätigt. Danach sendet das FTF asynchron seine Statusänderungen, welche sich auf die zuvor zugewiesene Transportaufgabe beziehen, an die *Execution Engine* weiter. Die CPPS-Kommunikation und die Adapter sind für die entsprechende Verpackung und Abstraktion der Kommunikation zuständig.

4.6.2 Core

Die *Core*-Komponente von *MFDL* besteht aus drei Teilkomponenten: der Verarbeitung von Events (*Eventing*), der Überprüfung der domänenspezifischen Sprache *DSL Model Checking* und die Ausführung des Programms *Execution Engine*. Diese werden nun im Einzelnen erklärt.

Eventing

Das *Eventing* hilft bei der Verbreitung der erhaltenen Events von der Infrastruktur. Die *Infrastruktur*-Komponente transformiert die erhaltenen Events in ein gewünschtes Format. Nach der Transformation sind diese Events in der *Event Queue*, wel-

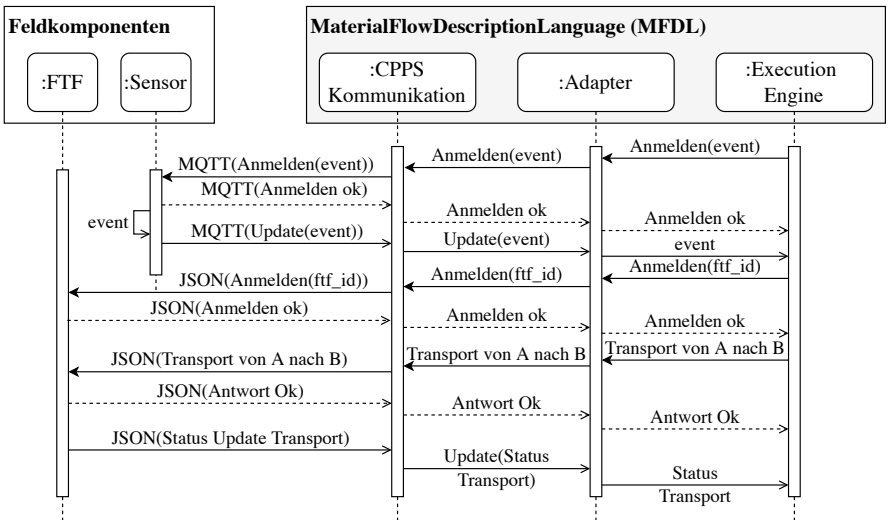


Abb. 4.17: Interaktion der *MFDL* Komponenten zur Laufzeit mit den entsprechenden Feldkomponenten, einem Fahrerlosen Transportfahrzeug (FTF) und einem Sensor.

che sequenziell abgearbeitet werden. Die einzelnen Komponenten innerhalb eines *MFDL*-Programms kommunizieren über das *Eventing*.

DSL Model Checking

Das *Model Checking* führt mit den eingegebenen Daten, der DSL, eine formale Überprüfung der Syntax und Semantik durch. Dabei wird in der *Verifizierung* überprüft, ob das *MFDL*-Programm auch der definierten Grammatik entspricht. Die Grammatik ist dabei mithilfe von *ANTLR*[9] [Par13] spezifiziert worden und enthält einen *Lexer* und einen *Parser*. Das *MFDL*-Programm wird nun in einen abstrakten Syntaxbaum (engl.: *Abstract Syntax Tree (AST)*) transformiert. Mithilfe des ASTs wird in der *Validierung* überprüft, ob die Semantik korrekt ist und alle Attribute den entsprechenden Wertebereich bzw. die entsprechenden Zuweisungen haben. Der *AST* ist die Eingabe für die *Execution Engine*.

Execution Engine

Innerhalb der *Execution Engine* wird der *AST* in eine interne Darstellung überführt (*Konvertierung*). Diese Darstellung basiert auf einer Erweiterung für ein Petri-Netz (PN) [Mur89, ZIN+08], mit dessen Hilfe die definierten *Tasks* (vgl. Abschn. 4.4.3) traversiert werden. Petri-Netze sind aufgrund ihrer formalen Beschreibung und ihrer mathematischen Einfachheit weit verbreitet, um komplexe Verhalten einfach und klar darzustellen [HM10, ZIN+08].

Ein PN ist ein gerichteter Graph, bestehend aus Orten (engl.: *places*), Übergängen (engl.: *transitions*), Kanten (engl.: *edges*) und Markern (engl.: *token*). Die ersten drei werden für die Systemmodellierung verwendet, um bspw. Zustände, Konditionen und Events zu beschreiben. Die Marker werden für das dynamische Verhalten zur Laufzeit verwendet. Formal kann das PN als ein Quintupel $S = \{P,T,I,O,M_0\}$ dargestellt werden. Dabei repräsentiert $P = \{p_1, p_2, ..., p_m\}$ ein endliches Set an Orten, $T = \{t_1, t_2, ..., p_n\}$ ist ein endliches Set an Übergängen (mit $P \cup T \neq \emptyset$ und $P \cap T \neq \emptyset$), $I : T \times P$ ist eine Eingangsmatrix für die Kanten von Orten auf Übergänge, $O : T \times P \rightarrow N$ ist eine Ausgangsmatrix für die Kanten von Übergängen auf Orte und $M_0 : P \rightarrow N$, welche die initiale Anfangsmarkierung darstellt.

Als Beispiel für die Konvertierung der MFDL in ein PN wird die nachfolgende Auflistung 4.26 verwendet. Das konvertierte PN ist in Abb. 4.18 dargestellt. Es handelt sich hierbei um einen Transport einer Palette vom Warenlager in die Produktion. Der Einfachheithalber sind Parameter für die Be- und Entladung vernachlässigt.

Auflistung 4.26: Beispiel einer einfachen Transportaufgabe, dargestellt mithilfe der MFDL.

```
1 Import intralogistic
2
```

[9] ANother Tool for Language Recognition (ANTLR):https://www.antlr.org

```
3  Location warenlager
4      id: "0x4711"
5      type: "pallet"
6  End
7
8  Location produktion
9      id: "0x4712"
10     type: "pallet"
11 End
12
13 TransportOrderStep abholung
14     Location : warenlager
15 End
16
17 TransportOrderStep lieferung
18     Location : produktion
19 End
20
21 Task hello_world
22     Transport
23     From abholung
24     To lieferung
25 End
```

Die Konvertierung in ein PN ermöglicht eine implizite Flusskontrolle. Ferner müssen nicht alle Transitionen zwischen den einzelnen Schritten angegeben werden, da diese durch die High-Level-Beschreibung innerhalb *MFDL* automatisiert erstellt werden.

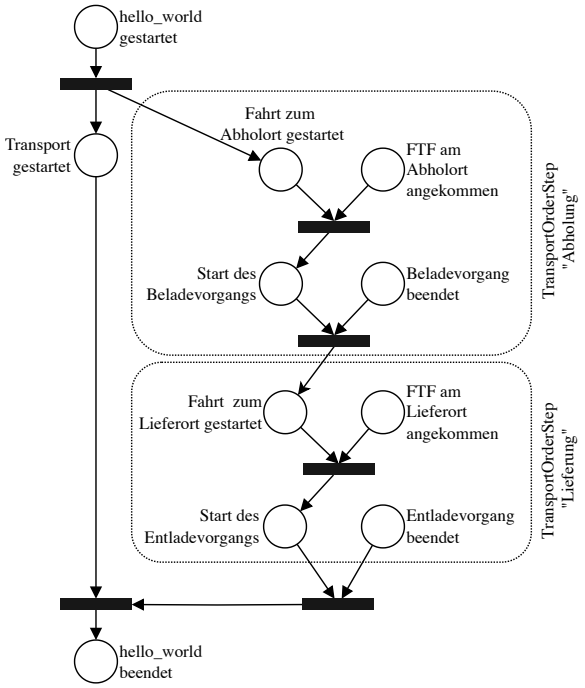


Abb. 4.18: Konvertierung von *MFDL* in ein Petri-Netz

Nach der Konvertierung in das PN kann nun die Interaktion mit den Feldkomponenten (bspw. den FTF) erfolgen. Der gesamte Interaktionsablauf ist in dem folgenden Sequenzdiagramm in Abb. 4.19 dargestellt. Um die Selbstorganisation des Transports zu fördern, wird eine Möglichkeit benötigt, um nach den entsprechenden Feldkomponenten zu suchen (vgl. Abschn. 3.3). Diese wird nachfolgend *Service Registry* genannt. Die *Execution Engine* hat Zugriff auf diese *Service Registry* und kann die zuvor angemeldeten Fahrzeuge darüber suchen und auffinden. Nachdem die *Execution Engine* die Liste mit den verfügbaren FTF erhalten hat, können die Status der entsprechenden Fahrzeuge eingeholt werden. Hier wird davon ausgegangen, dass lediglich mit den FTF interagiert wird, die für einen Transport infrage kommen. Im Rahmen der Selbstorganisation des Materialflusses und der Dezentralisierung verhandelt jede Transportaufgabe mit den entsprechenden FTF und entscheidet sich auch für ein Fahrzeug. Aus Platzgründen wurde diese Interaktion vernachlässigt. Im Anschluss, nach der Auswahl eines FTF, erfolgt die Zuweisung an das Fahrzeug über die entsprechende Service-Schnittstelle (vgl. Abschn. 4.1). Hier gilt es zu beachten, dass der synchrone Service-Aufruf eine unmittelbare Antwort auf eine Anfrage liefert. War die Antwort positiv, so ist der Transport gestartet und das Fahrzeug kann seine Status-Updates zu jedem jeweiligen Transport an die *Execution Engine* schicken.

Die Granularität der Updates zwischen dem FTF und der *Execution Engine* hängt von der Implementierung ab. So kann für jede Positionsänderung des FTF ein Update an die *Execution Engine* des Transportauftrags geschickt werden. Aus Gründen der Übersichtlichkeit wurde in diesem Beispiel der Beginn der Interaktion, die Zuweisung und die Ankunft am jeweiligen Ort im Sequenzdiagramm dargestellt.

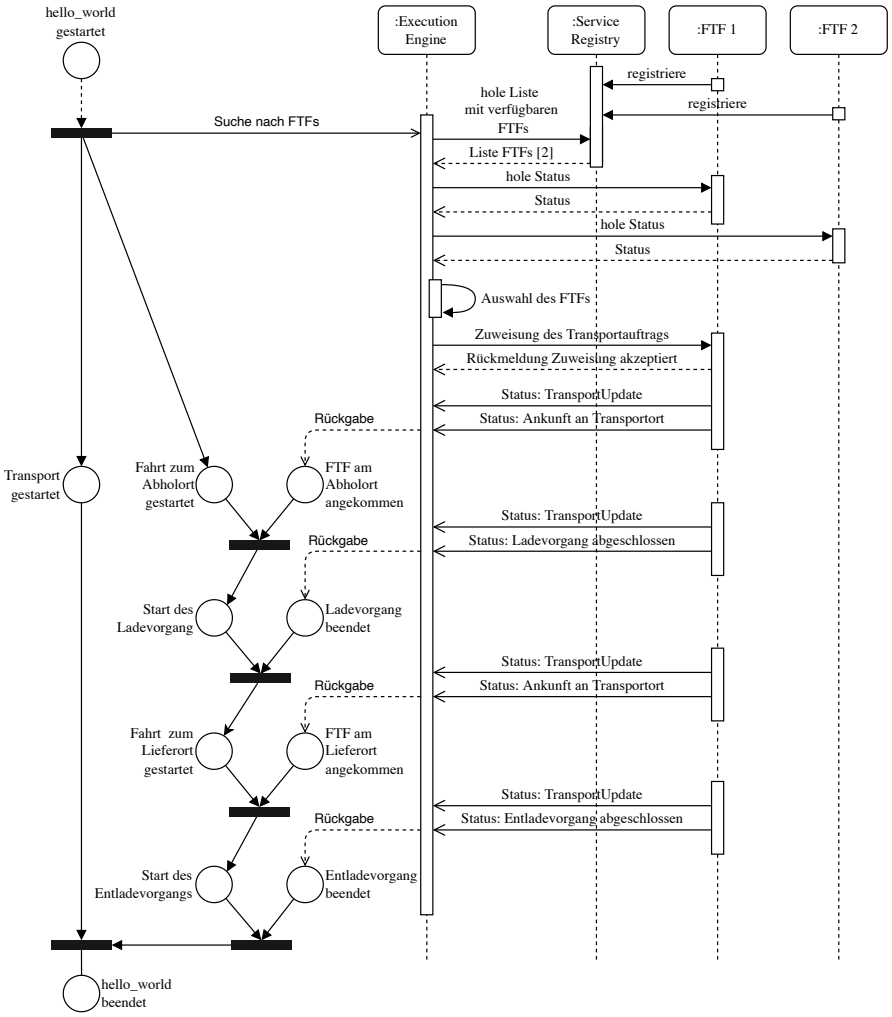


Abb. 4.19: Konvertierung eines *MFDL*-Programms in ein Petri-Netz und der Interaktion zwischen der *Execution Engine* und zwei Fahrerlosen Transportfahrzeugen

Kapitel 5
SOLA: Ein dezentral organisiertes Kommunikationsframework

"*Knowledge is wasted when it isn't shared.*"

— J.M. Cornwell

Nach Betrachtung der Grundlagen der Vernetzung von cyberphysischen Produktionssystemen (vgl. Kapitel 3), eines ganzheitlichen Informationsmodells eines Fahrerlosen Transportfahrzeugs und der digitalen Repräsentation eines Transportauftrags (vgl. Kapitel 4) wird nachfolgend das Konzept der selbstorganisierenden Kommunikation vorgestellt. Ziel ist es, dass Entitäten untereinander vernetzt werden und sich die Kommunikation der Anzahl an Teilnehmern anpasst. Das hier vorgestellte Overlay-Netzwerk, nachfolgend auch SOLA genannt, organisiert sich dezentral, folglich ohne einen zentralen Broker (broker-less). SOLA ist eine Sammlung von unabhängigen Komponenten, die als ein einzelnes, kohärentes System erscheint (vgl. Abb. 5.1).

Dabei repräsentieren die Peers physikalische (z. B. ein Fahrerloses Transportfahrzeug oder eine stationäre Maschine) und virtuelle Entitäten (z. B. einen

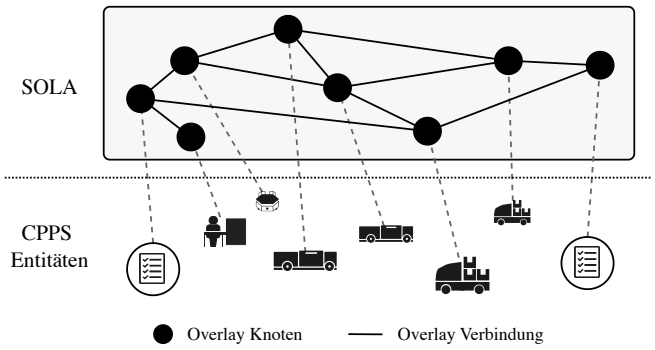


Abb. 5.1: Das dezentral organisierte Overlay-Netzwerk SOLA erscheint als ein kohärentes System, bei dem Teilnehmer Teil des Systems sind.

Transport- oder einen Produktionsauftrag). Wichtig ist dabei zu erwähnen, dass kein einzelner Peer über ein globales Wissen verfügt. Stattdessen hat jeder Peer nur ein beschränktes Wissen, auch lokales Wissen genannt und eine Menge an Verbindungen zu anderen Peers (vgl. Abschn. 5.2 und Abschn. 5.3). Durch eine Aggregation der lokalen Sichten aller Peers kann ein globales Wissen erstellt werden, bspw. welche Dienste und Informationen von Peers angeboten werden (vgl. Abschn. 5.4). Diese Informationen und das darin enthaltene Wissen kann an alle Peers oder nur eine Teilmenge davon verteilt werden (vgl. Abschn. 5.5).

5.1 Motivation

Ein Overlay-Netzwerk ist eine Sammlung von logischen Verbindungen, welche eine Menge an Knoten miteinander vernetzt. Jeder Knoten, der ein Teil des P2P-Netzwerks bzw. des Overlay-Netzwerks ist, wird Peer genannt. Nachfolgend wird jeder Peer von einer Entität innerhalb des CPPS repräsentiert. Denkbar sind auch die Szenarien, in dem mehrere CPPS-Entitäten gemeinsam einen Peer (N:1) repräsentieren bzw. dass eine CPPS-Entität mehrere Peers (1:N) repräsentiert. Der Aufbau des Overlay-Netzwerks ist nicht von der darunter liegenden Schicht abhängig, dennoch können Informationen aus der unteren Schicht verwendet werden. [KKH+13] Im Folgenden werden die bereits eingeführten funktionalen und nicht-funktionalen Anforderungen aus Abschnitt 3.3.1 für das Overlay-Netzwerk erweitert.

5.1.1 Funktionale Anforderungen

Ein P2P-Netzwerk ist ein Zusammenschluss von autonomen Peers, unabhängig davon, ob dieses Netzwerk strukturiert oder unstrukturiert ist (vgl. Abschn. 3.2.1). Die verbundenen Peers stellen neben dem Overlay-Netzwerk eine bestimmte Funktionalität bereit, wie bspw. eine Speicher- oder Applikationsschicht, oder wie im Folgenden ein dezentral organisiertes Kommunikationsframework. Um die gewünschte Funktionalität zu formalisieren, werden nun zunächst die funktionalen Anforderungen beschrieben, um hieraus die geeigneten Schnittstellen abzuleiten.

- Um Teil eines Netzwerks zu werden, muss eine Entität dem Netzwerk beitreten oder dieses verlassen.
- Knoten bzw. Peers handeln autonom. Tritt ein Knoten bei bzw. verlässt ein Knoten das Netzwerk, werden die relevanten Peers diesbezüglich informiert.
- Peers können ihre (verfügbaren) eigenen Ressourcen, angebotenen Dienste und sonstigen Daten in dem Overlay-Netzwerk ablegen. Ein Beispiel für einen Dienst ist der *Transport* eines Fahrerlosen Transportfahrzeugs. Ein Beispiel für eine Ressource hingegen ist Rechenleistung, welche wiederum Dienste anbieten könnte.

- Ein Peer kann entweder seine eigenen Daten (Lokale Daten) oder Daten und Informationen von anderen Peers (Remote Daten) speichern. Ein CPPS besteht aus einer Menge unterschiedlicher Typen, sodass eine Harmonisierung der Daten nur schwer möglich ist. Peers können Daten hinzufügen, aktualisieren, abrufen und löschen.
- Peers können auch nach Ressourcen, Diensten oder Daten anderer Peers suchen. Die Suche kann ein einfaches Attribut enthalten oder komplexere Ausdrücke, basierend auf boolescher Algebra (z. B. $Attribut_1 < 42$ *AND* $Attribut_2 \neq 99$).
- Peers können andere Peers über Statusänderungen informieren. Hierbei handelt es sich um eine selektive Verteilung von Informationen (vgl. Abschn. 3.2.2). Dabei erfolgt diese durch die Verwendung von Unicasts und nicht von Multicasts (vgl. Abschn. 3.4).

Die hier aufgeführten funktionalen Anforderungen sind in Abb. 5.2 dargestellt. Beispielhafte Teilnehmer für SOLA sind sowohl FTF als auch Transportaufträge (TO). Dabei verwalten einzelne Peers ihre eigenen Daten (Lokale Daten) und, wenn notwendig, auch Daten von anderen Peers (Remote Daten). Peers können Daten hinzufügen (publish) oder nach anderen Peers über eine dedizierte Suchanfrage (engl.: *query*) abrufen.

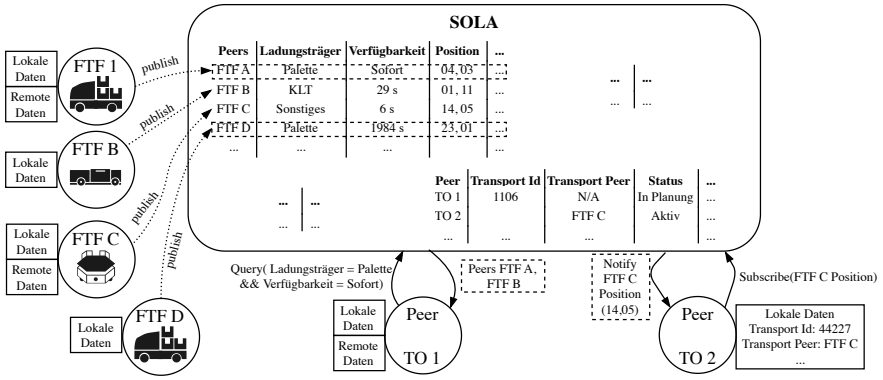


Abb. 5.2: Übersicht der Architektur von SOLA. Fahrerlose Transportfahrzeuge (FTF) oder Transportaufträge (TA) bilden gemeinsam das Overlay-Netzwerk SOLA.

Die Möglichkeit zur Realisierung einer dezentralen Vernetzung von Entitäten in einem CPPS resultiert in einer Gruppierung folgender Funktionen [DZD+03, PEK+07, KKH+13, AS04, MCL20, MG20]:

Verwaltung des P2P-Netzwerks (Vernetzung)

- **start**(): Funktion, um Teil des P2P-Netzwerks zu werden
- **stop**(): Funktion zum ordnungsgemäßen Verlassen des P2P-Netzwerks

- **set_cb_recv_data(cb)**: Funktion zum Setzen einer Rückruffunktion (engl.: *callback*), die automatisch aufgerufen wird, sobald neue Daten über die Netzwerkschnittstelle (engl.: *network interface*) eintreffen

Verwaltung von Informationen (Verteilte Datenspeicherung)

- **add_data(key, value)**: Funktionen, um Daten zu einem bestimmten *key* verteilt in dem P2P-Netzwerk abzulegen
- **update_data(key, value)**: Funktionen, um die Daten zu einem bestimmten *key* in dem P2P-Netzwerk zu aktualisieren
- **remove_data(key, value)**: Funktionen, um Daten zu einem bestimmten *key* verteilt aus dem P2P-Netzwerk zu entfernen
- **find_data(query)**: Funktionen zur Suche nach Daten im P2P-Netzwerk. Daten können hier unter anderem einen Service repräsentieren
- **use_service(service)**: Funktion zum Aufruf eines Services

Verteilung von Informationsänderungen (Event Dissemination)

- **subscribe_to_event(event_type)**: Funktion, um sich für eine Änderung für ein bestimmtes Event anzumelden.
- **publish_event(event_content)**: Funktion zur Verbreitung der Änderung eines bestimmten Event-Typens bzw. einer Nachricht.
- **unsubscribe_from_event(event_type)**: Funktion zur Abmeldung eines Abonnements der Statusänderungen.

Diese hier vorgestellten Funktionen sind zusammengefasst in Abb. 5.3.

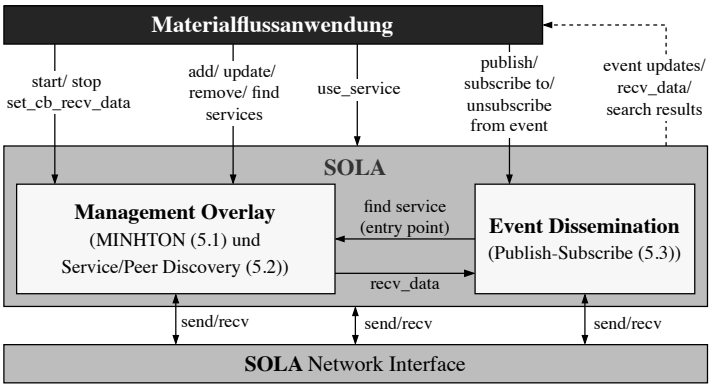


Abb. 5.3: Übersicht der abgebildeten Funktionen innerhalb SOLA.

5.1.2 Nicht-funktionale Anforderungen

Im Folgenden werden die nicht-funktionalen Anforderungen hinsichtlich ihrer Qualitätseigenschaften beschrieben. Während sich die funktionalen Anforderungen auf die Aufgabe und die Schnittstellen beziehen, beziehen sich die nicht-funktionalen Anforderungen auf die Qualität der Lösung, wie z. B. die notwendigen Kosten in Bezug auf die Kommunikation zur Erfüllung der funktionalen Anforderungen.

- **Skalierbarkeit**: Eine mögliche Lösung ist skalierbar, wenn u. a. die Anzahl an Peers in einem Netzwerk, die Anzahl an Verbindungen die ein einzelner Peer zu verwalten hat, oder der ein- und ausgehende Datenverkehr berücksichtigt werden. Dabei sind idealerweise die Kosten, die ein einzelner Peer zu tragen hat, nach oben hin beschränkt und wachsen nicht linear oder exponentiell mit der Anzahl an verfügbaren Peers im Netzwerk [YAH+17, APR+18, CSRL+19]
- **Datenaktualität**: Sucht ein Peer nach Informationen bzw. Daten oder Diensten eines anderen Peers, so ist die Aktualität der Informationen von Bedeutung. Eine hohe Aktualität entspricht einem jungen Informationsalter. Diese ist verbunden mit einer hohen Genauigkeit, welche wiederum mit hohen Kosten zur Sicherstellung dieser verbunden ist. Das Verhältnis von Aktualität zu den Kosten sollte dabei angemessen sein.
- **Zuverlässigkeit**: Unabhängig von der Anzahl der Teilnehmer wird vorausgesetzt, dass alle Teilnehmer innerhalb des Overlay-Netzwerks erfasst werden. Zu jedem Zeitpunkt kann eine globale Sicht auf das gesamte Netzwerk durch eine Aggregation von lokalen Informationen einzelner Peers erstellt werden. Eine Zuverlässigkeit ist gegeben, wenn nach Informationen bzw. Daten oder Diensten gesucht wird und die Menge an Peers gefunden wird, welche die Suchanfrage erfüllen.
- **Effizienz**: Die Kosten des Overlay-Netzwerks werden in Form der benötigten Bandbreite bzw. der ein- und ausgehenden Nachrichten eines einzelnen Peers gemessen. Andere Kosten, wie bspw. die Verarbeitung von Informationen und die dafür benötigte Rechenzeit sind nicht relevant. Das Overlay-Netzwerk sollte dabei nur einen kleinen Teil an Netzwerkbandbreite verbrauchen. Neben dem Verbrauch an Netzwerkbandbreite ist auch die benötigte Anzahl an Nachrichten (Hops) für die Erfüllung einer Funktion von Bedeutung, wie des Betretens bzw. Verlassens des Netzwerks, des Auffindens eines Peers anhand bestimmter Kriterien oder der Verteilung einer Statusänderung. Werden Nachrichten mehrfach versendet, so hat dies einen Einfluss auf die Effizienz. Daher sollte das Versenden von redundanten Nachrichten minimal gehalten werden.
- **Stabilität**: Betritt bzw. verlässt ein Peer das Netzwerk, so passt sich das Netzwerk schnell der neuen Situation an. Ferner wird nach einer kurzen Stabilisierungsphase die zuvor erbrachte Qualität sichergestellt und Leistung erbracht. Der Ausfall eines einzelnen Peers soll nicht zu einem Ausfall des gesamten Systems führen.

5.2 Das Management Overlay MINHTON

Das *Management Overlay* hat die Aufgabe, ein Overlay-Netzwerk zu erstellen und zu verwalten. Dabei kann ein Overlay-Netzwerk unterschiedliche Strukturen annehmen, wie z. B. einen Ring (vgl. Chord [SMK+01]), ein multi-dimensionales Gitter (vgl. CAN [RFH+]), einen Stern oder einen Baum. In einer Stern-Topologie sind alle Peers mit einem Knoten verbunden, welcher dem Client-Server-Modell ähnelt (vgl. Abschn. 3.2.1). In einer Ring-Topologie wie Chord sind alle Knoten zirkulär miteinander verbunden. Allerdings können keine Hierarchien abgebildet werden und alle Knoten, unabhängig von ihren Eigenschaften, haben die gleiche Anzahl an Verbindungen. Steigt die Anzahl an Knoten im Netzwerk, so steigt auch die Anzahl an Verbindungen je Knoten (vgl. [SMK+01, Abschn. 4.3]). In einem Baum hingegen kann mit einer fixen Anzahl an m Kindern (engl.: *fanout*) eine Hierarchie aufgebaut werden. Zwischen dem Wurzelknoten (engl.: *root*) und dem Knoten auf dem höchsten Level besteht immer ein Pfad mit den Kosten $O(\log_{fanout} N)$. Ferner kann bei einem Baum auch eine einfache Aggregation von Informationen zwischen der fixen Anzahl an Kindern und dem Elternknoten erfolgen. Um bspw. die Größe eines Netzwerks zu bestimmen, können alle Knoten ihre lokalen Informationen an ihren Elternknoten schicken. Sobald der Wurzelknoten die Informationen von seinen Kindern erhalten hat, ist die Aggregation von lokalen Informationen aller Knoten abgeschlossen, sodass ein globales Wissen entstanden ist. Baumbasierte Datenstrukturen sind in der Informatik weit verbreitet und ausgiebig betrachtet worden [Knu98, FO82]. Es gibt eine Vielzahl an Veröffentlichungen, welche sich mit dem Einfügen und Suchen von Daten beschäftigen. Zusätzlich können Hierarchien einfach dargestellt werden. Das hier entworfene *Management Overlay* basiert auf einer Baumstruktur. Teile dieses Abschnitts basieren auf den Vorarbeiten des Autors in [DGB21, Las21, DGB22].

Eine bekannte Baumstruktur ist das *BAlanced Tree Overlay Network* (BATON), bei dem jeder Knoten maximal zwei Kinder haben kann [JOV05]. Die Baumstruktur *BAlanced m-ary Tree Overlay Network* (BATON*) ist eine Erweiterung von BATON, bei der jeder Knoten eine festgelegte maximale Anzahl an Kindern m haben kann, mit $m \in \mathbb{N} \geq 2$ [JOT+06]. Die Anzahl an Kindern wird auch *Fanout* genannt. Folglich ist BATON eine Sonderform von BATON*, bei dem die maximale Anzahl an Kindern $m = 2$ ist. Sowohl BATON als auch BATON* sind beides höhenbalancierte Bäume, basierend auf der folgenden Definition:

Definition höhenbalancierter Baum [JOT+06]
Ein Baum ist dann und nur dann höhenbalanciert, wenn sich die Höhen zweier Teilbäume seiner Kinder an jedem beliebigen Knoten des Baumes um höchstens 1 unterscheiden.

Innerhalb von BATON* ist es nicht garantiert, dass ein Baum mit minimaler Höhe aufgebaut wird. Dies resultiert aus der lokalen Sicht der einzelnen Peers im

Netzwerk. Vielmehr können zwei höhen-balancierte Bäume mit der gleichen Anzahl von Knoten sehr unterschiedlich dargestellt werden.

Eine Herausforderung ist auch die Erhaltung der Höhe des Baumes. Detektiert ein Knoten innerhalb BATON*, dass der Baum nicht mehr höhenbalanciert ist, so erfolgt eine kostspielige Rotation zur Wiederherstellung des Balancierungskriteriums (vgl. [JOT+06, Abschn. III-E]). Wenn ein Knoten während einer Rotation dem Netzwerk beitreten bzw. dieses verlassen will, kann dies zu fehlerhaften Einträgen in den Routingtabellen führen, sodass die Kosten ∞ entsprechen. Abgeschwächt werden kann dies mit einer besonderen Behandlung, bei der in einem rotierenden Baum einem Knoten ein Beitritt bzw. das Verlassen des Netzwerks temporär untersagt wird [MG20, DGB21]. Wann genau eine Rotation notwendig ist, kann nicht genau bestimmt werden, da der Baum nicht deterministisch aufgebaut wird. Zur Vermeidung der Rotation und um die Höhe minimal zu halten, wurde in [DGB21] eine Anpassung an BATON* vorgenommen. Diese beinhaltet unter anderem eine Anpassung hinsichtlich des Kriteriums des Baumes:

Definition nullbalancierter Baum [Cha12]
Ein m*-ary Baum ist nullbalanciert, wenn zwei Blattknoten sich in ihrem Level um höchstens* ≤ 1 *unterscheiden. Blattknoten, auch Nullknoten genannt, sind dabei als Knoten mit fehlenden Kindern definiert und befinden sich immer auf Level h oder* $h-1$.

Die nullbalancierte Baumstruktur nBATON* benötigt zwar keine Rotation mehr und ihre Höhe ist minimal, allerdings sind die Kosten zum Auf- und Abbau des Baums mit $O(\frac{N}{2m^2})$ im Vergleich zu BATON*'s $O(m \log_m N))$ kostenintensiv [DGB21]. Die Grenzwertbetrachtung in Gl. (5.1) für nBATON* zeigt, dass je größer der Fanout wird, desto weniger Hops benötigt werden, um das Overlay-Netzwerk aufzubauen.

$$\lim_{m\to\infty} \frac{N}{2 \cdot m^2} = 0 \ , \ \ N \in \mathbb{N} \textit{ und } m \ \geq 2 \tag{5.1}$$

Dies impliziert allerdings auch, dass ein Knoten unendlich viele Ressourcen zur Verfügung haben müsste. Daher ist die Betrachtung mit $m \to \infty$ kein realistisches Szenario. Im Weiteren wird eine effizientere Lösung zum Auf- und Abbau des Baumes vorgestellt, basierend auf dem nachfolgenden Systemmodell.

5.2.1 Systemmodell

In diesem Abschnitt wird die Baumstruktur *Minimal Height Tree Overlay Network* (MINHTON) vorgestellt. Konträr zu BATON* bzw. nBATON* erstellt MINHTON eine nullbalancierte, vollständige Baumstruktur basierend auf der folgenden Definition:

Definition nullbalancierter, vollständiger Baum [Cha12]
Ein m-ary Baum ist nullbalanciert und vollständig, wenn der Baum ein perfekter Baum der Höhe $h-2$ ist und neue Knoten auf dem Level $h-1$ von links nach rechts lückenlos hinzugefügt werden und entsprechend von rechts nach links lückenlos entfernt werden.

In einem nullbalancierten, vollständigen Baum können Knoten als Kinder nur von Knoten auf dem Level $h-2$ akzeptiert werden. Wie in der Abb. 5.4 dargestellt, ist dies auf dem Level 2 der Fall. Handelt es sich um einen perfekten Baum, bei dem Level $h-1$ komplett gefüllt ist, so wird ein neuer Knoten auf dem Level $h-1$ akzeptiert.

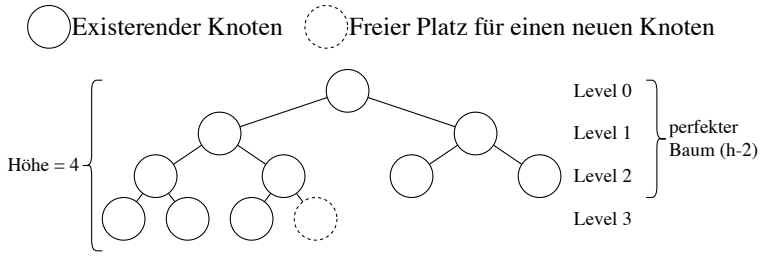


Abb. 5.4: In ein einem nullbalancierten, vollständigen m-ary Baum (hier $m = 2$) existiert nur ein freier Platz, da dieser von links nach rechts aufgebaut wird. Der Baum hat die Höhe $h = 4$, dass höchste Level ist $l = h-1 = 3$. Nur Nullknoten auf dem Level $h-2 = 2$ dürfen neue Knoten als Kinder akzeptieren.

Formal betrachtet ist ein P2P-Netzwerk ein Tupel (P,L), bei dem $P = \{p_1, \ldots, p_n\}$ einer endlichen Menge von Peers entspricht und $L \subseteq P \times P$ einer Menge an Verbindungen, bei der jede Verbindung $(p_i, p_j) \in L$ das Vorhandensein einer Verbindung für die Kommunikation zwischen diesen Peers zeigt. Aus Sicht eines Peers $p_i \in P$ besitzt dieser eine Menge an Verbindungen V_i, mit $V_i \subseteq P - \{p_i\} : \forall p_j \in V_i,\ \exists(a_i, a_j) \in L$. Unter der Annahme $|V_i| > 0$ hat jeder Peer p_i mindestens einen Link innerhalb von P und mit $|V_i| \ll |P|$ hat kein Peer p_i Verbindungen, idealerweise, zu allen anderen Peers.

Im Folgenden entspricht ein Peer p einem Knoten im m-ary Baum MINHTON. Jeder Peer hat eine logische ID, welche aus einem Level (l) und einer Nummer (n) besteht. Per Definition startet der Wurzelknoten auf dem Level 0. Kinderknoten befinden sich ein Level höher als ihr Elternknoten. Auf einem Level l existieren maximal m^l Knoten, welche von links nach rechts sortiert sind. Der Knoten ganz links hat die Nummer 0, der Knoten ganz rechts hat die Nummer $m^l - 1$.

Jeder Peer hat lediglich eine lokale Sicht auf einige wenige andere Knoten im Netzwerk, sodass die folgenden Verbindungen/Informationen zu anderen Teilnehmern verwaltet werden:

- eine Verbindung (engl.: *link*) zu seinem Elternknoten (engl.: *parent*), außer es ist der Wurzelknoten (engl.: *root*);
- Verbindungen bis zu m Kinderknoten (engl.: *children*);
- eine Verbindung zu einem linken und zu einem rechten Adjazenten;
- Verbindungen zu bestimmten Nachbarn in dedizierten Tabellen. In der linke Routingtabelle (LRT) eines Knotens p_i existieren Verbindungen zu Nachbarn, welche sich links von ihm befinden. In der rechte Routingtabelle (RRT) eines Knotens p_i existieren Verbindungen zu Nachbarn, welche sich rechts von ihm befinden. Hat ein Peer p_i eine Verbindung zu einem Peer p_j auf einem Level l, so besteht auch eine Verbindung von p_j nach p_i (vgl. [JOT+06, Abschn. III-2]). Folglich sind diese Verbindungen bidirektional. Für einen Knoten p mit der Nummer n auf einem Level l existieren Einträge in der RRT mit den Distanzen:

  $$RRT(p) = \{(p_n + d \cdot m^i) < m^l \mid d = 1 \ldots m-1, i \geq 0\} \tag{5.2}$$

  Für denselben Knoten p mit der Nummer n auf demselben Level l existieren Einträge in der LRT mit den Distanzen:

  $$LRT(p) = \{(p_n - d \cdot m^i) \geq 0 \mid d = 1 \ldots m-1, i \geq 0\} \tag{5.3}$$

  Die Routingtabelle (RT) ist die Vereinigung der LRT und der RRT, daher gilt $\text{RT} = \text{LRT} \cup \text{RRT}$.
  Somit existieren für einen Knoten p mit der Nummer n auf einem Level l Knoten mit dem gleichen Abstand in beiden Richtungen mit:

  $$RT(p) = \{d \cdot m^i \mid d = 1 \ldots m-1, i \geq 0\} \tag{5.4}$$

  Der Aufbau der Routingtabellen in einem Netzwerk mit dem Fanout $m = 2$ ähnelt dem Aufbau der *Fingertables* in Chord (vgl. [SMK+01, Abschn. 4.3]).
- Links zu den Kindern zu jedem Eintrag in der RT. Diese werden auch Nachbarskinder (engl.: *Routing Table Children (RTC)*) genannt. Diese Verbindungen sind unidirektional, sodass nur ein Knoten p Informationen die Kinder der Einträge in der RRT bzw. der LRT besitzt.

Die maximale Anzahl an Links in einer RT ist nach oben hin beschränkt mit $|RT| = m \cdot l$ [JOT+06, Abschn. III-2]. Da jeder Nachbar in der RT Platz für maximal m Kinderknoten hat, ist die Anzahl an Einträgen auf einem Level l in der RTC nach oben auf $m \cdot |RT(p)| = m \cdot (m \cdot l) = (m^2) \cdot l$ beschränkt. Die Höhe eines nullbalancierten, vollständigen Baumes ist gegeben durch [Cha12]:

$$h(N_m(n)) = \lceil \log_m(N(m-1)+1) \rceil \tag{5.5}$$

Die lokale Sicht eines Knotens in Bezug auf seine Verbindungen sind in der nachfolgenden Abb. 5.5 dargestellt. Für den Knoten $p_{4:8}$ auf Level 4 ergeben sich in der RRT nach Gl. 5.2 die Distanzen zu den Nachbarn $1 \cdot 2^0$, $1 \cdot 2^1$ und $1 \cdot 2^2$ bzw. Verbindungen zu den Knoten (*4:9*), (*4:10*) und (*4:12*). In der LRT existieren Einträge zu den Nachbarn nach Gl. 5.3 mit den Distanzen $1 \cdot 2^0$, $1 \cdot 2^1$, $1 \cdot 2^2$ und

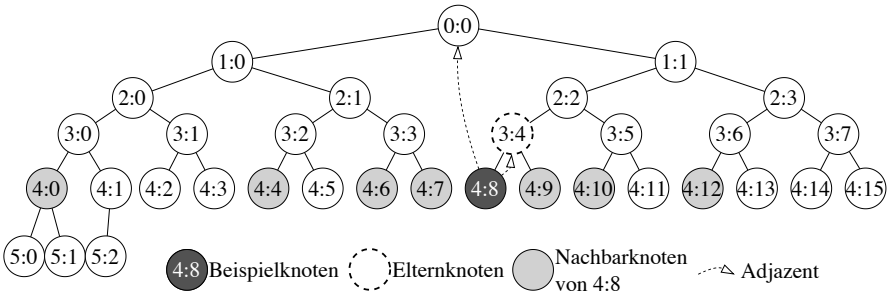


Abb. 5.5: Beispiel für einen MINHTON-Baum mit insgesamt 34 Knoten und Fanout $m = 2$. Dieser hat eine absolute Höhe von $h = 6$. Der Beispielknoten (*4:8*) hat eine Verbindung zu dem Knoten (*4:0*), sodass auch Informationen zu dessen Kindern (*5:0*) und (*5:1*) vorgehalten werden.

$1 \cdot 2^3$, folglich zu Knoten (*4:7*), (*4:6*), (*4:4*) und (*4:0*). Der Wurzelknoten befindet sich auf Level 0, Knoten auf dem höchsten Level l befinden sich auf $h-1$. Das Beispiel in Abb. 5.5 hat eine Höhe $h = 6$ und das höchste Level entspricht $l = 5$. Nur Knoten auf dem Level $l = 4$ dürfen neue Knoten aufnehmen. Da kein Knoten in einem P2P-Netzwerk eine globale Sicht hat, müssen hierfür Informationen über mehrere Knoten im Baum aggregiert werden, damit sichergestellt werden kann, dass der Baum auf jedem Level l von links nach rechts aufgebaut wird. In Tabelle 5.1 ist eine Übersicht der verwendeten Notation dargestellt, die nachfolgend verwendet wird.

Tabelle 5.1: Übersicht der Notation, die innerhalb des Management Overlays verwendet wird

Symbol	**Beschreibung**
$m \in \mathbb{N}^{\geq 2}$	Fanout des Baumes
$N_m \in \mathbb{N}$	Anzahl der Knoten im gesamten Netzwerk mit dem Fanout m
$N_{m,l} \in \mathbb{N}$	Anzahl der Knoten auf einem Level l
$V_i \in \mathbb{N}$	Anzahl der Verbindungen für einen Peer p_i
$l \in \mathbb{N}$	Level im Baum
$F_l(N) \in \mathbb{N}$	Freie Plätze auf einem Level l
$F(N) \in \mathbb{N}$	Freie Plätze im gesamten Baum bei einer Baumgröße von N
$h \in \mathbb{N}$	Höhe des Baumes
p	Peer p
$p_{l:n}$	Peer p auf Level l mit der Nummer n

5.3 Auf- und Abbau des Management Overlays

Soll ein neuer Knoten dem Netzwerk beitreten, so wird durch diesen ein freier Platz besetzt, allerdings werden m neue Plätze im Netzwerk frei. Demnach würden so viele freie Plätze im Netzwerk hinzukommen:

$$F(N) = F(N-1) + m - 1$$

Aufgrund der Definition für einen nullbalancierten, vollständigen Baum wird ein Level l vollständig von links nach rechts aufgefüllt. Ein neues Level wird erst eröffnet, wenn ein perfekter Baum existiert. Daher existieren für einen Knoten, welcher dem Netzwerk beitragen möchte, exakt $F(N) = 1$ mögliche Plätze im Baum (vgl. Abb. 5.4). Damit ein neuer Knoten ins Netzwerk aufgenommen werden kann, muss eine Vorbedingung erfüllt sein. Basierend auf der Definition für einen nullbalancierten, vollständigen Baum (vgl. S. 104) hat jeder Knoten einen linken Nachbarn. Ausgenommen hiervon sind Knoten mit der Position 0 auf einem Level l und der Wurzelknoten auf dem Level 0.

Zusätzlich wird ein neues Level nur betreten, wenn das zu dem Zeitpunkt höchste Level l keine Kapazität für neue Knoten hat. Daher befinden sich die Nullknoten immer auf dem höchsten Level $h-1$ oder $h-2$, sodass ein vollständiger Baum auch ein nullbalancierter Baum ist. Im Algorithmus 1 ist das Verfahren zur Erstellung des nullbalancierten, vollständigen Baumes dargestellt. Dieser sieht vor, dass bis zu der Position im Baum gesprungen wird, dessen rechte RT keine Einträge enthält. Der Suchvorgang zur Bestimmung dieser Position besteht aus den drei Phasen Springe zu einem Nullknoten (Z. 3–7), Bestimmung des Levels (Z. 8–32) und Springe zum letzten Knoten auf dem Level (Z. 33–36), welche nun nachfolgend erklärt werden.

Springe zu einem Nullknoten

Der Sprung zu einem Nullknoten (vgl. Z. 3–7 Algorithmus 1) erfolgt durch die Ausnutzung der RT-Struktur in MINHTON. Hierfür ist in Abb. 5.6 ein Beispiel für $m = 3$ dargestellt, bestehend aus den Eltern-Kind-Verbindungen (vgl. Abb. 5.6a), den Adjazenten (vgl. Abb. 5.6b) und einer Projektion aller Knoten auf eine 1-dimensionale Achse (vgl. Abb. 5.6c bzw. Abb. 5.6d).

Der Sprung zu einem Nullknoten sorgt dafür, dass sich im Anschluss die Anfrage entweder auf $h-1$ oder auf $h-2$ befindet. Ausgehend von Knoten (*1:0*) erfolgt der Sprung auf Level 2; ausgehend von Knoten (*2:0*) erfolgt der Sprung über die Adjazenten auf Level 3. Sobald ein Blattknoten erreicht wurde, wird mit der Bestimmung des Levels fortgefahren.

Algorithmus 1: NodeJoin(joinMessage msg)

```
1  bool acceptChild = False;
2  switch msg.procedureProgress do
3      case Reach a null node do // part_1
4          if Not a null node then
5              forwardToAdjacentNode();
6              break;
7          end
8      case Determine the level do // part_2
9          if First node x of part_2 then
10             if hasChildrenInLeftRT() then
11                 if isCorrectParent() then
12                     acceptChild = True;
13                 else if leftmostNeighborHasChildren() then
14                     searchEndOnLevel();
15                     acceptChild = False;
16                 end
17             else
18                 if rightmostNeighborDoesNotExist() then
19                     forwardRequest(parent, searchRight);
20                 else if rightmostNeighborOrXIsAtLevelEdge() then
21                     if isCorrectParent() then
22                         acceptChild = True;
23                     else
24                         searchEndOnLevel();
25                     end
26                 else
27                     forwardRequest(rightmostNeighbor, checkRight);
28                 end
29         else
30             checkLevelCapacity();
31         end
32         break;
33     case Navigate to the end do // part_3
34         if !(acceptChild = isCorrectParent()) then
35             searchEndOnLevel();
36         end
37 end
38 if acceptChild then
39     performAcceptChild();
40 end
```

Bestimmung des Levels

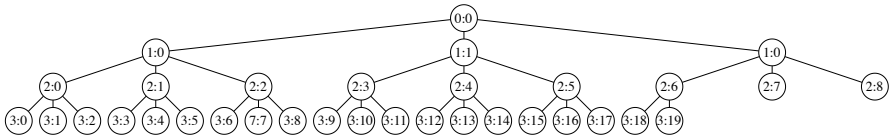
Nachdem ein Knoten auf Level $h-1$ oder $h-2$ erreicht wurde, muss nun das entsprechende Level bestimmt werden. Unter Zuhilfenahme folgender Fallannahmen kann ein Knoten x so das Level bestimmen:

Fall 1. $\exists$ ein Kindknoten k im ersten Knoten in der linken RT, so handelt es sich um das Level $h-2$:

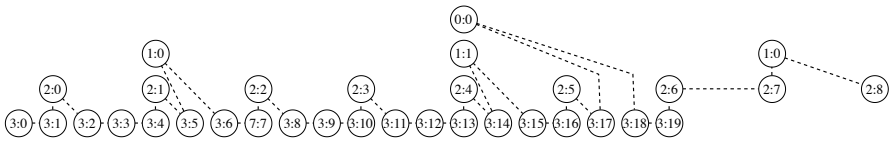
a. Der direkte linke Nachbar hat keine Kapazität für Kinder (kein Nullknoten).
b. Der direkte linke Nachbar hat Kapazität für weitere Kinder (Nullknoten).

Fall 2. $\nexists$ Kinderknoten k im ersten Knoten in der linken RT von x

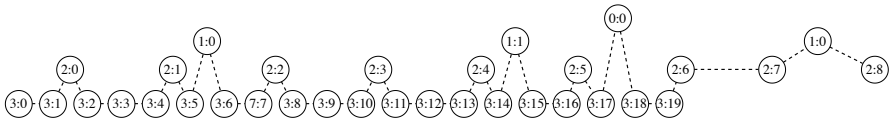
a. Ist das aktuelle Level von x komplett gefüllt, so handelt es sich um Level $h-1$ bzw. Level h bei einem perfekten Baum.



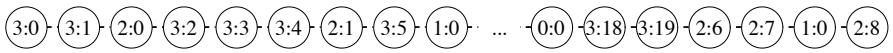
(a) Darstellung der Eltern-Kind-Verbindungen



(b) Darstellung der Adjazent-Verbindungen



(c) Nach links verschobene Teilbäume, zur Einsortierung der Adjazent-Verbindungen



(d) Lineare Projektion der Knoten auf eine 1-dimensionale Achse mit Darstellung der Adjazent-Verbindungen

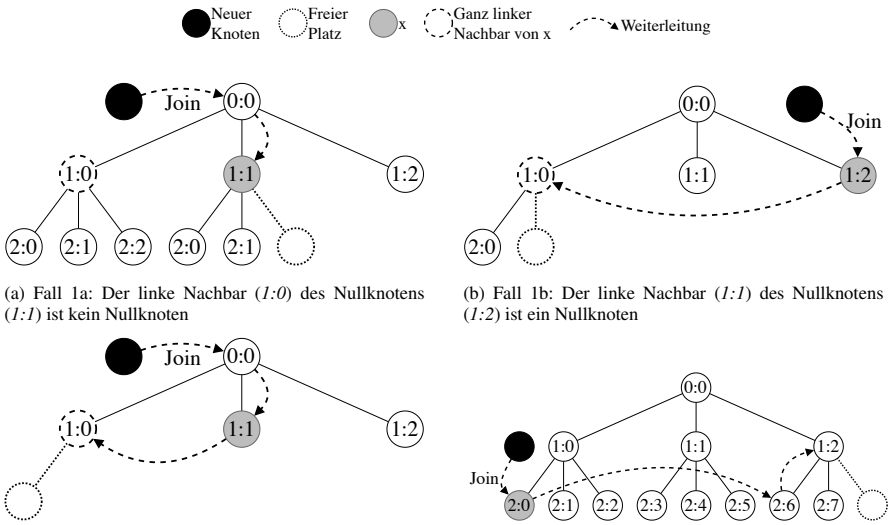
Abb. 5.6: Unterschiedliche Darstellungen derselben vollständig nullbalancierten Baumstruktur mit ungeradem Fanout ($m = 3$). Die Verbindungen der Adjazenten sind schwarz gestrichelt.

b. Ist das aktuelle Level von x nicht komplett gefüllt, handelt es sich um Level $h-1$.

Eine logische Vollständigkeit beim Aufbau von MINHTON ist gegeben durch die gezeigte Fallunterscheidung, da die disjunkten Fälle 1a, 1b, 2a und 2b alle Möglichkeiten abdecken. Ferner ergibt sich die Disjunktion aus der logischen Umkehrung von Fall 1 zu Fall 2 und den Teilfällen a) und b). Als Ausnahme sind die Fälle anzusehen, in denen die linke RT eines Knotens leer ist. Dies ist lediglich der Fall, wenn der Knoten an Position 0 auf einem Level ist. Hierfür wird allerdings direkt nach Fall 2 entschieden, um das Level bestimmen zu können.

In der Abb. 5.7 sind die unterschiedlichen Fälle bei der Bestimmung des Levels für einen Knoten x dargestellt. Für den Fall 1a (vgl. Abb. 5.7a) kann der Knoten x (*1:1*) die Aufnahme des neuen Knotens als ein Kind direkt akzeptieren. Im Fall 1b (vgl. Abb. 5.7b) weiß der Knoten x (*1:2*), dass dieser sich auf Level $h-2$ befindet, da sein linker Nachbar (*1:0*) noch einen Platz für einen Knoten frei hat. In Abb. 5.7c hat der Knoten x (*1:1*) lediglich Wissen über die Existenz von (1:2) und das Fehlen von (*2:0*). Dies reicht aus, um das Level als $h-1$ zu bestimmen.

Wie in Abb. 5.7d werden auch Weiterleitungen zur Bestimmung des Levels benötigt. Diese erfolgt durch einen Sprung zum am weitesten entfernten existierenden Eintrag in der rechten RT eines Knotens. Dies ist auch die Grundlage für die Bestimmung zum letzten existierenden Knoten auf dem gleichen Level.



(a) Fall 1a: Der linke Nachbar (*1:0*) des Nullknotens (*1:1*) ist kein Nullknoten

(b) Fall 1b: Der linke Nachbar (*1:1*) des Nullknotens (*1:2*) ist ein Nullknoten

(c) Fall 2a: Der erste Nullknoten (*1:1*) hat in seiner RT einen Eintrag zu (*1:2*), daher wird angenommen, dass das Level $l = h - 1$ entspricht

(d) Fall 2b: Der erste Nullknoten (*2:0*) leitet die Anfrage weiter an (*2:6*), da das Level $l = h - 1$ entspricht

Abb. 5.7: Beispiele für die Bestimmung des Levels beim Eintreten eines Knotens in MINHTON mit $m = 3$

Springe zu einem letzten Knoten auf dem Level *l*

Die dritte Phase des Algorithmus 1 besteht aus den verbleibenden – meist horizontalen – Routing-Sprüngen, um eine gültige Einfügeposition zu erreichen. In Abb. 5.7d wird vom Knoten x (*2:0*) die Anfrage an Knoten (*2:6*) weitergeleitet. Diese erfolgt durch einen Sprung zum am weitesten entfernten existierenden Eintrag in der rechten RT eines Knotens. Ein Sprung auf ein höheres Level, hier $h - 2$, erfolgt dann, wenn ein Knoten seinen finalen Platz auf dem Level $h - 1$ gefunden hat, sodass die Anfrage an den Elternknoten (*1:2*) weitergeleitet wird.

Bevor allerdings ein Knoten k dem Netzwerk beitreten kann, so muss der Elternknoten p zuvor seinen ersten linken und rechten Nachbarn temporär darüber informieren, dass nun ein neuer Knoten dem Netzwerk beitritt. Existiert in dem Elternknoten p kein rechter Nachbar, weil dieser der Knoten mit der höchsten Position auf dem aktuellen Level ist, so sperrt dieser den ersten Knoten auf dem nächsten Level. Dies ist der Tatsache geschuldet, dass es sonst mit einer konkurrierenden Anfrage zum Beitritt des Netzwerk kollidieren kann. Daher kann es ggf. möglich sein, dass ein neuer Knoten k kurzfristig abgelehnt wird und dieser eine neue Anfrage senden muss. Nachdem der neue Knoten k dem Netzwerk beigetreten ist, kann der Elternknoten p und dessen gesperrten Nachbarknoten wieder entsperrt werden.

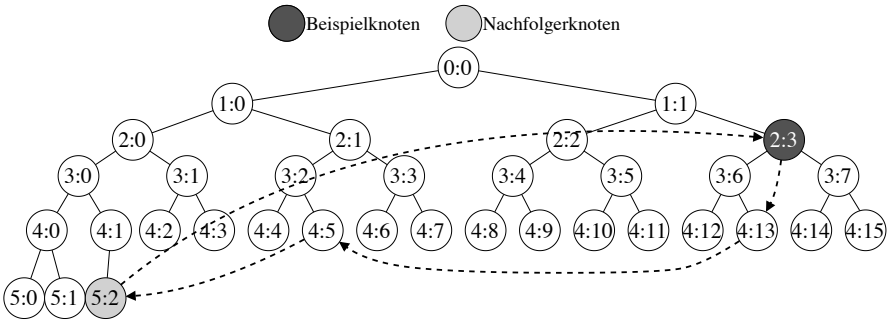
Beim Aufbau des Netzwerks wird das Kriterium des nullbalancierten, vollständigen Baumes eingehalten. Bevor ein Knoten v das Netzwerk allerdings verlassen

kann, muss ein Nachfolgerknoten s gefunden werden, sodass das Kriterium nicht verletzt wird. Daher ist das Verfahren zur Findung des Nachfolgers äquivalent zum Finden des nächsten freien Platzes für einen Knoten, welcher dem Netzwerk beitreten will. Allerdings werden noch zwei Operationen auf die unmittelbare Nachbarschaft angewendet:

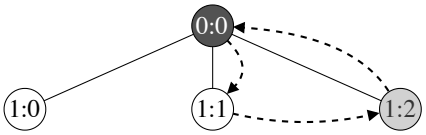
- **Prüfung, ob ein Nachfolger benötigt wird**: Der verlassende Knoten v prüft, ob ein Nachfolgerknoten s benötigt wird. Wenn die RRT von v leer ist, also $\mathrm{RRT}(v) = \emptyset$, so ist dies der letzte Knoten und es wird kein Nachfolgerknoten s benötigt, andernfalls wird der Nachfolgerknoten s im Netzwerk gesucht.
- **Sperrung des Elternknotens vom Nachfolgerknotens**: Sobald der letzte Knoten s die Anfrage bekommt, den Knoten v zu ersetzen, so sperrt s seinen Elternknoten p_s. Verlässt bereits ein Kindsknoten von p_s gerade das Netzwerks, weil es einen anderen Knoten ersetzt, so wird die Anfrage abgelehnt. Andernfalls sperrt p_s sperrt seinen ersten rechten und ersten linken Nachbarn von sich aus. Ist p_s der Knoten mit der höchsten Position auf dem Level, so wird der erste Knoten auf dem nächsten Level ebenfalls gesperrt. Der Elternknoten p_s entfernt die Routinginformationen von v und informiert alle Nachbarn in der RRT und LRT, dass diese s aus ihrer RT entfernen sollen. Sobald p_s von allen Nachbarn die Bestätigung erhalten hat, kann s die Position verlassen und die Position von v übernehmen.
- **Abmeldung des Nachfolgerknotens**: Nachdem s von p_s die Bestätigung erhalten hat, die Position verlassen zu dürfen, so meldet sich der Knoten s von seinen Nachbarn in der RRT und LRT incl. der Adjazenten ab. Nach Erhalt der Bestätigung der Abmeldung von den Nachbarn und der Adjazenten kann sich s dem Knoten v anbieten.
- **Übernahme der neuen Position**: Der verlassende Knoten v bestätigt s, dass dieser sein Nachfolgeknoten sein darf. Hierbei übermittelt v auch s seine RT. Der Nachfolgeknoten s wechselt an die Position von v und meldet sich bei seinen neuen Nachbarn in der RT incl. Kinder und Adjazenten an. Hierauf wartet s, welcher nun an der Position von v ist, auf die Bestätigung der Knoten.
- **Entsperrung der Knoten**: Sobald s, welcher nun an der Position von v ist, alle Bestätigungen erhalten hat, so informiert s seinen vorherigen Elternknoten p_s darüber. Folglich kann p_s seinen seine linken und rechten Nachbarn entsperren.

In Abb. 5.8 sind verschiedene Szenarien dargestellt, bei denen ein Knoten das Netzwerk verlassen will. Im ersten Szenario (vgl. Abb. 5.8a) will der Knoten (*2:3*) das Netzwerk verlassen und sendet über seinen linken Adjazenten die Anfrage an einen Nullknoten (*4:13*). Der Nullknoten (*4:13*) versucht nun, das Level zu bestimmen und sieht, dass von ihm rechts aus noch die Knoten (*4:14*) und (*4:15*) in seiner RT existieren. Zusätzlich sieht Knoten (*4:13*), dass kein Knoten in seiner LRT einen Eintrag zu seinen Kindern hat. Daher ist eine Bestimmung des Levels noch immer nicht möglich, ob es sich um Level $h-1$ oder $h-2$ handelt. Die Anfrage wird folglich nach links an den entferntesten Eintrag – Knoten (*4:5*) – in seiner LRT gesendet werden. Der Knoten (*4:5*) hat aufgrund seiner Verbindung zu Knoten (*4:1*) auch einen unidirektionalen Eintrag zu Knoten (*5:2*). Mit Knoten (*5:2*) handelt es

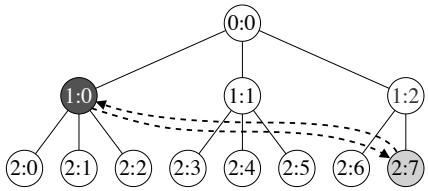
sich hierbei um den Nachfolgerknoten. Der Knoten (*5:2*) bei seinem Elternknoten (*4:1*) anfragen, ob dieser die Position verlassen darf, oder ob bereits ein anderer Kinderknoten von (*4:1*) gerade die Position bzw. das Netzwerk verlassen will. Angenommen der Knoten (*5:2*) ist der einzige, so sperrt der Elternknoten (*5:2*) seine direkten Nachbarn (*4:0*) und (*4:2*). Nachdem der Elternknoten (*4:1*) die Bestätigung der Sperrung seiner ersten Nachbarn erhalten hat, wird der Knoten (*5:2*) darüber informiert, dass dieser sich bei seinen Nachbarn abmelden darf. Der Knoten (*5:2*) meldet sich bei seinen Nachbarn in der LRT ab und bietet sich dem Knoten (*2:3*) an. Wenn der Knoten (*2:3*) das Angebot annimmt, so wird hierbei die RT von Knoten (*2:3*) an Knoten (*5:2*) übermittelt. Im Anschluss kann der neue Knoten (*2:3*) sich bei seinen neuen Nachbarn anmelden und bei seinem altern Elternknoten (*4:1*) aufheben. Der ehemalige Elternknoten von (*5:2*), Knoten (*4:1*), kann nun seinen linken und rechten Nachbarn entsperren.



(a) Knoten (*2:3*) sendet die Anfrage an seinen Adjazenten (*4:13*). Dieser leitet die Anfrage an den Knoten (*4:5*) aus seiner LRT weiter. Knoten (*4:5*) sieht den fehlenden Eintrag in seiner linken RTC, sodass der Knoten (*5:2*) als Nachfolger bestimmt ist



(b) Knoten (*0:0*) sendet seine Anfrage an seinen Adjazenten (*1:1*). Knoten (*1:1*) kann mithilfe der RT den Knoten (*1:2*) als Nachfolger bestimmen



(c) Knoten (*1:0*) sieht über seine RTC einen fehlenden Eintrag im Knoten (*1:2*), sodass Knoten (*2:7*) als Nachfolger bestimmt ist

Abb. 5.8: Beispiele für die Bestimmung eines Nachfolger-Knotens für Netzwerke mit Fanout $m = 2$ und $m = 3$

In Abb. 5.8b will der Wurzelknoten (*0:0*) das Netzwerk verlassen und sendet seine Anfrage an seinen Adjazenten (*1:1*). Dieser Blattknoten kann direkt das Level bestimmen und den Knoten (*1:2*) als Nachfolger identifizieren. Gemäß dem hier eingeführten Verfahrens würde der Elternknoten (*0:0*) für den Vorgang den ersten Knoten auf dem nächsten Level, hier (*1:0*) temporär sperren, bis der Knoten (*1:2*) die Position von (*0:0*) übernommen hat. In Abb. 5.8c hingegen wird nicht der Adjazente

verwendet, um die Anfrage an einen Nullknoten zu senden. Da Knoten (*1:0*) Informationen über die Kinderknoten von (*1:2*) in seiner RTC vorhält, weiß der Knoten (*1:0*) direkt, dass es sich bei dem Knoten (*2:7*) um seinen Nachfolger handelt. Der Elternknoten von (*2:7*) würde hier temporär sperren seinen linken Nachbarn (*1:1*) und den ersten Knoten auf dem nächsten Level (*2:0*) während des Vorgangs sperren.

Mit der Sperrung eines Elternknotens p und dessen ersten linken und rechten Nachbarn wird vermieden, dass sich zwei oder mehr Operationen (z. B. Beitritt-Betritt oder Beitritt-Verlassen) nicht überschneiden. Somit werden konkurrierende Operationen sequenziell abgearbeitet. Dadurch wird zwar das Netzwerk hinsichtlich des Auf- und Abbaus langsamer, aber es wird sichergestellt, dass keine Routinginformationen in den Knoten mit fehlerhaften bzw. falschen Informationen be- bzw. überschrieben werden.

5.3.1 Suchen nach anderen Knoten anhand der logischen ID

Um innerhalb des Baumes nach anderen Knoten zu suchen, wird eine möglichst effiziente Suchmethode (engl.: *SearchExact*) benötigt. Effizienz ist hier dadurch definiert, dass die Anzahl an Sprüngen vom Ursprungsknoten zum Zielknoten minimal ist. Da auch hier die Knoten nur eine lokale Sicht auf ihre Nachbarn haben, besteht ein Ansatz darin, so viele Informationen eines Knoten wie möglich zu berücksichtigen. Daher werden die MINHTON-Strukturmerkmale und ein *TreeMapper* für die verschiedenen Level des Baumes verwendet.

Der *TreeMapper* bildet jeden Knoten des Baumes auf einer 1-dimensionalen Achse ab. Dadurch kann ein Knoten lokal die relative horizontale Entfernung zu jedem anderen Knoten annähern, unabhängig davon, ob dieser existiert oder nicht und ohne zusätzliche Nachrichten durchs Netzwerk zu schicken. Diese Information der Annäherung wird in *SearchExact* verwendet, um den nächsten Empfänger einer Anfrage effizient zu bestimmen. Darüber hinaus kann außerdem die Gesamtzahl der Routing-Schritte – auch Routing-Hops genannt – zum gewünschten Ziel sogar angenähert werden. Die Funktion (vgl. Gl. 5.9), welche die Position eines Knotens auf der 1-dimensionalen Achse bestimmt, ist gegeben durch

$$tm_{k,m}(l,n) = b^{low}_{k,m}(l,n) + w_{k,m}(l,n) \cdot \lceil \frac{m}{2} \rceil \tag{5.6}$$

wobei l das Level ist, n die Nummer eines Knotens, m der Fanout des m-ary Baumes und $k \in \mathbb{R}^+$ ein konstanter Wert, der dem Wurzelknoten zugeordnet ist. Der tatsächliche Wert von k kann zufällig gewählt werden und wird während der Verwendung des Baumes nicht mehr verändert. Die Funktionen $w_{k,m}(l,n)$ und $b^{low}_{k,m}(l,n)$ sind in (5.9) bzw. (5.10) angegeben. Ein Beispiel für die 1-dimensionale Projektion von Knoten ist Abb. 5.9 dargestellt, für einen Baum mit $m = 2$ und $k = 100$.

Die Bedingung für die Gültigkeit des *TreeMapper* ist, dass der Wert eines linken Adjazenten eines Knotens immer kleiner sein muss als der Wert des Knotens selbst. Als Konsequenz muss der Wert eines rechten Adjazenten immer größer sein.

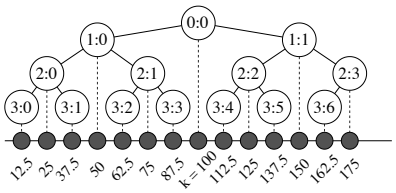


Abb. 5.9: Annäherung der Distanz von Knoten durch eine Projektion der Positionen aller Knoten eines Baumes mit Fanout $m = 2$ auf eine 1-dimensionale Achse mit dem initialen Wert für den Wurzelknoten $k = 100$

Gleichzeitig darf es keinen anderen Knoten im Baum geben, welcher einen Wert dazwischen besitzt. Somit ergibt sich eine streng monoton steigende Funktion, wenn die Reihenfolge der Knoten in einer sortierten, aufsteigenden Ordnung existiert. Abhängig vom Wert des Elternknotens und von der relativen Position des Knotens im Vergleich zu seinen Nachbarknoten wird der projizierte Wert eines Knotens basierend auf Gl. (5.6) bestimmt. Der *TreeMapper* verwendet das Konzept der unteren und oberen Schranken. Alle Knotenwerte innerhalb eines Teilbaumes werden durch die untere und obere Grenze der Wurzel des Teilbaumes begrenzt, während die Schranken selbst nie erreicht werden. Der äußerste linke und der äußerste rechte Knoten innerhalb des Teilbaumes auf einer beliebig hohen Ebene konvergieren nur gegen diese Schranken. Gl. (5.7) und Gl. (5.8) werden als Abbruchbedingungen für die Rekursion verwendet, sofern es sich um den Wurzelknoten handelt.

$$b^{low}_{k,m}(0,0) = 0 \tag{5.7}$$

$$b^{up}_{k,m}(0,0) = \frac{k}{\lceil \frac{m}{2} \rceil} \cdot m \tag{5.8}$$

Um die untere und obere Grenze anderer Knoten zu berechnen, muss zuerst die untere und obere Grenze des übergeordneten Knotens gemäß Gl. (5.10) und (5.11) berechnet werden. Hier wird die Hilfsfunktion (5.9) verwendet, um die Schrittweite zwischen den Kindern eines Elternknotens zu berechnen. l_p und n_p bezeichnen dabei das Level und die Nummer des Elternknotens, die sich als $l_p = l - 1$ und $n_p = \frac{n-(n \mod m)}{m}$ berechnen lassen.

$$w_{k,m}(l,n) = \frac{b^{up}_{k,m}(l_p,n_p) - b^{low}_{k,m}(l_p,n_p)}{m} \tag{5.9}$$

$$b^{low}_{k,m}(l,n) = b^{low}_{k,m}(l_p,n_p) + w_{k,m}(l,n) \cdot (n \mod m) \tag{5.10}$$

$$b^{up}_{k,m}(l,n) = b^{low}_{k,m}(l,n) + w_{k,m}(l,n) \tag{5.11}$$

Nachdem die Grenzen berechnet wurden, gibt der *TreeMapper* einen zentrierten Wert innerhalb der Grenzen zurück. Dieser hängt von dem Fanout m ab. Bei geraden Fanouts liegt er in der Mitte der Schranken. Bei ungeraden Fanouts ist die Mitte leicht nach rechts verschoben, weil es auf der linken Seite ein Kind mehr gibt als auf der rechten Seite. Der *TreeMapper* ist so konstruiert, dass ein Knoten den relativen Abstand zwischen beliebigen Knoten anhand ihrer horizontalen Distanz lokal auswerten kann. Anhand des Ergebnisses kann der Knoten einen (nahezu) optimalen Weg einschlagen. Die Rekursionstiefe entspricht im ungünstigsten Fall der maximalen Höhe des Baumes (vgl. Gl. 5.5).

In Algorithmus 2 ist der Pseudocode für *SearchExact* gelistet. Zunächst werden die horizontalen Positionen für den Knoten selbst und den Zielknoten berechnet. Hierfür werden der Fanout m und der Anfangswert für k für den *TreeMapper* benötigt. Sind die berechneten Werte identisch, so hat die Nachricht ihr Ziel erreicht. Andernfalls wird für jeden Eintrag in der RT der *TreeMapper*-Wert berechnet, um die neue horizontale Position für die Weiterleitung zu bestimmen.

Algorithmus 2: SearchExact(node target, query q)

```
1 selfValue = TreeMapper(self.level, self.number, m, k);
2 targetValue = TreeMapper(target.level, target.number, m, k);
3 if selfValue == targetValue then
4 |   execute search exact query q locally;
5 else
6 |   Forward target and q to closestNode;
7 |   SearchExact(closestNode, q);
8 end
```

Ziel ist es, den Abstand der horizontalen Position zwischen zwei Knoten zu verringern, insbesondere zwischen dem nächsten Knoten, an den die Anfrage weitergeleitet wird, und dem Zielknoten. Dieser Vorgang wird so lange wiederholt, bis der Abstand zwischen dem nächsten Sprung bzw. Hop-Knoten und dem Zielknoten $= 0$ beträgt. Dies ist der Fall, wenn der Zielknoten die *SearchExact*-Anfrage erhält. Die obere Grenze in MINHTON für die Suche in nach einem exakten Wert bleibt dieselbe wie in BATON*, nämlich $O(log_m N)$ [JOT+06].

5.3.2 Analytische Betrachtung des Auf- und Abbaus

Wie im vorherigen Abschnitt diskutiert, besteht das Beitreten bzw. das Verlassen eines Knotens aus drei Phasen. Diese werden nun im Folgenden hinsichtlich der Laufzeit betrachtet. Dabei muss neben der Positionsfindung auch das das Aktualisieren der Nachbarn berücksichtigt werden.

- **Springe zu einem Nullknoten**: Der Sprung zu einem Nullknoten erfolgt über einen Adjazenten. Dieser Sprung ist nur notwendig, wenn eine Anfrage von einem Knoten kommt, welcher nicht ein Nullknoten ist. Wie in Abb. 5.6 bereits

ersichtlich wurde, wird von einem beliebigen Knoten zu einem Nullknoten maximal 1 Sprung benötigt.

- **Bestimmung des Levels**: Nachdem ein Nullknoten mithilfe des Adjazenten-Sprungs erreicht wurde, muss nun bestimmt werden, ob es sich um Level $h-1$ oder $h-2$ handelt. Dies geschieht durch horizontale Sprünge unter Zuhilfenahme der RT.
  Als Worst Case kann hierbei angenommen werden, dass sich der Knoten am Ende des Levels befindet und zum ersten Knoten auf dem Level gesprungen werden muss, wie in einem perfekten Baum.
  In der nachfolgenden Tabelle 5.2 sind die Knoten aufgelistet, um von der ersten Position 0 auf einem Level l zum letzten Knoten an der Position $m^l - 1$ zu springen. Zusätzlich gilt die Annahme, dass immer der Knoten in der RRT genommen wird, welcher die weiteste Distanz hat zu sich selbst hat. Dieses Verfahren ist auch umkehrbar, um vom letzten Knoten zum ersten Knoten des Levels zu springen.
  In einem Netzwerk mit Fanout $m = 2$ auf Level 4 werden die Knoten $\{0,8,12,14,15\}$ benötigt, um vom ersten Knoten zum letzten Knoten zu springen; in einem Netzwerk mit Fanout $m = 3$ auf Level 3 werden entsprechend die Knoten $\{0,18,24,26\}$ benötigt. Diese Sprünge, ausgehend vom ersten Knoten auf dem Level, und die dabei verbleibenden Knoten bis zum letzten Knoten sind in Tabelle 5.3 dargestellt.
  Unter Verwendung der RRT und des Eintrags mit der weitesten Distanz können für den Fanout $m = 2$ auf Level 4 im ersten Sprung $\frac{N_{m,l}}{2} = 8$ Knoten übersprungen werden, sodass die gleiche Anzahl an Knoten verbleibt bis zum Ende des Levels. Dies kann wiederholt werden, sodass im nächsten Sprung $\frac{N_{m,l}}{4}$ Knoten verbleiben. Hieraus lässt sich das Muster ableiten, dass bei einem k-ten Sprung

Tabelle 5.2: Übersicht der rechten Routingtabelle (RRT) für Peers. Der folgende Knoten mit der weitesten Distanz wird als nächster Zielknoten genommen.

Peer	**Rechte Routingtabelle**	**Knoten mit der maximalen Distanz**
Fanout $m = 2$; Level 4; $N_{2,4} = 16$		
$p_{4:0}$	$\{1,2,4,8\}$	8
$p_{4:8}$	$\{9,10,12\}$	12
$p_{4:12}$	$\{13,14\}$	14
$p_{4:14}$	$\{15\}$	15
Fanout $m = 3$; Level 3; $N_{3,2} = 27$		
$p_{3:0}$	$\{1,2,3,6,9,18\}$	18
$p_{3:18}$	$\{19,20,21,24\}$	24
$p_{3:24}$	$\{25,26\}$	26

Tabelle 5.3: Exemplarische Darstellung der benötigten Sprünge, um vom ersten Knoten auf einem Level l zum letzten Knoten zu kommen

	$m=2, l=4$		$m=3, l=2$		…	m, l
Sprung	**Pos**	**VK**	**Pos**	**VK**	…	**VK**
-	0	16	0	27	…	$N_{m,l}$
1	8	8	18	9	…	$N_{m,l}/m^1$
2	12	4	24	3	…	$N_{m,l}/m^2$
3	14	1	26	0	…	$N_{m,l}/m^3$
4	15	0	-	-	…	$N_{m,l}/m^4$
⋮	-	-	-	-	⋮	⋮
k-ter	-	-	-	-	…	$N_{m,l}/m^k$

VK: Verbleibende Knoten bis zum Ende des Levels

immer $\frac{N_{m,l}}{2^k}$ Knoten überbleiben. Nach k Vergleichen bleibt kein Knoten mehr über, sodass der Zielknoten erreicht ist. Hieraus lässt sich, unabhängig vom Fanout, die benötigte Anzahl an Sprüngen vom ersten Knoten auf einem Level l zum letzten Knoten auf demselben Level l durch die folgende Gleichung herleiten:

$$\begin{aligned} 1 &= \frac{N_{m,l}}{m^{k-1}} \\ m^{k-1} &= N_{m,l} \\ k-1 &= \log_m(N_{m,l}) \\ k &= \log_m(N_{m,l})+1 \\ &= \log_m(N_{m,l}) \\ &= \log_m(m^l) \Rightarrow l \end{aligned} \tag{5.12}$$

Hieraus resultiert

$$hops_{\max}(l) = \begin{cases} 0, & l=0 \\ 1, & l=1 \\ 1+hops_{\max}(l-1), & l>1 \end{cases} \tag{5.13}$$

bzw. auch vereinfacht als $hops_{\max} = l$.
Somit werden l Sprünge auf einem Level l benötigt, um vom ersten zum letzten bzw. vom letzten zum ersten Knoten zu springen. Sobald ein Knoten den ersten Knoten seines Levels in seiner RT besitzt, kann das Level entsprechend mithilfe der RTC des ersten Knotens bestimmt werden.

- **Springe zum letzten Knoten auf dem Level** l: das Erreichen eines beliebigen anderen Knotens auf demselben Level l äquivalent zu der Bestimmung des Levels, sodass die Kosten ebenfalls mit $\log_m(N_{m,l})$ zu beziffern sind

Neben den Kosten für die Positionsfindung eines freien Platzes bzw. eines Nachfolgerknotens ($O(\log_m(N_{m,l}))$) sind noch die Kosten für die Aktualisierung der RT relevant ($O(m \cdot \log_m(N_{m,l}))$). Daher entsprechen die Kosten für den Aufbau bzw. Abbau des Baumes $O(\log_m(N_l)) + O(m \cdot \log_m(N_{m,l}))$. Nach Anwendung der Logarithmusgesetze entsprechen die Kosten für die Verwaltung des Baumes den Kosten der Aktualisierung der Routingtabellen ($O(m \cdot \log_m(N_{m,l}))$). In Tabelle 5.4 ist eine Übersicht der Laufzeiten einiger Extremfälle dargestellt.

Tabelle 5.4: Übersicht der (minimalen/maximalen) Laufzeiten für Extremfälle, unterteilt nach dem Vorkommen in der Positionsfindung vom *Join* (J) bzw. *Leave* (L) und/oder dem *Update* (U) der Nachbarn

Nr.	Extremfall	J/L/U	Min/Max	Laufzeit	Anmerkung
1	$N = 1$	J/U	Min	0	nur Wurzelknoten vorhanden
2	$N = 2$	L	Min	$[0,1]$	nur Wurzelknoten mit einem Kind vorhanden
3	Perfekter Baum	J	Max	$< 2\log_m N + 2$	Alle Nullknoten befinden sich auf dem höchsten Level $h-1$.
4		L	Max	$< 3\log_m N + 2$	
5		U	Min	$O(m \cdot \log_m N_{m,l})$	
6	$\lim_{m\to\infty} m$	J/L	Min	$[0,1]$	vollvermaschtes Netzwerk
7	$\lim_{m\to\infty} m$	U	Max	$O(N)$	vollvermaschtes Netzwerk
8	Knoten am Levelrand	U	Min	$O(m \cdot \log_m N_{m,l})$	Position eines Knotens auf einem Level l entspricht entweder 0 oder $m^l - 1$
9	Knoten mit stärkster Vernetzung	U	Max	$O(m \cdot \log_m N_{m,l})$	maximal viele verlinkte Nachbarn und Adjazenten

5.3.3 Simulative Evaluation

Ziel dieses Abschnitts ist es, das vorgeschlagene Management Overlay MINHTON anhand von Simulationsergebnissen zu verifizieren. Dabei soll das das untersuchte System quantitativ bewertet und die durchgeführte theoretische Analyse ergänzt werden. Die Simulationsexperimente wurden mit dem diskreten Ereignis-Netzwerksimulator ns-3 [RH10] durchgeführt. Zu den Merkmalen, die von Interesse sind, gehören der generierte Verkehr (Anzahl der ausgetauschten Nachrichten) zum Beitreten oder Verlassen des Netzwerks, die Verteilung der Last auf die teilnehmenden Peers, die Höhe des Baumes und die benötigte Anzahl an Sprüngen, um einen anderen Peer anhand seiner logischen ID mithilfe einer *SearchExact*-Anfrage zu finden. Es wurden Netzwerke mit unterschiedlichen Größen N von 1.000

bis 10.000 Knoten mit unterschiedlichen Fanouts $m \in \{2, 4, 8, 16\}$ simuliert. Für den *SearchExact*-Test wurden 1.000 Abfragen mit exakter Übereinstimmung ausgeführt und die durchschnittlichen Kosten der Operationen ermittelt. Jedes Experiment wurde 10-mal – jeweils mit verschiedenen Startwerten für einen Zufallsgenerator – durchgeführt und für die nachfolgenden Abbildungen gemittelt.

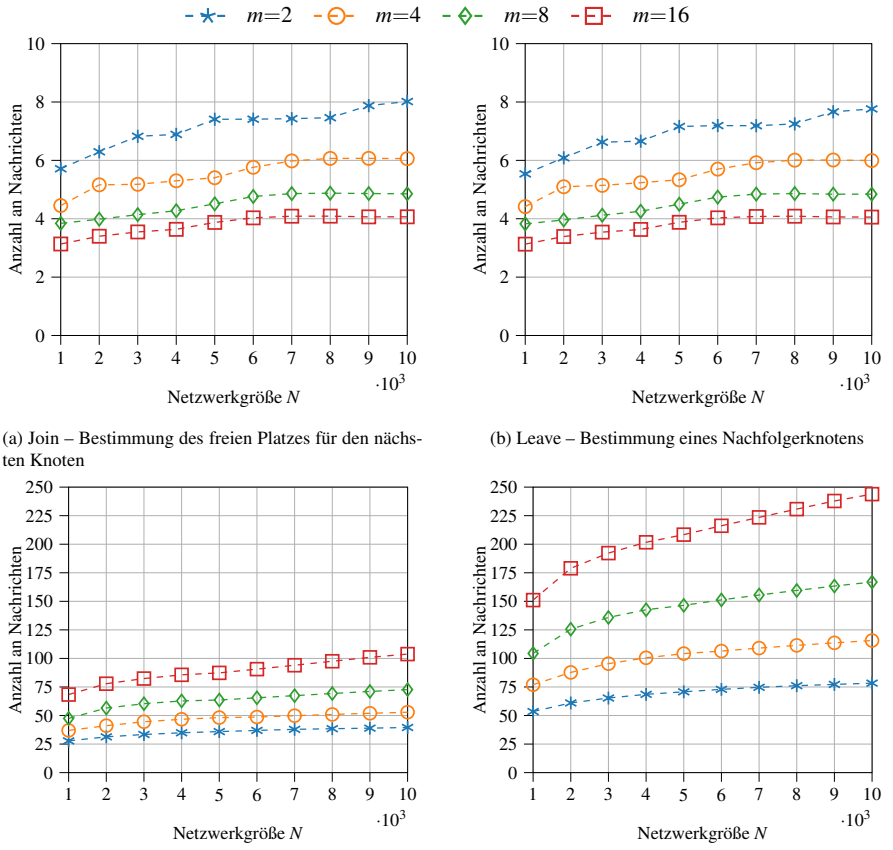
Netzwerkbeitritt (Join)

In Abb. 5.10a ist die durchschnittliche Anzahl an Nachrichten dargestellt, welche benötigt werden, um den nächsten freien Platz für einen Knoten zu finden, welcher dem Netzwerk beitreten will. Festzuhalten ist, dass, wenn der Fanout erhöht wird, sich auch die Vernetzung der einzelnen Knoten erhöht, was wiederum Einfluss auf die benötigten Routing-Hops hat. Die Vernetzung bezieht sich hier auf die Anzahl der Einträge innerhalb der RT und der RTC. Abb. 5.10c beinhaltet zusätzlich noch die dazu benötigten Nachrichten, um die Nachbarknoten über das Beitreten zu informieren. Als dominanter Faktor bei den Kosten ist die Aktualisierung der Routingtabellen anzusehen.

Netzwerkaustritt (Leave)

Soll bzw. will ein Knoten das Netzwerk verlassen, so muss zunächst ein Nachfolger bestimmt werden. Die dafür benötigte Anzahl an Nachrichten ist in Abb. 5.10b dargestellt. Bevor aber der Nachfolger die Position des Knotens übernehmen kann, welcher das Netzwerk verlassen will, muss dieser sich bei seinen Nachbarn abmelden. Entsprechend stellt Abb. 5.10d die Anzahl an Nachrichten zur Bestimmung des Nachfolgers dar, die Aktualisierung der Nachbarn für den Nachfolger und die Übernahme der Position des Knotens, welcher das Netzwerk verlässt. Da es sich hierbei um den letzten Knoten im Baum handelt, besitzt dieser keine Einträge in seiner LRT, sodass die Anzahl an Nachrichten zur Bestimmung des Nachfolgers und der Aktualisierung der Nachbarknoten $< 3 \cdot |\mathrm{RT}|$ entspricht.

In der Abb. 5.11 sind die Kosten dargestellt, die für den Aufbau eines Netzwerks mit jeweils 10.000 Knoten für unterschiedliche Fanouts aus Sicht eines Knoten benötigt werden. In Abb. 5.11a ist die benötigte Anzahl an Nachrichten zur Positionsfindung für den nächsten freien Platz dargestellt. Mit einem steigenden Fanout steigt auch die Vernetzung der Knoten, sodass die Anzahl an Nachrichten zur Findung des nächsten freien Platzes reduziert wird. Die Ausreißer, bspw. für $m = 2$ bei $N = 1.023$ bzw. $N = 2.047$, sind auf die Auslastung des Baumes zurückzuführen. Da es sich hier fast um einen perfekten Baum handelt, muss beim Betreten des Netzwerks die Höhe des Baums bestimmt werden, also ob es sich das Level $h-1$ bzw. $h-2$ handelt. Dies erfolgt durch eine Traversierung der entsprechenden Level und unter Auswertung der RT und der RTC.



(a) Join – Bestimmung des freien Platzes für den nächsten Knoten

(b) Leave – Bestimmung eines Nachfolgerknotens

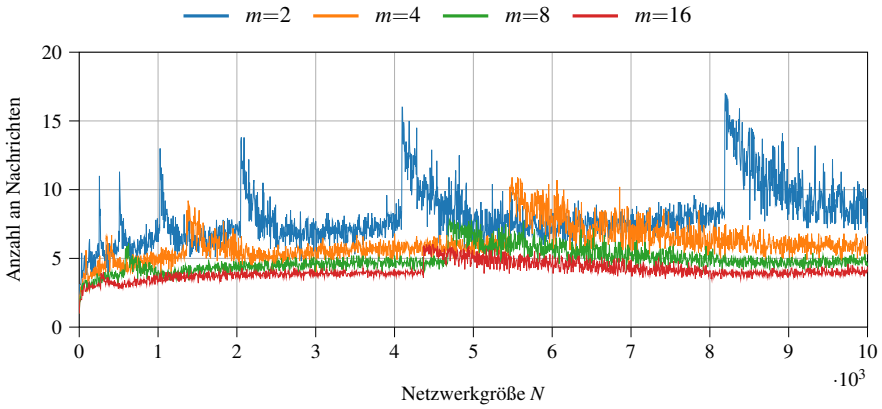
(c) Join – Bestimmung freier Platz für den nächsten Knoten und Aktualisierung der Nachbarknoten.

(d) Leave – Bestimmung eines Nachfolgers und Aktualisierung der Nachbarknoten.

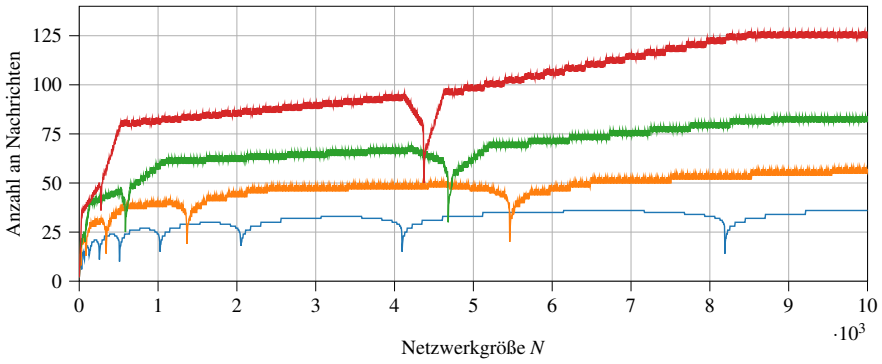
Abb. 5.10: Durchschnittlich benötigte Anzahl an Nachrichten für einen Knoten beim Betreten (Join) bzw. Verlassen (Leave) des Overlay-Netzwerks MINHTON in Abhängigkeit von der Netzwerkgröße und dem Fanout

In Abb. 5.11b sind neben den benötigten Sprüngen zur Findung des nächsten freien Platzes auch die ausgetauschten Nachrichten zur Aktualisierung der Nachbarn dargestellt.

Abb. 5.12 stellt neben der benötigten Höhe des Baumes auch die durchschnittliche Anzahl an symmetrischen Verbindungen dar. Die symmetrischen Verbindungen beinhalten Verbindungen zu dem Elternknoten, bis zu m Kindern, den Nachbarknoten in der RT und den Verbindungen zu den Adjazenten.



(a) Bestimmung freier Platz für den nächsten Knoten, welcher dem Netzwerk beitreten will



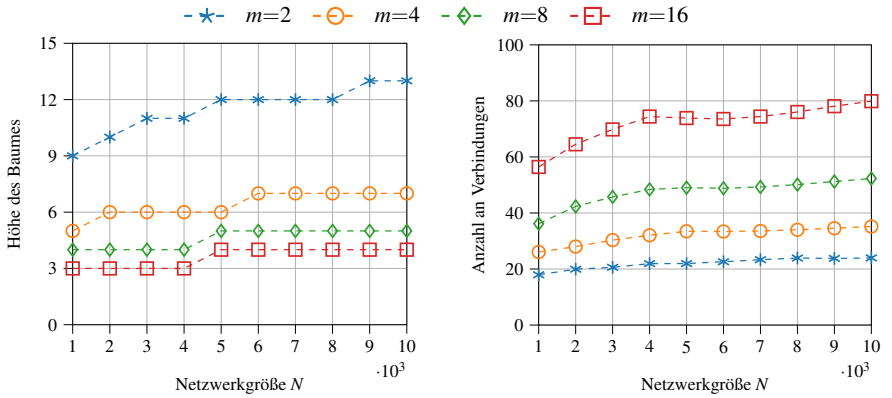
(b) Aktualisierung der Nachbarn

Abb. 5.11: Benötigte Nachrichten für den Aufbau von MINHTON aus Sicht eines Knotens, um ein Netzwerk der Größe von 10.000 Teilnehmern mit verschiedenen Fanouts aufzubauen

Suchen nach anderen Knoten

In Abb. 5.13 sind die durchschnittlich benötigten Sprünge (engl.: *routing hops*) für eine *SearchExact*-Anfrage dargestellt in Abhängigkeit von der Netzwerkgröße N und des Fanouts m. Eine *SearchExact*-Anfrage kann mittels logischen ID und oder eines beliebigen Wertes innerhalb des *TreeMappers* durchgeführt werden (vgl. Abschn. 5.2.1). In Abb. 5.13a sind die Kosten für BATON* dargestellt, entnommen aus [JOT+06]. Abb. 5.13b hingegen stellt die Simulationsergebnisse für die benötigte Anzahl an Sprüngen für MINHTON dar.

Wird der Fanout erhöht, werden die Kosten für eine *SearchExact*-Anfrage reduziert. Dies lässt sich auf die engere Vernetzung von MINHTON und der Definition eines null-balancierten, vollständigen Baumes. Basierend auf den Ergebnissen der Simulation belaufen sich die Kosten für eine *SearchExact*-Anfrage von MINHTON



(a) Die Höhe des Baumes

(b) Durchschnittliche Anzahl an symmetrischen Verbindungen für einen Knoten

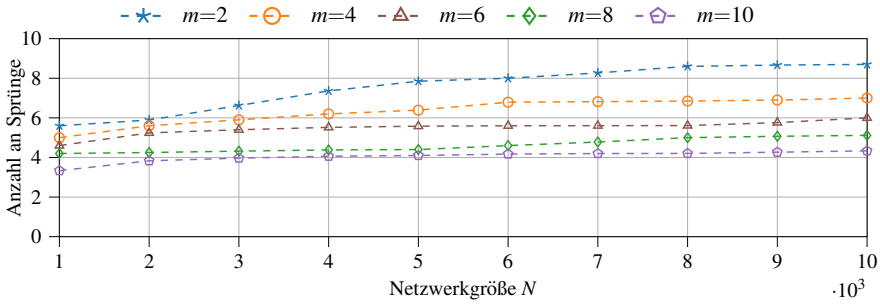
Abb. 5.12: Allgemeine Informationen über MINHTON, in Abhängigkeit von der Netzwerkgröße (N) und dem Fanout (m)

auf $O(\log_m N)$ begrenzt ist. Allerdings ist MINHTON im Vergleich zu BATON* in Bezug auf die benötigten Routing Hops um bis zu 50 % effizienter.

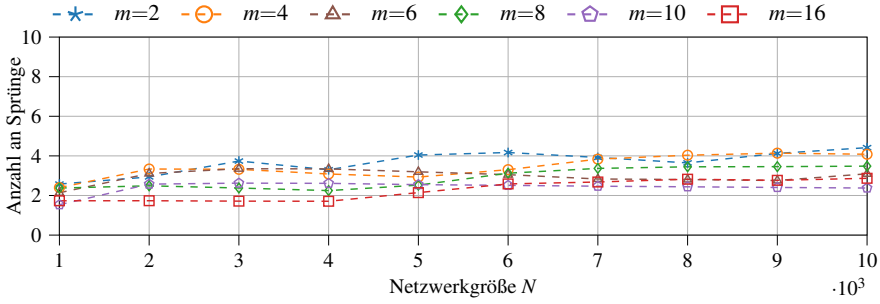
5.3.4 Zusammenfassung der Ergebnisse

Mit der Erzeugung eines nullbalancierten, vollständigen Baumes wie bei MINHTON wird dieser deterministisch aufgebaut. Zusätzlich haben alle Knoten in dieser Konstellation stets die maximale Anzahl an horizontalen Verbindungen. Hierdurch steigt sowohl die Stabilität des Netzwerks als auch die Sucheffizienz des Overlays, welche aus der größeren Sprungmöglichkeit zwischen zwei Knoten beim Routing resultiert. In der nachfolgenden Tabelle 5.5 ist eine Übersicht der Kosten für MINHTON, nBATON* und BATON* dargestellt.

Es wurde gezeigt, dass mithilfe von MINHTON die Kosten für den Auf- und Abbau des Netzwerks gegenüber nBATON* deutlich reduziert werden können, sodass die Vorteile eines Baumes mit minimaler Höhe ausgenutzt werden können. Im Gegensatz zu dem höhenbalancierten Baum BATON* und seinen Derivaten [JOV+06, SJG19] wird MINHTON deterministisch aufgebaut und benötigt keine Rotation. Mit der Erhöhung des Fanouts m werden die hier vorgestellten Algorithmen effizienter. Allerdings bleibt zu erwähnen, dass durch diese Erhöhung auch die Anzahl der Verbindungen zu anderen Knoten steigt.



(a) Kosten für eine *SearchExact*-Anfrage für BATON* [JOT+06]



(b) Kosten für eine *SearchExact*-Anfrage für MINHTON

Abb. 5.13: Durchschnittlichen Kosten für eine Suchanfrage (*SearchExact*) für einen bestimmten Knoten von BATON* (oben) mit MINHTON (unten). MINHTON ist um bis zu 50 % effizienter als BATON*.

Tabelle 5.5: Übersicht der Laufzeiten bzw. der Kosten von MINHTON, nBATON* und BATON*

	MINHTON	nBATON* [DGB21]	BATON* [JOT$^+$06]
Join	$O(m \cdot \log_m N)$	$O(\frac{N}{2m^2})$	$O(m \cdot \log_m N)$
Positionsfindung	$O(\log_m N)$	$O(\frac{N}{2m^2})$	$O(\log_m N)$
Update	$O(m \cdot \log_m N_{m,l})$	$O(m \cdot \log_m N_{m,l})$	$O(m \cdot \log_m N_{m,l})$
Leave	$O(m \cdot \log_m N)$	$O(\frac{N}{2m^2})$	$O(m \cdot \log_m N)$
Positionsfindung	$O(\log_m N)$	$O(\frac{N}{2m^2})$	$O(\log_m N)$
Update des Nachfolgerknotens	$O(m \cdot \log_m N_{m,l})$	$O(m \cdot \log_m N_{m,l})$	$O(m \cdot \log_m N_{m,l})$
Update des verlassenden Knotens	$O(m \cdot \log_m N_{m,l})$	$O(m \cdot \log_m N_{m,l})$	$O(m \cdot \log_m N_{m,l})$
SearchExact			
analytisch	$O(\log_m N)$	$O(\log_m N)$	$O(\log_m N)$
simulativ	$O(\sim \frac{1}{2} \cdot \log_m N)$	$O(\sim \frac{1}{2} \cdot \log_m N)$	$O(\log_m N)$
Kriterium des Baumes	nullbalanciert, vollständig	nullbalanciert	höhenbalanciert
Höhe des Baumes	minimal $\log_m N$	minimal $\log_m N$	nie minimal $1,44 \cdot \log_m N$ [Knu98, FO82]
Rotation des Baumes notwendig?	✗	✗	✓

5.4 Das Auffinden von Peers in MINHTON

Im Folgenden wird das Auffinden von Peers vorgestellt, welches auf dem Systemmodell von MINHTON basiert. Dabei besitzen Peers Fähigkeiten und Eigenschaften, die als ein unabhängiges Set von Schlüsselwertpaaren (engl.: *Key-Value-Pair (KVP)*) modellierbar sind. Folglich sind die KVPs eines Peers p definiert als $KV(p) = \{(K_1,V_1),\ldots,(K_n,V_n)\}$. Teile dieses Abschnitts basieren auf den Vorarbeiten des Autors in [DGB22].

In Abb. 5.14 ist das Konzept der Peer Discovery dargestellt, welches sich in zwei Phasen aufteilt: In der ersten Phase sendet der Knoten (*2:1*) seine Suchanfrage ϕ an eine Auswahl von bestimmten Knoten auf jedem geraden Level, z. B. Knoten (*0:0*), (*2:2*) und (*4:6*). Diese Knoten werden nachfolgend auch Dominating Set Nodes (DSNs) genannt. Jeder Knoten p kann die Höhe von MINHTON mithilfe

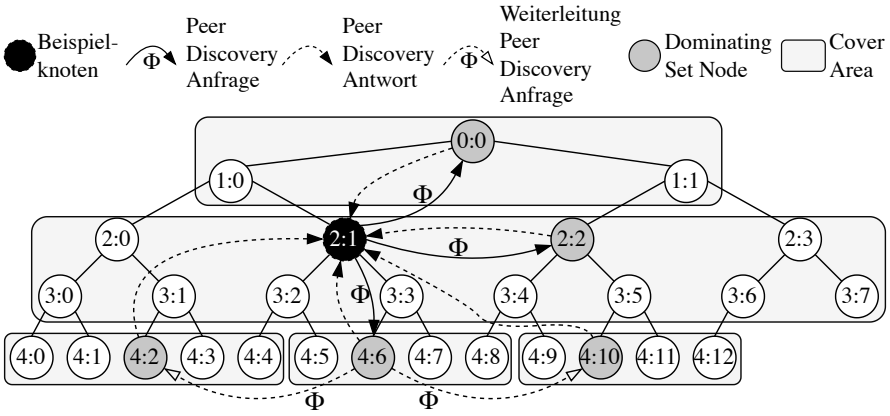


Abb. 5.14: Knoten (*2:1*) sendet seine Suchanfrage ϕ an eine Dominating Set Node (DSN) pro Level, auf dem mindestens ein DSN existiert. Jeder DSN leitet die Suchanfrage an die anderen DSNs auf demselben Level weiter. Jeder DSN wertet die Suchanfrage aus und schickt sein Ergebnis an den initialen Knoten.

seiner Adjazenten abschätzen, da mindestens ein Nachbar von p entweder auf dem höchsten Level $h-1$ oder ein Level unter dem höchsten Level $h-2$ liegt. Nachdem die Suchanfrage an einen DSN pro Level gesendet wurde, erfolgt die Weiterleitung der Suchanfrage ϕ an die weiteren DSNs auf diesem Level. Zum Beispiel leitet der DSN (*4:6*) die Suchanfrage an die anderen DSNs auf Level 4, konkret an die DSNs (*4:2*) und (*4:10*). Diese Suchanfrage wird weitergeleitet, bis alle DSNs auf einem Level sie erhalten haben. Auf einem Level l, auf dem DSNs existieren, bildet die Menge aller DSNs zusammen ein Connected Dominating Set (CDS). Jedes CDS auf einem Level steht für sich selbst, folglich ist jedes CDS unabhängig.

Definition Connected Dominating Set [WL99]
Eine Untermenge *D* von Knoten in einem Graphen *G* ist ein *Dominating Set*, wenn jeder Knoten, der nicht in *D* vorhanden ist, eine Verbindung zu einem Knoten in *D* hat. Wenn die Untermenge in *D* miteinander verbunden ist, dann ist *D* ein Connected Dominating Set.

Knoten, die einem DSN zugeordnet sind, können der Definition zufolge mit nur einer Nachricht diesen erreichen. Somit können KVPs in einen DSN eingefügt, aktualisiert oder entfernt werden mit einem Aufwand von $\mathcal{O}(1)$. In dem Netzwerk, welches in Abb. 5.14 dargestellt ist, existieren drei unabhängige CDSs auf den Leveln 0, 2 und 4.

Die zweite Phase beinhaltet eine Auswertung der Suchanfrage ϕ. Ein DSN ist für die Verwaltung einer Menge an Peers und deren KVPs zuständig, welche nachfolgend auch Cover Area (CA) genannt wird. Jeder DSN wertet die vorhandenen KVPs in seiner CA aus. Diese basiert entweder auf seinen gespeicherten Informationen oder neu angeforderten Informationen von den Knoten, welche in seiner CA existieren. Die Auswertung der Suchanfrage und somit das Ergebnis ϕ_r wird von jedem DSN an den anfragenden Knoten (*2:1*) zurückgeschickt.

5.4.1 Herausforderungen beim Finden eines Peers

Neben dem autonomen Verhalten von Peers, bspw. in Bezug auf das Betreten oder Verlassen des Netzwerks, sind ihre Eigenschaften und Fähigkeiten heterogen [Sch01]. Dies macht die Suche nach einem oder mehreren Peers und ihren Fähigkeiten oder Eigenschaften, auch bekannt als das *Peer Discovery Problem*, besonders herausfordernd [dRB14]. Nachfolgend werden am Beispiel von Teilnehmern eines CPPSs (vgl. Abschn. 3.1.2) die heterogenen Eigenschaften und Fähigkeiten beschrieben, bei denen diese als KVPs modelliert sind:

- **Heterogene Keys**: In einem CPPS existieren unterschiedliche Teilnehmer, bei denen jeder Peer unterschiedliche Keys haben kann. Dies geht aus der Tatsache hervor, dass bspw. ein FTF andere Eigenschaften und Fähigkeiten besitzt als ein Sensor, eine Maschine oder ein Mensch. Neben den standardisierten Informationen eines FTF (vgl. FTF-Informationsmodell Abschn. 4.1) kann jeder Hersteller noch weitere herstellerspezifische Daten hinzufügen.
- **Einfügen und entfernen von Keys**: Zu jedem Zeitpunkt können neue KVPs hinzukommen, aber auch wieder entfernt werden. Folglich ist die Menge an KVPs nicht vorherbestimmt. Bei einem Transport einer Ware mittels eines FTFs können z. B. die Informationen über den Ladungsträger aufgenommen werden. Diese Informationen sind nur so lange mit dem FTF verknüpft, bis die Ware den Bestimmungsort erreicht hat. Nach Abschluss des Transports sind diese Informationen nicht mehr Teil des FTF.

- **Unterschiedliche Aktualisierungsraten**: Ein Peer kann statische, aber auch dynamische KVPs besitzen. Dabei können die dynamischen KVPs unterschiedliche Aktualisierungsfrequenzen haben, welche sich sogar über die Zeit noch ändern. Der Ladungsträgertyp eines Fahrerlosen Transportfahrzeugs ist ein Beispiel für einen statischen KVP, da dieser sich nicht mehr über die Zeit ändert. Dargestellt werden kann dieser als $K_1 = LoadCarrierType$ und $V_1 = Pallet$. Weitere Beispiele für statische KVPs sind u. a. die Dimensionen des Fahrzeugs, der Hersteller oder das maximale Ladegewicht, welches das FTF transportieren kann. Der Batteriezustand hingegen, $K_2 = Battery$ und $V_2 = 42\,\%$, ist ein Beispiel für einen dynamischen KVP, da dieser sich über die Zeit verändern kann. Je nach aktuell zu transportierender Last nimmt dieser schneller bzw. langsamer ab. Als weitere Beispiele lassen sich hier das aktuelle Gewicht, eine Auftragswarteschlange oder die aktuelle Position des Fahrzeugs nennen.
- **Verschiedene Value Typen**: Die einem Key zugehörigen Values können verschiedene Datentypen haben (u. a. numerisch, boolean, Zeichenketten oder Listen). Während die Position eines FTF vereinfacht dargestellt werden kann als zwei Numerische Werte für die Position, so kann der Hersteller ein String sein oder die Auftragswarteschlange eine Komposition aus verschiedenen Datentypen.

Bei einer Peer Discovery wird eine Suchanfrage an das P2P-Netzwerk geschickt. Dabei kann die Abfrage auf unterschiedliche Weise formuliert werden. Dies reicht von einfachen, semantikfreien Suchen nach einem knotenidentifizierenden Key bis hin zu komplexen booleschen Ausdrücken, die aus zahlreichen Schlüsseln oder zugehörigen, ressourcenbeschreibenden Werten bestehen [RM06]. Eine Suchanfrage ϕ kann auf boolscher Algebra basieren, welche Literale zum Vergleich verwendet. So kann eine Anfrage ϕ logische Verknüpfungen (*und* ($\wedge$), *oder* ($\vee$) und *nicht* ($\neg$)) sowie Vergleiche (*größer* ($>$), *kleiner* ($<$), *größer oder gleich* ($\geq$), *kleiner oder gleich* ($\leq$), *gleich* ($=$) oder *ungleich* ($\neq$)) enthalten, z. B. $\phi := (\psi_1 \vee \neg\, \psi_2) \wedge \psi_3$. Die Literale ψ_i werden für den Vergleich der KVPs benutzt. Als Beispiel ist $\psi_1 := (K_1 < 5)$ dann *wahr*, wenn $V_1 < 5$. Die Suchanfrage ϕ wird mit der Antwort ϕ_r beantwortet und enthält die Informationen über jeden Peer $p_i \in P$, dessen $KV(p_i)$ die Suchanfrage ϕ erfüllt.

In strukturierten P2P-Netzwerken haben insbesondere DHTs effiziente Einfüge- und Suchoperationen mit logarithmischer Komplexität, allerdings haben diese dafür höhere Wartungskosten [LTL11]. Diese beinhalten die Datendelegation und den Lastausgleich/Lastverteilung (engl.: *load balancing*) [MG20]. Die Datendelegation setzt beispielsweise voraus, dass jeder Peer für einen bestimmten Bereich von zugehörigen
Schlüsseln verantwortlich ist. Zum Beispiel ist der Peer p_i für den Wertebereich $0x0815 - 0x4711$ zuständig. Will nun ein Peer p_k Daten in der DHT ablegen, so müssen die Daten gemäß der Hashing-Funktion zunächst gehashed werden. Mithilfe der Hashing-Funktion erhält der Peer p_k für $Key = Hash(Battery) = 0x0842$, welcher sich im Wertebereich von p_i befindet. Damit nun p_k seine Daten bei p_i einfügen kann, muss dieser zunächst gefunden werden. Dies erfolgt mithilfe der $Lookup(Key)$-Operation, welche p_k den zugehörigen Peer für den Key zurück

liefert. Erst im Anschluss können die Daten eingefügt bzw. aktualisiert werden. Ändern sich nun die Daten beim Peer p_k und die Hashing-Funktion liefert einen anderen Key zurück, welcher nicht im Wertebereich von p_i liegt, so müssen die bisherigen Daten, welche beim Peer p_i eingetragen waren, entfernt werden. Hierfür muss zunächst eine *Lookup*-Operation auf den alten Key $0x0842$ ausgeführt werden und die Daten dort ausgetragen werden. Erst im Anschluss können die Daten beim neuen Peer eingetragen werden. Allerdings besteht auch die Möglichkeit, dass der bisherige Peer p_i nicht mehr für diesen Key zuständig ist. Je nach Aktualisierungsrate der Daten in einem Peer p kann dies zu einem hohen Wartungsaufwand führen. Das Load Balancing hingegen sorgt dafür, dass kein einzelner Peer eine wesentlich höhere Last bewältigen muss als ein anderer [JOT+06, MKL+]. Die zugewiesenen Wertebereiche werden unter den Peers aufgeteilt und die dazugehörigen Daten entsprechend verteilt. Dadurch ist es möglich, dass ein zuvor zugewiesener Peer nicht mehr für die Verwaltung eines bestimmten Wertebereichs zuständig ist. In beiden Fällen verursacht die Suche nach dem zugehörigen Peer bzw. die Verteilung der Daten unter den Peers zusätzlichen Kommunikationsaufwand. In Szenarien, in denen sich die KVPs häufig ändern, muss der entsprechende Peer gefunden werden, bevor diese aktualisiert werden können. Tritt zum gleichen Zeitpunkt ein Load Balancing ein, wird der Wertebereich auf diverse Peers aufgeteilt. Ändert sich der KVP erneut, so kann ein ganz anderer Peer für den Key zuständig sein, sodass das Load Balancing ggf. nicht notwendig gewesen wäre bzw. die Fertigstellung des Load Balancings den zeitlichen Aufwand übersteigt, bis die nächste Aktualisierung des KVPs erfolgt ist.

In unstrukturierten P2P-Netzwerken kann eine Suchanfrage mithilfe eines Flutalgorithmus (engl.: *flooding*) überflutet werden, indem alle Knoten des Netzwerks angesprochen und ggf. ungewollte Duplikate versendet werden, was zu einer höheren Netzwerklast führt [TR03, LW05]. Eine andere Möglichkeit ist, die Suchanfrage an eine zufällig ausgewählte Teilmenge an Knoten zu schicken, bei der jede Anfrage sich einen eigenen Pfad durchs Netzwerk sucht, begrenzt durch die maximale Anzahl an erlaubten Sprüngen [TR03]. Weitere Ansätze sind zum Beispiel die Verwendung von Dominating Nodes oder Super-Peers [LW05]. Dominating Nodes werden aus bestehenden Knoten im Netz ausgewählt, und Super-Peers werden dem Netz in der Regel hinzugefügt, um beispielsweise mehr Rechenressourcen, einen höheren Grad an Konnektivität bzw. Zentralität oder eine höhere Stabilität zu bieten [LZL+05, BYG03]. Eine Peer-Discovery-Anfrage kann entweder direkt von diesen Peers beantwortet werden oder an andere Knoten delegiert werden. Das Finden eines minimalen CDSs gehört der Klasse von Problemen an, welche in nichtdeterministisch polynomieller Zeit (NP-schwer) gelöst werden kann [GK98].

5.4.2 Die Konstruktion des Connected Dominating Sets

Die Wahl eines CDSs kann bspw. auf inhaltlicher Ähnlichkeit der Daten, der verfügbaren (Rechen- oder Speicher-)Kapazität eines Knotens oder auf Verhandlun-

gen zur Laufzeit basieren [MHC06]. Letzteres benötigt zusätzliche Kommunikation zur Laufzeit des P2P-Netzwerks. Wird ein P2P-Netzwerk deterministisch auf- und abgebaut, so kann ein CDS ohne zusätzliche Kommunikation konstruiert werden.

Anstatt ein CDS über den gesamten Baum zu erstellen, werden nachfolgend unabhängige CDSs auf bestimmten Leveln des Baums erstellt. Allgemein gilt, dass jeder Knoten in MINHTON ein DSN werden kann. Ob ein Knoten ein DSN wird oder nicht, ist abhängig von der Position eines Knotens innerhalb von MINHTON (vgl. Abschn. 5.2.1). Um eine effiziente, aber einfache Annäherung an ein kleines CDS für ein Level l in MINHTON zu bestimmen, wird hierfür die Erstellung der RT näher betrachtet. Per Definition hat jeder Knoten Verbindungen zu den $1,\ldots,m$ Knoten links und rechts auf dem gleichen Level, unter der Voraussetzung, dass diese existieren. Zusätzlich hat jeder Knoten Verbindungen zu seinen Kindern und eine Verbindung zu jedem Kind eines jeden RT-Eintrags, den RTC. Darüber hinaus hat jeder Knoten Verknüpfungen zu seinen Kindern und den Kindern der RT-Einträge. Wird diese Konstruktion berücksichtig, so kann eine Kommunikation zwischen dem DSN und den Knoten der RT und RTC in $\mathcal{O}(1)$ realisiert werden, ohne die Notwendigkeit von zusätzlichen Verbindungen und Aushandlungen zur Laufzeit.

Werden DSNs auf einem Level l mit dem Abstand von $2m$ zueinander gewählt, so haben alle $2m-1$ Knoten zwischen zwei DSNs mindestens eine Verbindung zu einem DSN. Darüber hinaus bestehen Verbindungen zwischen den beiden DSNs, wie in Abb. 5.15 dargestellt. Die Erstellung des CDSs erfordert keine zusätzliche Kommunikation oder Verwaltung, sondern nutzt die Struktureigenschaften von MINHTON aus.

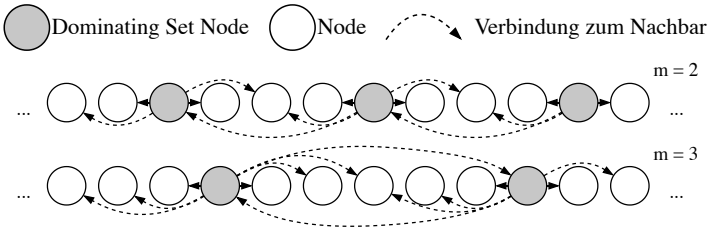


Abb. 5.15: Ein Connected Dominating Set wird erstellt basierend auf der Verlinkung mit den Nachbarn innerhalb der RT. Jeder DSN hat Verbindungen zu anderen Knoten und zu mindestens einem anderen DSN.

Mit dieser Konstruktion kann die Anzahl der DSNs auf einem gegebenen Level l für Fanout $m \geq 2$ durch Gl. (5.14) ausgedrückt werden. Wie in Gl. (5.15) angegeben, ist der Wurzelknoten des Baums immer ein DSN.

$$Pos_m(l) := \{i \cdot m \mid i = 2k+1, k \in \mathbb{N}_0, i \cdot m < m^l\} \tag{5.14}$$

$$Pos_m(0) := \{0\} \tag{5.15}$$

Ein Beispiel für Fanout $m = 2$ auf Level 4 und Gl. (5.14) sind die Knoten mit den Positionen $\{2, 6, 10, 14\}$ DSNs. Ein DSN auf Level l hat auch Verbindungen zu seinen RTC, die sich auf Level $l + 1$ befinden. Daher treten DSNs nur auf jedem zweiten Level auf, z. B. auf Leveln mit einer geraden Nummer. Zusätzlich ist eine Definition notwendig, welche Knoten von einem DSN abgedeckt werden. Aufgrund der Überschneidung von RT-Verbindungen von DSNs auf demselben Level könnten einige Knoten von mehreren DSNs abgedeckt werden. Zur Vermeidung zusätzlicher Kommunikation gilt daher die Definition der sogenannten CA – die Menge der Knoten, für die ein DSN zuständig ist – wie folgt:

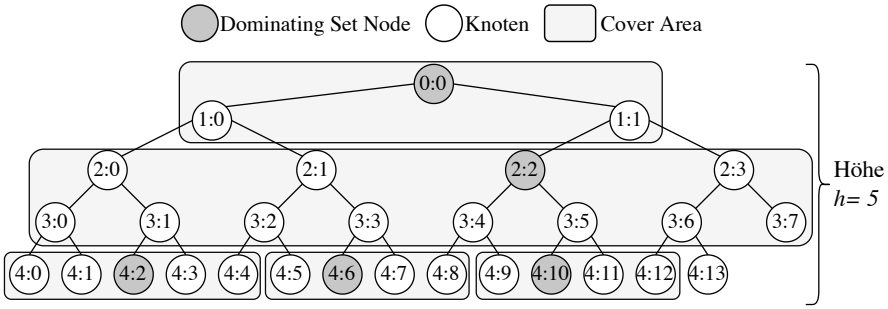
Definition 5.1 (Cover Area). Das Cover Area definiert die Menge der Knoten und deren KVPs, die ein DSN verwaltet. Diese beinhaltet, sofern die Knoten vorhanden sind:

- den DSN selbst,
- seine m RT Link-Nachbarn auf der linken und rechten Seite,
- seine eigenen m Kinder,
- die Kinder der RT Link-Nachbarn, die RTC.

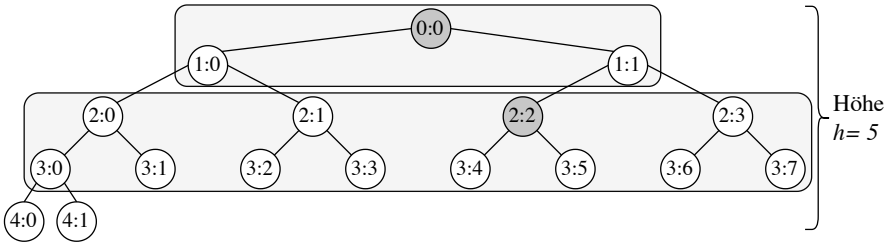
Wenn ein DSN einen benachbarten DSN auf der linken Seite hat, werden nur $m - 1$ Nachbarn auf der linken Seite in den CA aufgenommen. Daraus folgt, dass die CAs aller DSNs disjunkt sind und jeweils bis zu $m \cdot (2m + 3) + 1$ Knoten enthalten. Mit zunehmendem Fanout m wächst die Anzahl der Knoten innerhalb einer CA aufgrund des zunehmenden Abstandes von $2m$ zwischen den DSNs.

Ein DSN wird aufgrund der Konstruktion von MINHTON und dessen RT immer über den Beitritt oder das Verlassen eines Knotens innerhalb seiner CA informiert (vgl. Abschn. 5.3). Daher wird kein zusätzlicher Kommunikationsaufwand benötigt, um Informationen über die Knoten innerhalb einer CA zu erhalten.

Aufgrund der Auswahl von DSNs auf geraden Leveln besteht die Möglichkeit, dass einzelne Knoten möglicherweise nicht von einem DSN abgedeckt werden. Dieser Fall wird in Abb. 5.16a dargestellt, bei dem Knoten (*4:13*) keinem DSN zugeordnet ist, obwohl auf diesem Level bereits ein DSN existiert. Daher erweitert das nächstgelegene DSN (*4:10*), links von Knoten (*4:13*), vorübergehend sein CA. Dies verursacht nur eine geringe zusätzliche Kommunikation in Höhe von $\mathcal{O}(1)$ in diesem Sonderfall. Sobald der Knoten (*4:14*) dem P2P-Netzwerk beitritt und ein DSN wird, wird der Knoten (*4:13*) der CA des Knotens (*4:14*) zugewiesen. Ein weiterer Sonderfall ist, dass noch kein DSN auf einem Level existiert (vgl. Abb. 5.16b). Dabei wird ein temporärer DSN auf Level 3 benötigt. Einfachheitshalber wird per Definition der Elternknoten (*3:1*) des nicht existierenden DSNs (*4:2*) gewählt. Aufgrund der RT-Verbindung des temporären DSNs (*3:1*) zu (*3:0*) existieren auch Verbindungen zu dessen Kindern. Daher benötigt diese temporäre Zuweisung ebenfalls keinen zusätzlichen Kommunikationsaufwand. Durch die Verwendung der bestehenden RT-Verbindungen kann ein CDS statisch bestimmt werden, ohne zusätzli-



(a) Knoten (4:13) ist von keinem DSN abgedeckt, sodass der DSN (4:10) seine CA temporär um Knoten (4:13) erweitert.



(b) Knoten (4:0) und (4:1) sind keinem DSN zugewiesen. Daher wird der Knoten (3:1) – Elternknoten des nicht existierenden DSNs (4:2) – ein temporärer DSN und deckt so Knoten (4:0) and (4:1) mit ab.

Abb. 5.16: Ein DSN deckt eine Anzahl von Knoten in seiner CA ab, mit der Ausnahme der Knoten auf Level 4.

chen Kommunikationsaufwand zur Laufzeit. Mit steigender Anzahl an Knoten in MINHTON skaliert die Wahl des CDS inhärent mit und die Anzahl an Knoten in jedem DSN ist durch die Verwendung der RT nach oben hin beschränkt.

5.4.3 Weiterleitung einer Suchfanfrage im Connected Dominating Set

Für die Durchführung einer Suchanfrage müssen alle unabhängigen CDSs auf den jeweiligen Leveln bzw. alle zugehörigen DSNs besucht werden. Aufgrund der Konstruktion der RTs hat ein DSN auch Verbindungen zu anderen DSNs. Daher muss nur ein DSN eines jeden CDSs angesprochen werden. Gegeben sei die RT des Knotens (*6:2*) durch $RT(6:2) = \{3,4,6,10,18,34\}$. Durch die Konstruktion des CDS (vgl. Gl. 5.14, S. 129) sind die Knoten $6,10,18,34$ ebenfalls auf diesem Level DSN. Daher kann die Weiterleitung der Suchanfrage ϕ unter den CDSs aufgeteilt werden:

Eine Suchanfrage enthält neben dem Query ϕ auch zwei Intervalle, gegeben als $I = [b_{low}, b_{up}]$. In dem Fall, dass ein DSN (*l:n*) die Suchanfrage zuerst auf dem Level erhält, ist der DSN für die Weiterleitung der Nachricht von seiner Position n zu beiden Seiten verantwortlich, d. h. damit ergibt sich ein Intervall für die linke

Tabelle 5.6: Exemplarische Weiterleitung einer Suchanfrage über das Connected Dominating Set auf Level 6 und Fanout $m = 2$, beginnend bei Knoten 2

DSNs auf Level 6			2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62			
Runde	**Sender**	**Knoten**	**RRT(Knoten)**	**DSNs in RRT**	**RI**	**DSNs in RI**
0	-	2	3, 4, 6, 10, 18, 34	6, 10, 18, 34	[3, 63]	6, 10, 18, 34
1	2	6	7, 8, 10, 14, 22, 38	10, 14, 22, 38	[6,9]	-
1	2	10	11, 12, 14, 18, 26, 42	14, 18, 26, 42	[11,17]	14
1	2	18	19, 20, 22, 26, 34, 50	22, 26, 34, 50	[19,33]	22, 26
1	2	34	35, 36, 38, 42, 50	38, 42, 50	[35,63]	38, 42, 50
2	10	14	15, 16, 18, 22, 30, 46	18, 22, 30, 46	[15,17]	-
2	18	22	23, 24, 26, 30, 38, 54	26, 30, 38, 54	[22,25]	-
2	18	26	27, 28, 30, 34, 42, 58	30, 34, 42, 58	[27,33]	30
2	34	38	39, 40, 42, 46, 54	42, 46, 54	[39,41]	-
2	34	42	43, 44, 46, 50, 58	46, 50, 58	[43,49]	46
2	34	50	51, 52, 54, 58	54, 58	[51,63]	54, 58
3	26	30	31, 32, 34, 38, 46, 62	34, 38, 46, 62	[31,33]	-
3	42	46	47, 48, 50, 54, 62	50, 54, 62	[47,49]	-
3	50	54	55, 56, 58, 62	58, 62	[55,57]	-
3	50	58	59, 60, 62	62	[59,63]	62
4	58	62	62	-	[63]	-

RRT : Rechte Routingtabelle; DSN : Dominating Set Node; RI: Rechtes Intervall

Seite $L = [0, n-1]$ und für die rechte Seite $R = [n+1, m^l - 1]$. Der initiale DSN ist für die Adressierung jedes DSNs innerhalb des Intervalls zuständig, zu dem Verbindungen in seiner RT existieren. In Tabelle 5.6 ist die Weiterleitung der Suchanfrage exemplarisch für das Level 6 mit Fanout $m = 2$ dargestellt.

Zunächst berechnet ein DSN die Menge aller DSNs, zu denen es eine Verbindung in seiner RT und seinem zugehörigen Intervall R hat, gegeben durch $S_R = \{s \in RT \mid s \in Pos_m(l) \wedge b_{low} \leq s \leq b_{up}\}$. Ein DSN kann basierend darauf ein neues Intervall für jeden $s_j \in S_R$ berechnen und so die Anfrage weiterleiten. Sei $S_R = \{s_1, s_2, \ldots, s_{i-1}, s_i\}$ die DSNs in R. S_R ist dabei sortiert nach $s_1 < s_2 < \cdots < s_{i-1} < s_i$. Das zugeordnete Intervall für s_j mit $j = 1, \ldots, i-1$ ist $R_{s_j} = [s_j + 1, s_{j+1} - 1]$ und $R_{s_i} = [s_i + 1, b_{up}]$. Außerdem ist $L_{s_i} = \emptyset$ für alle $s_j \in S_R$. Der DSN leitet die Suchanfrage an jedes $s_j \in S_R$ mit den entsprechenden (Teil-)Intervallen R_{s_j} und L_{s_j} weiter. Dies wird so oft wiederholt, bis kein DSN innerhalb eines zuge-

wiesenen Intervalls mehr vorhanden ist, d. h. $S_R = \emptyset$. Die Weiterleitung der Suchanfrage an das linke Intervall L ist äquivalent.

5.4.4 Auswertung einer Suchanfrage in einer Dominating Set Node

Wenn ein DSN eine Suchanfrage ϕ erhält, so wertet dieser aus, welche Knoten in seiner CA ϕ erfüllen. Dies passiert auf Grundlage der gespeicherten lokalen Informationen, auch lokale Sicht genannt. Diese lokale Sicht beruht auf den Informationen, die Knoten innerhalb einer CA an den dafür zuständigen DSN gesendet haben. Zur Reduktion des Datenaufkommens im Netzwerk hält jeder DSN für jeden Knoten und den dazugehörigen KVP eine Unsicherheit (engl.: *uncertainty*) vor, basierend auf der Aktualität (engl.: *freshness*). Die Aktualität spiegelt das Alter der Informationen wider [CG00], während die $freshness_{threshold}$ eine Zeit angibt, wie lange ein KVP aktuell ist. Um die Aktualität einer KVP zu bewerten, wird ein relativer Zeitstempel der letzten Nachricht ($KVP(t)$) gespeichert. Auf diese Weise kann ein KVP als veraltet betrachtet werden, wenn $freshness_{threshold} < t - KVP(t)$ ist.

Gegeben ist ein MINHTON-Netzwerk mit $m = 2$ und sieben Knoten. Die lokale Sicht der DSNs (*0:0*) und (*2:2*) ist in Tabelle 5.7 dargestellt. Die Suchanfrage lautet $\phi := \psi_1 \wedge \psi_2$, mit $\psi_1 := (K_1 \leq 5)$ und $\psi_2 := (K_2 = 1)$. Zur Feststellung, ob ein Knoten die Anfrage erfüllt (*true*) oder nicht (*false*), wird die lokale Sicht eines DSNs berücksichtigt. Fehlen die KVPs bzw. diese veraltet sind, so werden diese innerhalb des DSNs als *undecided* markiert. Ist ein KVP als *undecided* markiert, muss dieser vom DSN abgerufen werden.

Tabelle 5.7: Lokale Sicht der DSN (*0:0*) und (*2:2*). DSN (*0:0*) deckt sich selbst und Knoten (*1:0*) und (*1:1*) ab; DSN (2:2) deckt sich selbst und Knoten (2:0), (2:1) und (2:3) ab. Die Peer-Discovery-Anfrage $\phi = \psi_1 \wedge \psi_2$ wird mit $\psi_1 = (K_1 \leq 50)$ und $\psi_2 = (K_2 = 10)$ ausgewertet.

DSN	**CA**	K_1	**freshness**	ψ_1	K_2	**freshness**	ψ_2	ϕ
(0:0)	(0:0)	40	✓	t	10	✓	t	t
	(1:0)	100	✓	f	10	✗	u	f
	(1:1)	60	✗	u	-	-	u	u
(2:2)	(2:0)	3	✗	u	10	✓	t	u
	(2:1)	10	✗	u	20	✗	u	u
	(2:2)	20	✓	t	-	-	f	f
	(2:3)	100	✗	u	10	✓	t	u

CA : Cover Area ; t = true; f = false; u = undecided

Damit die Suchanfrage ϕ erfüllt werden kann, müssen sowohl ψ_1 als auch ψ_2 *true* sein. Zum Beispiel erfüllt der Knoten (*1:0*) nicht die Bedingung für ψ_1, da $K_1 = 10$. Folglich ist es unnötig, ψ_2 des Knotens (*1:0*) zu betrachten, da ϕ nicht

mehr erfüllt werden kann. Allerdings ist der Wert K_1 des Knotens (*2:3*) veraltet. Obwohl die Informationen des Knotens (*2:3*) mit $K_1 = 10$ nicht ψ_1 erfüllt ist, besteht die Möglichkeit, dass (*2:3*) nach einer Aktualisierung ψ_1 erfüllen kann. Für den Knoten (*2:1*) ist $K_1 = 1$ ψ_1 erfüllt. Wird nun allerdings die *freshness* von K_1 berücksichtigt, so ist ψ_1 als unsicher klassifiziert. Im Knoten (*1:1*) existiert kein Eintrag für den Wert K_2. Daher kann nicht festgestellt werden, ob ψ_2 erfüllt ist oder nicht, und eine Aktualisierung der Daten ist erforderlich. Auch wenn ein Ausdruck *undecided* ist und der Knoten selbst ein DSN ist, ergibt die Auswertung *false*, da dieser DSN die vollständige Kenntnis hat.

Die Kombination von Literalen wie ψ_1 und ψ_2 folgt Wahrheitstabellen innerhalb der Fuzzy-Logik [TE15]. Hierbei wird *true* als 1 interpretiert, *false* als 0 und *undecided* als $1/2$. Folglich ist die *and*-Operation das Minimum aller literalen Werte und die *or*-Operation das Maximum. Eine Negation des Literalwertes x wird als $(1-x)$ berechnet.

In dem Fall, dass nur statische KVPs existieren, kann die *freshness* der KVPs ignoriert werden. Statisch bedeutet in diesem Zusammenhang, dass sich die Werte eines KVPs nie ändern. Ein DSN stellt durch Hinzufügen eines einfachen *static-flag* sicher, dass die KVPs immer als gültig markiert sind. Eine weitere Annahme ist, dass ein Knoten sein KVP, unabhängig ob ein neuer hinzukommt oder entfernt wird, seinem zugewiesenen DSN mitteilt. Daher weiß ein DSN, welche KVPs zu einem Knoten gehören. Wenn die lokale Auswertung eines DSNs einen KVP nicht enthält, so kann das Literal sofort als *false* ausgewertet werden.

5.4.5 Analytische Betrachtung

Für die Beantwortung einer Suchanfrage ϕ wird die Aggregation von Informationen über das gesamte Netzwerk benötigt. Die Kosten korrelieren mit der Anzahl der vorhandenen DSNs und der Anzahl der Knoten, die als *undecided* betrachtet werden. Die Anzahl der DSNs, die in einem Baum mit der Höhe h existieren, bei dem das höchste Level vollständig ist, ist gegeben durch (5.16).

$$Size_m^{comp}(h) := \sum_{i=0}^{k-1} \left\lceil \frac{m^{2\mathrm{i}}}{2m} \right\rceil, k = \left\lceil \frac{h}{2} \right\rceil \tag{5.16}$$

DSNs existieren nur auf geraden Leveln, daher sei k die Anzahl der geraden Level im Baum. Die Höhe des Baumes wird dabei durch Gl. (5.5) berechnet (vgl. Abschn. 5.2.1).

Eine vollständiges Level l besteht aus m^l Knoten, während DSNs in einem Abstand von $2m$ platziert sind. Somit hat ein vollständiges CDS auf Level l die Größe von $\lceil m^{\mathrm{l}}/2m \rceil$.

Um (5.16) auf eine beliebige Netzwerkgröße N zu erweitern, müssen zwei Fälle berücksichtigt werden: Im Falle einer geraden Baumhöhe h ist das höchste Level $h-1$ ungerade, und DSNs befinden sich nicht auf ungeraden Leveln. Bei einer un-

geraden Baumhöhe h enthält das gerade Level $h-1$ jedoch DSNs und ist möglicherweise nicht vollständig. Auf dem Level $h-1$ gibt es insgesamt $N_{h-1} = N - \sum_{i=0}^{h-2} m^i$ Knoten. Es gibt also $\lceil N_{h-1}/2m \rceil$ DSNs auf dem höchsten Level. Folglich ist die Anzahl an DSNs gegeben durch (5.17):

$$Size_m(N) := Size_m^{comp}(h-2) + \left\lceil \frac{N_{h-1}}{2m} \right\rceil \cdot (h \bmod 2) \tag{5.17}$$

In Abb. 5.17 ist die Anzahl der DSNs mittels (5.17) für verschiedene Fanouts m und Netzwerkgrößen N dargestellt. Die konstanten Geraden entsprechen dem Fall der geraden Baumhöhen, die lineare Steigung dem Fall der ungleichen Baumhöhen.

Basierend darauf kann die Anzahl der Nachrichten bestimmt werden, die notwendig sind um jeden DSN zu adressieren. Hierfür wird (5.17) verwendet. Seien $se(N) \in \mathcal{O}(\log_m(N))$ die Kosten für die Adressierung eines DSNs mithilfe des *SearchExact*-Algorithmus (vgl. 5.3.1, S. 113).

Die Adressierung eines DSNs auf jedem geraden Level erfordert daher $se(N) \cdot k$ Nachrichten. Die restlichen DSNs im Netzwerk werden mit einer Nachricht über die Weiterleitung der Suchanfrage im CDS (vgl. Abschn. 5.4.3) adressiert, was $Size_m(N) - k$ Nachrichten entspricht. Nach der Auswertung sendet jeder DSN eine Antwort an den anfragenden Knoten zurück, wodurch $Size_m(N)$ zusätzliche Nachrichten entstehen.

Nachdem ein DSN die Anfrage erhalten hat, findet die Auswertung der Anfrage statt. Jeder *undecided* Knoten kann von dem entsprechenden DSN innerhalb von $\mathcal{O}(1)$ direkt adressiert werden. Die Anzahl der *undecided* Knoten hängt jedoch von einigen Parametern ab, wie z. B. der $freshness_{threshold}$ (vgl. Abschn. 5.4.4). Unter Berücksichtigung der unbekannten Variablen sei $undecided(N,E,Q)$ die Menge der *undecided* Knoten in einem Netzwerk der Größe N, verschiedener Umgebungsvariablen E und der Art der Suchanfrage Q. Insgesamt können die Kosten C_m für eine Peer-Discovery-Anfrage in MINHTON der Größe N und Fanout m wie folgt geschätzt werden:

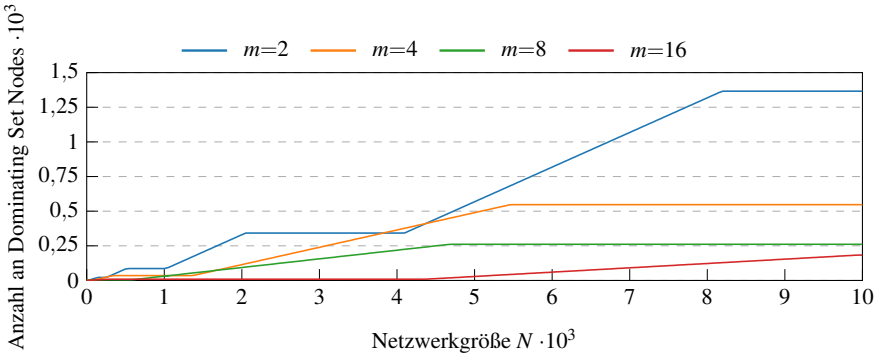


Abb. 5.17: Anzahl der Dominating Set Nodes in Abhängigkeit von der Netzwerkgröße N und dem Fanout m

$$C_m(N,E,Q) = se(N) \cdot k + 2 \cdot Size_m(N) - k + 2 \cdot |Undecided(N,E,Q)| \quad (5.18)$$

Im Worst Case, bei dem jeder DSN alle Knoten in seiner CA als *undecided* einstuft, können die Kosten $\mathcal{O}(N)$ entsprechen. Hierfür muss ein DSN von allen entsprechenden Knoten neue Werte anfordern. In Szenarien mit nur statischen KVPs korrelieren die Kosten für eine Suchanfrage mit der Anzahl der DSNs, die angesprochen werden müssen. Dennoch können Abfragen in vielen Netzwerkumgebungen effizient durchgeführt werden.

5.4.6 Simulative Evaluation

Im Folgenden Abschnitt wird das vorgeschlagene Peer Discovery für MINHTON anhand von Simulationsergebnissen verifiziert. Dabei soll das untersuchte System quantitativ bewertet und die durchgeführte theoretische Analyse ergänzt werden. Die Simulationsexperimente wurden mit dem diskreten Ereignis-Netzwerksimulator ns-3 [RH10] durchgeführt. Zu den Merkmalen, die von Interesse sind, gehören der generierte Verkehr (Anzahl der ausgetauschten Nachrichten) für das Auffinden von Peers anhand einer Suchanfrage ϕ innerhalb des Netzwerks und die Verteilung der Last auf die teilnehmenden Peers. Hierfür existieren vier verschiedene experimentelle Setups (ES), bei denen unterschiedliche Parameter variiert wurden. In Tabelle 5.8 sind die verschiedenen experimentellen Setups (ES) und ihre Parametrisierung dargestellt. Es wurden Netzwerke mit unterschiedlichen Größen N von 1.000 bis 10.000 Knoten und mit unterschiedlichen Fanouts ($m \in \{2, 4, 8, 16\}$) simuliert.

Jeder KVP hat eine unterschiedliche Aktualisierungsrate (Parameter 9) unter Berücksichtigung einer Normalverteilung mit unterschiedlichem Erwartungswert μ und Varianz σ. Um die Heterogenität der Peers hinsichtlich ihrer verfügbaren KVPs abzubilden, wurde eine Wahrscheinlichkeit eingeführt, welche beschreibt, ob ein KVP existiert oder nicht (Parameter 10). In einem DSN existiert ein Schwellenwert für die Gültigkeit ($freshness_{threshold}$) der KVPs, bevor ein DSN neue Daten anfordert (Parameter 11). Jedes Experiment wurde 10-mal – jeweils mit verschiedenen Startwerten für einen Zufallsgenerator – durchgeführt und für die nachfolgenden Abbildungen gemittelt.

Das Auffinden eines Peers

In Abb. 5.18 sind die Simulationsergebnisse von ES1 und ES2 für verschiedene Fanouts und Netzwerkgrößen dargestellt. Die Ergebnisse für ES1, bei dem alle KVPs statisch sind und ein DSN keine Daten von Knoten in seiner CA abrufen muss, sind in Abb. 5.18a dargestellt. Die Anzahl der Nachrichten besteht aus der Adressierung der DSNs auf geraden Leveln, der Weiterleitung der Anfrage über das entsprechend

Tabelle 5.8: Übersicht der Parametrisierung der experimentellen Setups (ES)

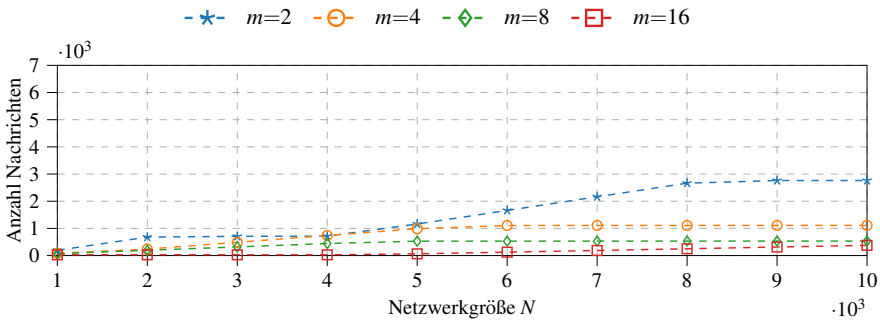
Nr.	Parameter	ES1	ES2	ES3	ES4
1	Fanouts $m \in \mathbb{N}$	$m \in \{2, 4, 8, 16\}$		$m \in \{8\}$	
2	Netzwerkgröße $N \in \mathbb{N}$	$N \in \{1k, 2k, 3k, \ldots, 10k\}$		$N \in \{2k, 4k, 6k, 8k, 10k\}$	
3	Anzahl der anfragenden Knoten	Randomisierte Verteilung; 10 % von N			
4	Anfrage Frequenz	Normalverteilung $\mu = 20\ s, \sigma = 4\ s$			
5	Query Form	$(\psi_i \wedge \psi_j)$ or $(\psi_i \vee \psi_j)$			
6	Literale Form	$\psi_k := (Key_k \leq x)$			
7	Maximale Anzahl an KVPs, die ein Knoten haben kann	10 statische	10 dynamische		
8	KVP-Wertebereich $\in \mathbb{R}$	Stetige Gleichverteilung auf dem Intervall $[0, 10]$			
9	Aktualisierungsrate für jeden KVP	-	$\{\mu_1 = 0.625\ s, \sigma_1 = 0.125\ s\}$, $\{\mu_2 = 1.25\ s, \sigma_2 = 0.25\ s\}$, ..., $\{\mu_{10} = 320\ s, \sigma_{10} = 64\ s\}$		
10	Wahrscheinlichkeit, dass ein Knoten den KVP hat	0.95		$\{0.5, 0.6, \ldots, 1.0\}$	0.95
11	Freshness-Schwellenwert für KVPs in einem DSN	$5\ s$			$\{1, 3, 5, ..., 21\}\ s$

KVP : Key-Value-Pair; DSN : Dominating Set Node

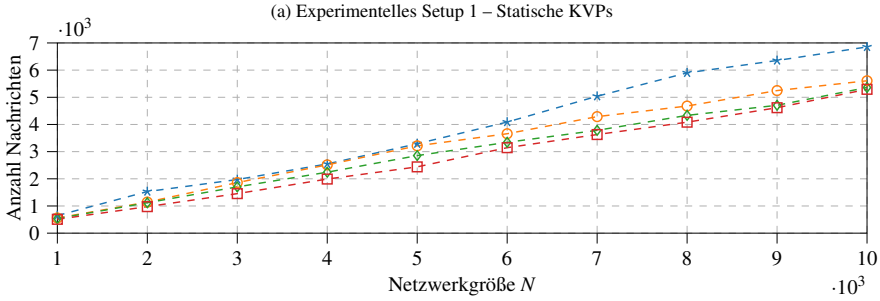
zugehörige CDS und dem Empfang aller Antworten. Daraus ergeben sich die Kosten von $se(N) \cdot k + 2 \cdot Size_m(N) - k$.

In Abb. 5.18b sind die Ergebnisse zum ES2 mit bis zu 10 dynamischen KVPs dargestellt. Die höhere Anzahl an erforderlichen Nachrichten hängt mit der Anzahl der *undecided* Knoten in einem DSN und dessen CA zusammen. Sobald ein DSN eine Anfrage als *undecided* bewertet, wird der letzte Status der KVPs abgerufen, sodass eine neue Kommunikation notwendig ist. Der *undecided*-Status eines KVP wird beeinflusst durch:

- die Aktualität der Daten: Kürzere Freshness-Schwellenwerte führen dazu, dass KVPs bei der Auswertung häufiger veraltet sind.
- das Verhältnis von statischen und dynamischen KVPs: Bei vielen statischen KVPs sind weniger *undecided* Bewertungen möglich.
- die Anzahl der KVPs in einer Abfrage: Eine Abfrage mit vielen verschiedenen KVPs hat eine höhere Wahrscheinlichkeit, *undecided*-Ergebnisse zu liefern.
- das Verhältnis von *and*- und *or*-Abfragen (ohne Berücksichtigung von verschachtelten Ausdrücken): In einfachen *or-queries* reicht ein wahres Literal aus, um einen Peer als wahr zu bewerten, während unentschiedene Literale ignoriert werden können. In *and*-Abfragen hingegen muss jedes *undecided* Literal abgefragt werden.



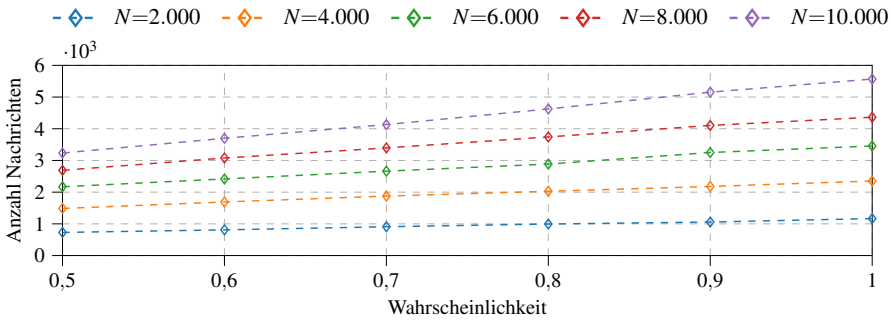
(a) Experimentelles Setup 1 – Statische KVPs



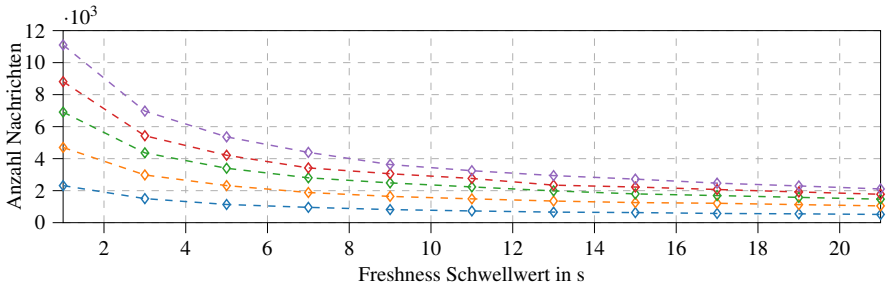
(b) Experimentelles Setup 2 – Dynamische KVPs

Abb. 5.18: Durchschnittliche Anzahl an Nachrichten für eine Peer-Discovery-Anfrage für die experimentellen Setups 1 (oben) und 2 (unten)

Abb. 5.19 zeigt die Ergebnisse für ES3 und ES4 mit einem festen Fanout $m = 8$ und unterschiedlichen Netzwerkgrößen N. Bei ES3 wurde die Wahrscheinlichkeit, dass ein Knoten eine KVP hat, variiert (vgl. Abb. 5.19a). Eine niedrige Wahrscheinlichkeit bedeutet, dass ein KVP seltener in einem Knoten existiert. Da ein DSN weiß, welche KVPs ein Knoten besitzt, kann die Auswertung des Literals abgebrochen werden, sofern kein Eintrag zu einem KVP existiert. Durch einen vorzeitigen Abbruch verringert sich die Zahl der *undecided* Knoten, sodass weniger Nachrichten gesendet werden müssen. In ES4 wurde der Freshness-Schwellenwert in einem DSN variiert (vgl. Abb. 5.19b). Dieser Schwellenwert bestimmt, wie lange ein KVP in einem DSN als aktuell bzw. gültig markiert ist. Sobald der Schwellenwert überschritten ist, muss ein DSN die Daten des entsprechenden KVP aktualisieren. Kürzere Freshness-Schwellenwerte führen dazu, dass häufig veraltete KVPs und Knoten als *undecided* bewertet werden. Im Fall von $freshness_{threshold} \rightarrow \infty$ geht ein DSN davon aus, dass jeder KVP statisch ist, sodass der Status eines KVPs nie aktualisiert werden muss. Dies würde dem ES1 entsprechen.



(a) Experimentelles Setup 3 – Variation der Wahrscheinlichkeit, dass ein Knoten einen KVP hat.



(b) Experimentelles Setup 4 – Variation des Freshness-Schwellenwerts, wie lange ein KVP in einem DSN als gültig gilt.

Abb. 5.19: Durchschnittliche Anzahl an Nachrichten für eine Peer Discovery Anfrage für die experimentellen Setups 3 (oben) und 4 (unten) für unterschiedliche Netzwerkgrößen N und festen Fanout $m = 8$

Ein- und ausgehende Belastung eines Peers

Welcher Last ein DSN ausgesetzt ist, ist in Abb. 5.20 dargestellt. Diese Last umfasst die durchschnittliche Anzahl der gesendeten Nachrichten für einen einzelnen DSN aus ES2. Die gesendeten Nachrichten beinhalten zwei verschiedene Nachrichtentypen: Der erste Nachrichtentyp ist die Weiterleitung einer Suchanfrage zwischen den CDS auf demselben Level (vgl. Abschn. 5.4.3). Die Suchanfrage wird an alle im Netzwerk verfügbaren DSN weitergeleitet. Bei x DSNs werden insgesamt $(x-1)$ Nachrichten versendet, sodass sich der Durchschnitt an $\frac{x-1}{x}$ versendeten Nachrichten je DSN annähert. Bei einem höheren Fanout existieren in einem Netzwerk weniger DSNs aufgrund der größeren Vernetzung und des daraus resultierenden größeren Abstands zwischen den DSNs. Daher ist das Verhältnis zwischen $(x-1)$ und x größer, sodass die durchschnittliche Anzahl an Nachrichten kleiner wird. Der zweite Nachrichtentyp entspricht der Anzahl der gesendeten Nachrichten an Knoten innerhalb seiner CA, basierend auf der Auswertung der Suchanfrage (vgl. Abschn. 5.4.4). Die absteigende Anzahl an Nachrichten, welche an Knoten innerhalb der CA gesendet werden, hängt mit der Gesamtanzahl der verfügbaren Knoten in einem Netz zusammen. Zum Beispiel gibt es in einem Netzwerk mit Fanout $m = 16$ und $N = 4.000$

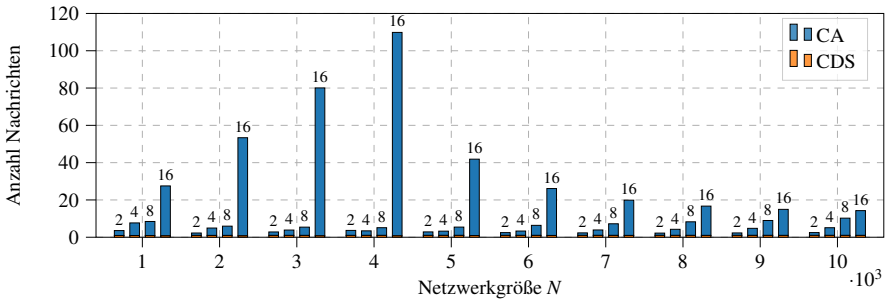


Abb. 5.20: Durchschnittliche Anzahl an versendeten Nachrichten für einen Dominating Set Node (DSN) während eines Peer-Discovery-Vorgangs für Fanouts $m = \{2,4,8,16\}$ und unterschiedliche Netzwerkgrößen für ES2. Der Wert überhalb der Säule entspricht dem Fanout.

jeweils ein CDS auf den Leveln 0 und 2. Das CDS auf Level 2 deckt auch alle verfügbaren Knoten auf Level 3 ab. Sobald jedoch das Level 4 ($N = 4.370$) betreten wird, wird ein neuer CDS erzeugt und die Last der DSNs auf Level 2 wird reduziert. In einem Netzwerk mit $N > 4.370$ hat jeder DSN auf Level 2 bis zu 528 Knoten in seiner CA. Solange das Level 4 nicht die maximale Anzahl an Knoten erreicht hat (65.536), besitzt jeder DSN auf Level 4 nur bis zu 32 Knoten in seiner CA. Somit sinkt die durchschnittliche Last je Peer.

Der hier vorgestellte Peer-Discovery-Algorithmus skaliert automatisch mit der Anzahl der Knoten, während der Fanout den Grad der Dezentralität bestimmt. Wird der Fanout $m \to \infty$ gewählt, so konvergiert der Algorithmus theoretisch gegen ein zentralisiertes System, in dem ein DSN für das gesamte Netzwerk zuständig ist. In einem System mit nur statischen Daten konvergiert die Anzahl an Nachrichten, welche für eine Suchanfrage benötigt werden, gegen die Adressierung der DSNs bzw. der Anzahl an vorhandenen DSNs.

5.5 Nachrichtenverteilung in einer Gruppe

Die Verbreitung von Nachrichten innerhalb eines Netzwerks ist bspw. dann notwendig, wenn sich ein Zustand eines Knotens bzw. eines Teilnehmers im Netzwerk ändert. Hierbei kann bspw. die Anzahl der Empfänger variieren. Existiert nur ein Empfänger, so handelt es sich um einen Unicast (*1:1*). Existiert eine ausgewählte Gruppe von Empfängern, so handelt es sich um einen Multicast (*1:n*). Wird eine Nachricht an alle Teilnehmer in einem Netzwerk versendet, so handelt es sich um einen Broadcast. Daher ist ein Multicast an eine Gruppe G gleichzusetzen mit einem Broadcast in G. Existieren mehrere Teilnehmer in einer Gruppe G, welche innerhalb dieser Gruppe Multicasts versenden wollen, so wird dies auch als (*m:n*)-Kommunikation bezeichnet. [ST17]. Teile dieses Abschnitts basieren auf den Vorarbeiten des Autors in [Sto22, Tön22].

Die Verbreitung einer Nachricht in einem Netzwerk kann mithilfe von Flooding, selektivem Routing oder einer epidemischen Ausbreitung (engl.: *gossip*) erfolgen [BQV09]. Beim Flooding sendet ein Knoten die Nachricht an alle seine bekannten Knoten weiter. Nachteilig wirkt sich hier die begrenzte Skalierbarkeit aus, da ein Knoten dieselbe Nachricht mehrfach erhält und somit unnötigerweise Duplikate erhält [QVL02]. Das selektive Routing hingegen versucht, das Versenden von Duplikaten beim Flooding zu reduzieren, indem nur eine Teilmenge der Knoten im Netzwerk jedes Abonnement speichert und eine Teilmenge der Knoten von jedem Ereignis besucht wird (beide Teilmengen, die sich möglicherweise über das gesamte System erstrecken). Die epidemische Ausbreitung basiert u. a. auf der Verteilung von Infektionskrankheiten in der Natur, welche für die Anwendung auf die Verteilung von Nachrichten in sehr großen verteilten Systemen Gegenstand der Forschung ist [ST17, Mon17].

Das Gossiping, auch Epidemic Broadcast Tree (EBT) genannt, ist ein Ansatz für hoch skalierbare Netzwerksysteme mit zuverlässigen Multi- oder Broadcasts [LPR07]. Die Zuverlässigkeit ist definiert als der Prozentsatz der verfügbaren Knoten, welche innerhalb der Multi- bzw. der Broadcast-Gruppe auch diese Nachricht erhalten [KMG03]. Folglich bedeutet eine Zuverlässigkeit von 100 %, dass das Verfahren in der Lage war, eine Nachricht an alle aktiven Knoten zu übermitteln. Allerdings handelt es sich bei dem EBT um einen Ansatz, bei dem der initiale Sender im Zentrum steht. Dies bedeutet, dass die Verbindungen in einem Netzwerk mit N Knoten lediglich für den Sender ausgehandelt werden, welcher als erstes eine Nachricht im Netzwerk versendet. Ändert sich der initiale Sender, so werden die Verbindungen neu ausgehandelt und es entstehen neue Strukturen. In einem (*m:n*)-Szenario, bei dem viele verschiedene Sender existieren, werden die Verbindungen in Abhängigkeit von der Anzahl der verschiedenen Sender wiederholt ausgehandelt. Ein Beispiel für für eine Implementierung eines EBT ist [TLM$^+$19], dessen Anwendung sich auf die konsistente Log-Replikation konzentriert.

In einem strukturierten P2P-Netzwerk hingegen existiert eine Topologie, in der ein deterministischer Broadcast durchgeführt werden kann [CH13, EAB$^+$03]. Basierend auf den Informationen der Topologie können einzelne Knoten verwendet werden, welche für einen bestimmten Teilbereich der Topologie zuständig sind.

5.5.1 Deterministische Verteilung einer Nachricht in MINHTON

Die Verteilung einer Multicast- bzw. Broadcast-Nachricht MINHTON basiert auf der Verwendung der Routingtabelle (RT) in Anlehnung an die Weiterleitung einer Nachricht über das CDS auf einem Level l (vgl. Abschn. 5.4.3) [EAB+03]. Da es sich bei MINHTON um einen Baum handelt, welcher mehrere Level haben kann, muss eine Verteilung der Nachricht sowohl horizontal als auch vertikal erfolgen.

In Algorithmus 3 ist der initiale Aufruf eines Knotens dargestellt, bei dem eine Nachricht innerhalb des Baumes an alle Knoten verteilt werden soll. Zu diesem Zeitpunkt hat der initiale Knoten keine Kenntnis darüber, ob der Baum eine Höhe von $h-1$ oder $h-2$ hat, sodass zunächst als höchstes Level ∞ angenommen wird.

Algorithmus 3: publish(msg)

```
1 broadcast(msg, initialNode, self, 0, 0, ∞);
```

In Algorithmus 4 ist das vereinfachte Verfahren ohne Betrachtung von Sonderfällen zur Verteilung einer einzelnen Nachricht dargestellt. Zunächst wird geprüft, ob der initiale Sendeknoten, sofern es sich um diesen handelt, um den Baum vertikal in zwei Hälften aufgeteilt werden kann (Z. 4–15). Hierfür wird das eigene Level sowie das Level der (sofern vorhandenen) Adjazenten ausgewertet (Z. 5) und, wenn möglich, die Höhe gleichmäßig aufgeteilt werden (vgl. Abb. 5.21). Nachfolgend sind dann beide Knoten, sowohl der initiale Sendeknoten als auch sein Adjazent-Link, für die Verteilung der Nachricht innerhalb dieser Grenzen zuständig. Befindet sich der Knoten in der oberen Hälfte des Baumes, so ist dieser für die Teilbäume zwischen dem Wurzelknoten (sofern der initiale Sender nicht der Wurzelknoten ist) und der Hälfte der Baumhöhe zuständig. Befindet sich der Knoten in der unteren Hälfte des Baumes, so ist dieser für verbleibende Hälfte zuständig. Diese umfasst unter anderem das höchste Level und ein Level über der anderen Hälfte.

Die vertikale Verteilung in den Hälften erfolgt jeweils durch den Elternknoten und die Kinderknoten (Z. 17–19). So schickt der erste Knoten, welcher die Nachricht auf einem Level erhält, dies an seinen Elternknoten. Die Verteilung der Nach-

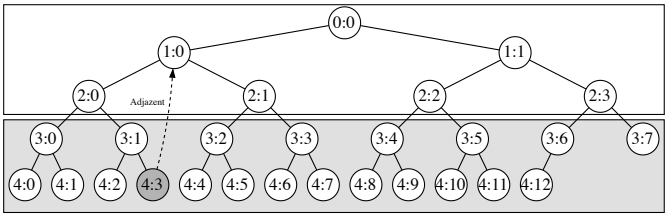


Abb. 5.21: Vertikale Aufteilung des Baumes für ein Netzwerk mit N = 28 Knoten und Fanout m = 2. Der initiale Sendeknoten verteilt maximal bis Level 3. Dessen Adjazent verteilt die Nachricht zwischen Level 0 und 2.

Algorithmus 4: broadcast(msg, initialNode, lastNode, hop, fwdLowerLimit, fwdUpLimit)

```
1  isInitialSender ← (initialNode == self);
2  receivedFromBelow ← self.Level < lastNode.Level;
3  receivedFromAbove ← self.Level > lastNode.Level;
4  if isInitialSender then
5      if abs(adjacent.Level - self.Level)>1 then
6          if adjacent.Level < self.Level then
7              fwdLowerLimit ← 0;
8              fwdUpperLimit ← floor((self.Level + adjacent.Level)/2);
9          else
10             fwdLowerLimit ← floor((self.Level + adjacent.Level + 1)/2);
11             fwdUpperLimit ← ∞;
12         end
13         forwardToAdjacent(msg, initialNode, self, hop+1, fwdLowerLimit, fwdUpperLimit);
14     end
15 end
16 if firstNodeReceivedOnLevel() then
17     calcInLevelIntervalAndForwardInLevel(msg, initialNode, self, hop+1);
18     if receivedFromAbove then
19         forwardToLeftMostAndRightMostChildren(msg, initialNode, self, hop+1, fwdLowerLimit,
             fwdUpperLimit);
20     end
21     if receivedFromBelow then
22         forwardToParent(msg, initialNode, self, hop+1, fwdLowerLimit, fwdUpperLimit);
23     end
24 else
25     calcInLevelIntervalAndForwardInLevel(msg, initialNode, self, hop+1);
26     if weAreFirstNodeOnThisLevelWithChildren() then
27         forwardToLeftMostAndRightMostChildren(msg, initialNode, self, hop+1, fwdLowerLimit,
             fwdUpperLimit);
28     end
29 end
```

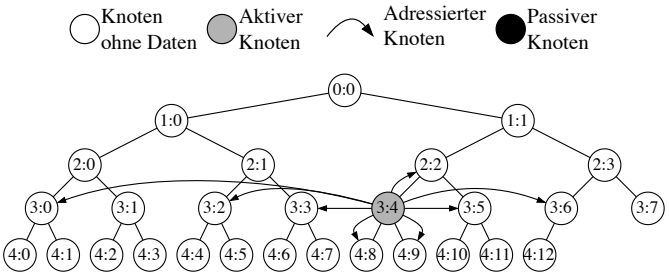
richt in Richtung der höchsten Level erfolgt über die zwei äußersten Kinderknoten. Das nächsthöhere Level wird immer nur von einem einzelnen Knoten betreten. Da jedoch an die zwei äußersten Kinderknoten die Nachricht geschickt wird und beide Kinderknoten eine Verbindung zueinander wissen, leitet der Knoten die Nachricht auf das nächste Level weiter, welcher sich mehr zur Mitte des Levels befindet ($\frac{m^L}{2}$).

Der initiale Sender bzw. der Knoten der erstmalig auf auf einem Level die Nachricht erhält, ist für die horizontale Verteilung dieser zuständig (Z. 17 bzw. Z. 21). Dieser Knoten, teilt abhängig von seiner Position n im Level, dieses in unabhängige Intervalle L und R auf. Hierbei hat jedes Intervall eine Untergrenze (b_{low}) und eine Obergrenze (b_{up}). Das erste Intervall befindet sich links und deckt die Positionen $L = [0, n-1]$ ab, das andere Intervall befindet sich rechts und deckt die Positionen der Knoten $R = [n+1, m^l - 1]$ ab. Basierend auf dem Intervall und den Einträgen in der RT werden nun alle Knoten selektiert. Zunächst berechnet ein Knoten die Menge aller Knoten, zu denen eine Verbindung in seiner RT und seinem zugehörigen Intervall existiert, gegeben durch $S_R = \{s \in RT \mid s \in Pos_m(l) \wedge b_{low} \leq s \leq b_{up}\}$. Jeder Knoten, welcher diese Nachricht in dem Intervall erhält, kann basierend darauf ein neues Intervall für jeden $s_j \in S_R$ berechnen und so die Anfrage weiterleiten. Sei $S_R = \{s_1, s_2, \ldots, s_{i-1}, s_i\}$ die Menge an Knoten in R. Dabei ist S_R sortiert nach $s_1 < s_2 < \cdots < s_{i-1} < s_i$. Das zugeordnete Intervall für s_j mit $j = 1, \ldots, i-1$

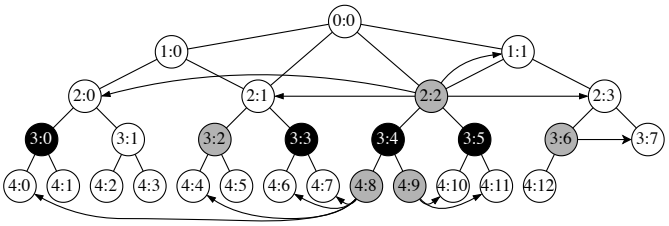
ist $R_{s_j} = [s_j + 1, s_{j+1} - 1]$ und $R_{s_i} = [s_i + 1, b_{up}]$. Außerdem ist $L_{s_i} = \emptyset$ für alle $s_j \in S_R$. Ein Knoten leitet die Nachricht an jeden Knoten $s_j \in S_R$ mit den entsprechenden (Teil-)Intervallen R_{s_j} und L_{s_j} weiter. Dies wird so oft wiederholt, bis kein Knoten mehr innerhalb eines zugewiesenen Intervalls vorhanden ist, d. h. $S_R = \emptyset$. Die Weiterleitung der Nachricht an das linke Intervall L ist äquivalent. Das erste Kind ganz links verteilt die Nachricht im Intervall $[0, Pos_{erstesKind}]$, während das letzte Kind ganz rechts die Nachricht im Intervall $[Pos_{letztesKind}, m^l - 1]$ verteilt. In einem Baum mit Fanout m hat jeder Knoten, sofern Knoten an diesen Positionen existieren, $m-1$ Verbindungen nach links und nach rechts. Folglich sind alle Kinderknoten des Elternknotens hierdurch abgedeckt. Um eine Überschneidung zwischen den äußersten Knoten zu vermeiden, ist derjenige für die verbleibenden $m-2$ Knoten zuständig, welcher die kleinere Menge an Knoten in seinem initialen (Teil-)Intervall auf dem Level besitzt.

In Abb. 5.22 ist das Verfahren zur Verteilung einer Nachricht im Netzwerk visuell dargestellt. Das Netzwerk besteht aus 28 Knoten und verwendet dabei Fanout $m = 2$. Der initiale Sender ist hierbei Knoten (*3:4*), dessen Adjazenten sich auf dem höchsten Level $h-1$ befinden. Daher wird der Baum nicht vertikal aufgeteilt, sodass in der ersten Runde der initiale Knoten (*3:4*) allen Einträgen aus seiner RT die Nachricht sendet (vgl. Abb. 5.22a). Hier werden auch Knoten sowohl aus dem höheren Level 4 (Knoten (*4:8*) und (*4:9*)) als auch aus dem niedrigeren Level 2 (Knoten (*2:2*)) adressiert. Basierend darauf werden, in Abhängigkeit von den berechneten Intervallen, nur noch selektive Knoten adressiert. Dabei leitet der Knoten, welcher als erstes auf einem bisher neuen Level adressiert wird, die Nachricht horizontal und vertikal weiter, sofern möglich (vgl. Abb. 5.22b). In Runde 3 (vgl. Abb. 5.22c) existiert nur noch eine niedrige Anzahl an verbleibenden Knoten, welche die Nachricht nicht erhalten haben. In Runde 4 (vgl. Abb. 5.22d) existiert lediglich ein Knoten (*4:1*), welcher die Nachricht noch nicht erhalten hat. In einem anderen Szenario mit Knoten (*3:3*) als initialem Sender entspricht die maximale Anzahl an Runden 3. Ist Knoten (*3:7*) der initiale Sendeknoten, so würden 5 Runden benötigt werden. Die Verteilung auf Level 3 benötigt 3 Level. Damit Knoten (*4:7*) die Nachricht erhält, werden ausgehend von (*3:4*) die Knoten (*3:6*), (*4:12*), (*4:4*) und (*4:2*) benötigt.

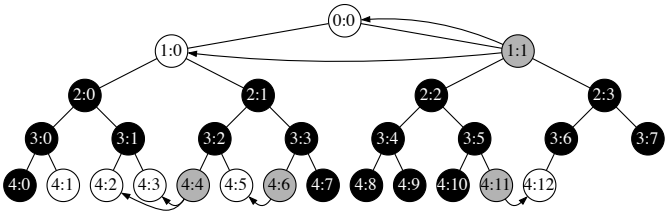
Aus Sicht des initialen Senders wird ein ein gerichteter Graph ohne Zyklus (engl.: *Directed Acyclic Graph (DAG)*) erstellt. Zwar besteht die Möglichkeit, aufgrund der Vernetzung einen Zyklus einzuführen, dieser wäre aber lediglich im Fehlerfall von Bedeutung, wenn der Fluss der Daten unterbrochen und eine alternative Route benötigt wird.



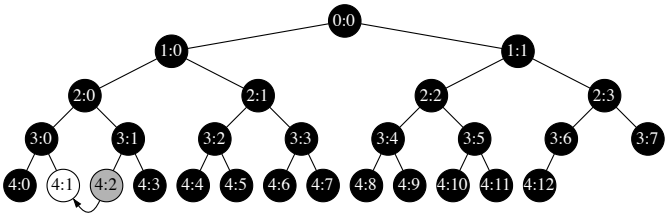
(a) Runde 1 – Knoten (*3:4*) schickt die Nachricht an alle seine Einträge in seiner Routingtabelle ((*3:0*), (*3:2*), (*3:3*), (*3:5*), (*3:6*), (*2:2*), (*4:8*) und (*4:9*)).



(b) Runde 2 – Alle adressierten Knoten aus Runde 1 leiten die Nachricht weiter, sofern sich Knoten zwischen ihnen selbst und den Knoten aus Runde 1 befinden ((*3:2*), (*3:6*), (*4:8*), (*4:9*) und (*2:2*)).



(c) Runde 3 – Alle adressierten Knoten aus Runde 2 leiten die Nachricht weiter, sofern sich Knoten zwischen ihnen selbst und den Knoten aus Runde 2 befinden. Dies gilt u. a. für Knoten ((*4:4*), (*4:6*), (*4:11*) und (*1:1*)).



(d) Runde 4 – Alle adressierten Knoten aus Runde 3 leiten die Nachricht weiter, sofern sich Knoten zwischen ihnen selbst und den Knoten aus Runde 3 befinden. Dies gilt lediglich für Knoten (*4:2*).

Abb. 5.22: Beispiel der epidemischen Ausbreitung einer Nachricht in MINHTON für ein Netzwerk mit Fanout $m = 2$ und $N = 28$ Knoten

5.5.2 Die Zuverlässigkeit des Verfahrens in Abhängigkeit von der Paketverlustrate

Die Zuverlässigkeit (engl.: *reliability (R)*) ist das Verhältnis der Knoten, welche die Nachricht erhalten haben, zu der Gesamtanzahl der Knoten im Netzwerk [KMG03]:

$$R = \frac{\text{Anzahl der Knoten, welche dieNachricht erhalten}}{\text{Gesamtanzahl der Knoten im Netzwerk}} \tag{5.19}$$

Die Zuverlässigkeit hängt von der Wahrscheinlichkeit ab, dass ein Nachbarknoten die Nachricht erhält (vgl. Abb. 5.23). Einem Knoten, welcher als initialer Sender einer Nachricht betrachtet wird, liegt die Nachricht vor. Das entspricht der Wahrscheinlichkeit 1, dass der initiale Sender die Nachricht empfängt, sodass die Anzahl an Knoten auf Level 0 folglich $M_R(0) = 1$ entspricht. Unter der Annahme, dass ein Paket mit der Wahrscheinlichkeit P_v verloren geht, empfängt ein Nachbarknoten vom initialen Sender dieses mit $P = 1 - P_v$. In einem Baum mit Fanout $m \geq 2$ hat ein Knoten Verbindungen zu allen Nachbarn in seinem Teilbaum. Somit ist die Anzahl an Knoten M auf Level 1, welche die Nachricht im Teilbaum erhalten, gegeben durch $M_R(m,1) = 1 + (m-1) \cdot P$. Hier wird angenommen, dass auf jedem Level mindestens 1 Knoten ein Paket mit der Wahrscheinlichkeit von 1 erhält.

In einer nullbalancierten vollständigen Baumstruktur lässt sich die Verteilung einer Nachricht basierend auf den Wahrscheinlichkeiten der beiden vorherigen Teilbäume auf den tieferen Leveln beschreiben. Als Grundlage gilt die Baumstruktur aus Abb. 5.23. Die Anzahl Knoten, welche die Nachricht erhalten bzw. die Verteilung zwischen den Knoten auf Level 2 im linken Teilbaum entspricht der Wahrscheinlichkeit im rechten Teilbaum bzw. der Wahrscheinlichkeit der Verteilung einer Nachricht zwischen zwei Nachbarknoten auf Level 1. Folglich lässt sich die Verteilung der Informationen auf Level > 1 als Teilproblem der darunter liegenden Level bzw. Teilbäume wie in Gl. 5.20 beschreiben:

$$\begin{aligned} M_R(0) &= 1 \\ M_R(m,1) &= 1 + (m-1) \cdot P \\ M_R(m,l) &= M_R(l-1) + M_R(l-1) \cdot (m-1) \cdot P \end{aligned} \tag{5.20}$$

Jeder Knoten kann eine Nachricht über seine RT verteilen. Diese kann sowohl im eigenen Teilbaum als auch in anderen Teilbäumen erfolgen. Die Wahrschein-

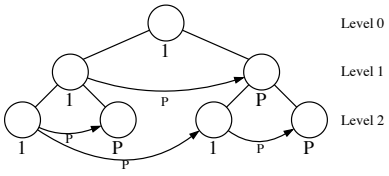


Abb. 5.23: Die Wahrscheinlichkeit, dass ein Knoten ein Paket bzw. eine Nachricht empfängt

lichkeit, dass ein Knoten in einem Teilbaum eine Nachricht erhält, steigt durch eine Sendewiederholung (engl.: *retransmission*) auf $1 - P_v^{RET}$. Basierend auf Gl. 5.20 und der Anzahl an Sendewiederholungen kann ein theoretisches Modell für die Zuverlässigkeit der Verteilung einer Nachricht auf einem Level l in Abhängigkeit der Paketverlustrate erstellt werden:

$$P_R(m,l) = \frac{(1 - P_v^{RET})^l}{m^l} \tag{5.21}$$

In einem Netzwerk mit einer Paketverlustrate $P_v = 15\,\%$ und einer Senderwiederholungsrate von $RET = 2$ erhöht sich die Wahrscheinlichkeit für eine erfolgreiche Zustellung aller Pakete von $P = 85\,\%$ auf $P = 97,5\,\%$. Die nachfolgende Abb. 5.24 zeigt das theoretische Modell aus Gl. 5.21 für die Zuverlässigkeit einzelner Level l in Abhängigkeit von der Paketverlustrate und den Sendewiederholungen. Sendewiederholungen spiegeln die Anzahl an Nachrichten wider, also wie oft ein einzelner Knoten dieselbe Nachricht verschickt. Dabei wird nicht berücksichtigt, wie oft ein einzelner Knoten eine Nachricht erhält.

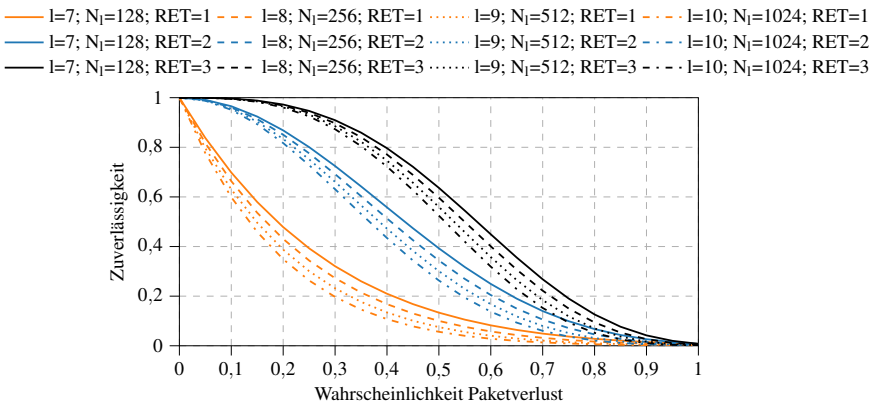


Abb. 5.24: Theoretisches Modell der durchschnittlichen Zuverlässigkeit für verschiedene Level (l) in einem Netzwerk mit Fanout $m = 2$ und der Erhöhung der Sendewiederholungen (RET)

Zwar steigt die Zuverlässigkeit mit Erhöhung der Sendewiederholungen, allerdings steigt dadurch auch die Anzahl redundanter Nachrichten. Existieren auf einem Level 128 Knoten und versendet jeder Knoten 2 Nachrichten, so werden bei der Verteilung der Nachricht insgesamt $128 * 2 = 256$ Nachrichten versendet.

5.5.3 Analytische Betrachtung

Im folgenden Abschnitt wird das zuvor vorgestellte Verfahren analytisch betrachtet. Als Metrik dient hierfür die Anzahl an Runden (engl.: *last delivery hop*). Diese misst die Anzahl an Runden bzw. Sprüngen, die benötigt wird, bis eine Broadcast-Nachricht an alle Teilnehmer verteilt ist [LPR07]. In anderen Worten entspricht die Anzahl an Runden der Anzahl der Weiterleitungen einer Nachricht. Bei der ersten Weiterleitung wird der Wert daher auf 1 gesetzt. Die Verteilung einer Nachricht in MINHTON beinhaltet sowohl eine horizontale als auch eine vertikale Verteilung, welche sich wie folgt aufteilt:

- **Vertikal**: Die vertikale Verteilung erfolgt über die Verwendung der Verbindung zu den Kinderknoten oder dem Elternknoten. In jeder Runde wird ein weiteres Level betreten. Hieraus resultiert, dass mindestens Sprünge gleich der Baumhöhe $(h-1)$ benötigt werden, sodass jedes Level und folglich der gesamte Baum abgedeckt ist. Ist die Differenz der Level zwischen einem Knoten und einem seiner Adjazenten > 1, so wird der Baum geteilt. Daher werden innerhalb der verbleibenden Baumhälften maximal $\lceil {}^{\text{level}}/_{2} \rceil$ Sprünge benötigt.
- **Horizontal**: Eine Nachricht in einem vollen Level l weiterzuleiten benötigt $(h-1)$ Sprünge, bzw. die Anzahl an Weiterleitungen entspricht dem aktuellen Level l (vgl. Abschn. 5.12). Innerhalb dieser Sprünge können alle Knoten adressiert werden durch die Tatsache, dass ein nullbalancierter, vollständiger Baum aufgebaut wird (vgl. Abschn. 5.2.1). Folglich ist die Anzahl der benötigten Sprünge auf einem Level l gleich dem Level l.

In Tabelle 5.9 ist eine Übersicht einiger Extremfälle bei der Verteilung einer Nachricht innerhalb von MINHTON für den Fanout $m = 2$ dargestellt. Die ersten beiden Extremfälle spiegelt ein Netzwerk aus zwei Knoten wider. Unabhängig davon, ob der Wurzelknoten (Extremfall 1) oder das erste Kind (Extremfall 2) sendet, entspricht die benötigte Anzahl an Runden 1. Der dritte Extremfall, bezogen auf Abb. 5.22, würde bedeuten, dass es sich bei dem initialen Sender um Knoten (*4:0*) handelt. Die Anzahl der benötigten Runden für die horizontale Verteilung entspricht auf dem initialen Level die des initialen Senders (4). Die Anzahl der Runden für die vertikale Verteilung entspricht $(h-1)$, denn der vertikale Pfad bis zum Wurzelknoten erstreckt sich über den jeweiligen Elternknoten, welcher sich ebenfalls immer ganz links an Position 0 befindet. Mit jedem Sprung auf ein tieferes Level wird ein Sprung für die horizontale Verteilung weniger benötigt. Folglich entspricht dies so maximal $(h-1)$ Runden.

Im vierten Extremfall, exemplarisch bezogen auf Abb. 5.22, wird Knoten (*4:12*) als initialer Sender angenommen. Durch die Verwendung des linken Adjazenten wird das Level 1 im ersten Sprung erreicht, über den Elternknoten (*3:6*) das Level 3. Zwischen dem initialen Sender und dem Adjazenten wird der Baum vertikal aufgeteilt, sodass der Knoten (*1:1*) neben Level 0 auch für das Level 2 zuständig ist. Der initiale Sender ist nur noch für die vertikale Verteilung bis einschließlich Level 3 zuständig. Bezogen auf diesen Baum dominiert die horizontale Verteilung,

sodass auch hier maximal $h-1$ Sprünge benötigt werden bis alle Knoten im Baum die Nachricht erhalten haben.

Im fünften Fall, bei dem das höchste Level $(h-1)$ mehr als zur Hälfte gefüllt ist, bedeutet, dass das höchste Level $(h-1)$ in zwei Sprüngen erreicht werden kann. Dies liegt an der Tatsache, dass jeder Knoten auf Level $(h-2)$ eine Verbindung zu einem Knoten in der anderen Hälfte des Teilbaums besitzt, weil die gesamte RRT und LRT gefüllt sind. Konkret bedeutet das, dass die initial gesendete Nachricht von Knoten (*3:7*) über den Knoten (*3:6*) an den Knoten (*4:12*) auf Level $(h-1)$ weitergeleitet wird. Würden die Knoten (*4:9*) bis (*4:12*) auf dem Level 4 nicht existieren, so würde die Weiterleitung der Nachricht auf Level 4 über den Knoten (*3:7*) erfolgen. Somit lässt sich zu jedem Zeitpunkt für den Fanout $m = 2$ in zwei Sprüngen erreichen. Die horizontale Verteilung auf dem Level $(h-1)$ ist sein Teilproblem des darüber liegenden Levels.

Im sechsten Extremfall erolgt eine nahezu vollständige horizontale Verteilung auf dem zweithöchsten Level $(h-2)$, bevor dass höchste Level $(h-1)$ erreicht werden kann. Hieraus folgt, je später das höchste Level $(h-1)$ erreicht wird, desto weniger Runden werden auf diesem Level benötigt. Dieser Extremfall ist auch in Abb. 5.25 vereinfacht dargestellt. In diesem Fall entspricht die benötigte Anzahl an Runden, bis alle Knoten die Nachricht erhalten haben, $\leq 2 \cdot (h-1) - InitialerSender_{Level}$ Runden. Die Weiterleitung von Level $(h-2)$ ins Level $(h-1)$ erfolgt durch den Knoten, welcher zuerst die Nachricht erhält und ein Kind auf dem höheren bzw. in diesem Fall auf dem höchsten Level $(h-1)$ besitzt.

Die Extremfälle 7, 8 und 9 beziehen sich auf einen perfekten Baum, in dem alle Nullknoten auf dem höchsten Level $(h-1)$ befinden. Die vertikale Verteilung in Extremfall 7 erfolgt, nachdem der Wurzelknoten an seine beiden jeweiligen Adjazenten auf dem höchsten Level $(h-1)$ gesendet hat, jeweils nur bis zur Hälfte des Baumes. Bei dem Fanout $m = 2$ entspricht auf dem höchsten Level $(h-1)$ der Anzahl an Sprüngen wie auf einem Level tieferen Level $(h-2)$, sodass die Verteilung auf einem niedrigeren Level als ein Teilproblem des höheren Levels angesehen werden kann.

Initiale Sendeknoten, deren Adjazenten-Verbindungen dem Elternknoten oder einem Kinderknoten entsprechen bzw. bei dem nur ein Adjazent existiert, führen dazu, dass der Baum nicht vertikal geteilt werden kann. So ist für den Extremfall 8 der Knoten ganz links an Position 0 bzw. für den Extremfall 9 der Knoten auf dem Level an Position $m^{h-1} - 1$ ein Beispiel. Auch hier folgt, Analog zu Extrem Fall 3, eine vertikale Verteilung über den jeweiligen Elternknoten. Horizontal entspricht die Anzahl an Sprüngen dem jeweiligen Level l bzw. $h-1$.

Tabelle 5.9: Übersicht einiger Extremfälle und die maximal benötigte Anzahl an Runden für die Verteilung einer Nachricht in MINHTON und $m = 2$

Nr.	Extremfall	Initialer Sender	Maximale Anzahl an Runden	Anmerkung
1	$N = 2$	0:0	1	sendet an einziges Kind
2		1:0	1	sendet an Elternknoten
3	N	höchstes Level $(h-1)$, ganz linker Knoten	$(h-1)$ bzw. Level des initialen Senders	höchstes Level $(h-1)$ ist nicht komplett gefüllt
4	N	höchstes Level $(h-1)$, ganz rechter Knoten	$\leq (h-1)$ bzw. Level des initialen Senders	
5	N	Nullknoten auf Level $(h-2)$	$\leq h$	höchstes Level $(h-1)$ ist mehr als zur Hälfte gefüllt
6	N	Nullknoten auf Level $(h-2)$	$\leq 2 \cdot (h-1) -$ Level des initialen Senders	höchstes Level $(h-1)$ ist weniger als zur Hälfte gefüllt
7	Perfekter Baum	Wurzelknoten	$(h-1)$	Alle Nullknoten sind auf dem höchsten Level $(h-1)$
8		höchstes Level $(h-1)$, ganz linker Knoten	$(h-1)$ bzw. Level des initialen Senders	
9		höchstes Level$(h-1)$, ganz rechter Knoten	$(h-1)$ bzw. Level des initialen Senders	

h : Baumhöhe – In einem Baum der Höhe h befinden sich Knoten höchstens auf Level $(h-1)$.

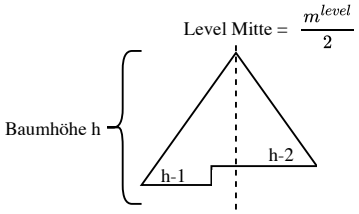


Abb. 5.25: Das höchste Level $h-1$ ist weniger als zur Hälfte gefüllt.

5.5.4 Simulative Evaluation

In diesem Abschnitt wird die Broadcast-Operation für MINHTON anhand von Simulationsergebnissen verifiziert. Dabei soll das untersuchte System quantitativ bewertet und die durchgeführte theoretische Analyse (vgl. Abschn. 5.5.3) ergänzt werden. Die Simulationsexperimente wurden mit dem diskreten Ereignis-Netzwerksimulator ns-3 [RH10] durchgeführt. In Tabelle 5.10 sind zwei unterschiedliche experimentelle Setups (ES) und ihre Parametrisierung dargestellt. Für jedes Experiment wurde 10-mal – jeweils mit verschiedenen Startwerten für einen Zufallsgenerator – durchgeführt und für die nachfolgenden Abbildungen gemittelt. Zu den Merkmalen, die von Interesse sind, gehört neben dem generierten

Verkehr (Anzahl der ausgetauschten Nachrichten) für die Verteilung einer Nachricht an alle Knoten in MINHTON auch die benötigte Anzahl an Runden. Es wurden Netzwerke mit unterschiedlichen Größen N von 1.000 bis 10.000 Knoten und mit unterschiedlichen Fanouts ($m \in \{2, 4, 8, 16\}$) simuliert.

Tabelle 5.10: Übersicht der Parametrisierung der experimentellen Setups (ES)

Nr.	Parameter	ES1	ES2
1	Fanouts $m \in \mathbb{N}$	$m \in \{2, 4, 8, 16\}$	
2	Netzwerkgröße $N \in \mathbb{N}$	$N \in \{1k, 2k, 3k, \ldots, 10k\}$	
3	Anzahl der Sender	1000	N
4	Anzahl der sendenden Knoten	randomisierte Verteilung	sequenziell
5	Anzahl an Wiederholungen	10	

Anzahl an benötigten Runden (Last Delivery Hop)

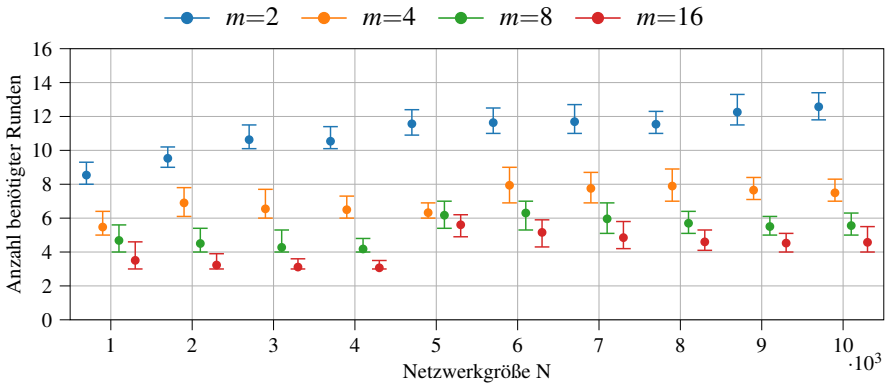
In Abb. 5.26 sind für Experimentelles Setup (ES)1 und ES2 die minimale, die maximale und die durchschnittliche Anzahl an Runden dargestellt, die benötigt werden, um eine Nachricht im gesamten Netzwerk für unterschiedliche Netzwerkgrößen N und Fanouts m zu verteilen. Während in Abb. 5.26a zufällig 1.000 Knoten ausgewählt wurden, hat in Abb. 5.26b jeder Knoten einmal eine Broadcast-Nachricht initiiert. Daher beinhaltet Abb. 5.26b auch die tatsächlichen Minima und Maxima die in einem Netzwerk der Größe N mit Fanout m auftreten können. Durch die Erhöhung des Fanouts steigt die Anzahl der Links in der RT, sodass die Anzahl an benötigten Runden sinkt.

Zu beachten ist: Je voller das höchste Level $h-1$ gefüllt ist, desto weniger Runden werden benötigt für eine Verteilung einer Broadcast-Nachricht. Dies wird besonders deutlich für Fanout $m = 16$ und die Netzwerkgrößen zwischen 1.000 und 4.000. Während bei der Netzwerkgröße $N_{16} = 1.000$ lediglich 727 von 4096 möglichen Knoten auf dem höchsten Level $h-1$ vorhanden sind, befinden sich bei der Netzwerkgröße $N_{16} = 4.000$ lediglich 3.727 Knoten auf dem höchsten Level $h-1$. In Abb. 5.26a sinkt die minimale und die maximale Anzahl an Runden sowie der Durchschnitt, bei der zufällig der initiale Sender ausgewählt wurde. In Abb. 5.26b hingegen sinkt lediglich der Durchschnitt, da in diesem experimentellen Setup jeder Knoten einmal der initiale Sender ist. Festzuhalten ist, dass mit steigendem Fanout die Anzahl an benötigten Runden sinkt und sich die Anzahl an Runden erhöht, sobald ein neues Level eröffnet wird. Zusätzlich sinkt die durchschnittlich benötigte Anzahl an Runden, je voller das höchste Level in einem Baum ist. Dennoch ist festzuhalten, dass die Position des initialen Senders und dessen Verbindungen Einfluss haben. In Tabelle 5.11 sind einige Beispiele für Knoten als initiale Sender darge-

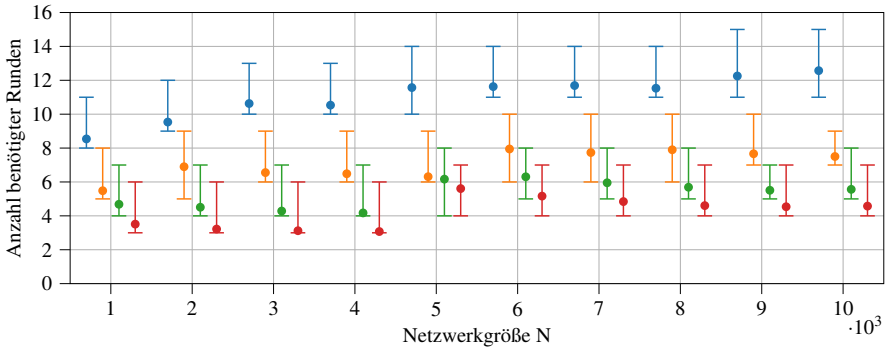
stellt, bei denen das Maximum erreicht wird. Diest hängt u. a. damit zusammen, ob ein Knoten einen Adjazenten auf einem Level besitzt, welcher ungleich dem Level seines Elternknotens oder einem seiner Kinder entspricht.

Anzahl an ein- und ausgehenden Nachrichten für einen Peer

In Abb. 5.27 ist die Anzahl an ein- und ausgehenden Nachrichten für einen einzelnen Peer dargestellt, für das ES2 in Abhängigkeit vom Fanout m und der NetzwerkgrößeN. In Abb. 5.27a darf das Minimum von 1 nicht unterschritten werden. Denn die Knoten in der letzten Runde erhalten lediglich die Nachricht, leiten diese aber an keine weiteren Knoten weiter. Beispiele hierfür sind der letzte Knoten in einem Intervall oder der ganz linke bzw. der ganz rechte Knoten auf einem Level. Dies gilt allerdings nur, sofern es sich hierbei nicht um den initialen Sender handelt.



(a) ES1 – Zufällige Auswahl des initialen Senders einer Broadcast-Nachricht



(b) ES2 – Jeder Knoten ist einmal der initiale Sender einer Broadcast-Nachricht

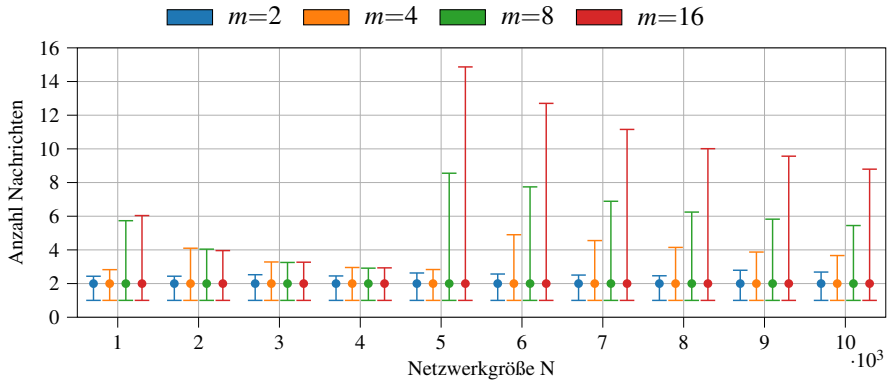
Abb. 5.26: Minimale, maximale und durchschnittlich Anzahl an Runden, die benötigt werden, bis alle Knoten eines Netzwerks der Größe N und Fanout m die Broadcast-Nachricht erhalten haben

Tabelle 5.11: Beispiele für initiale Sender, bei denen die maximale Anzahl an Runden benötigt in MINHTON für verschiedene Fanouts m

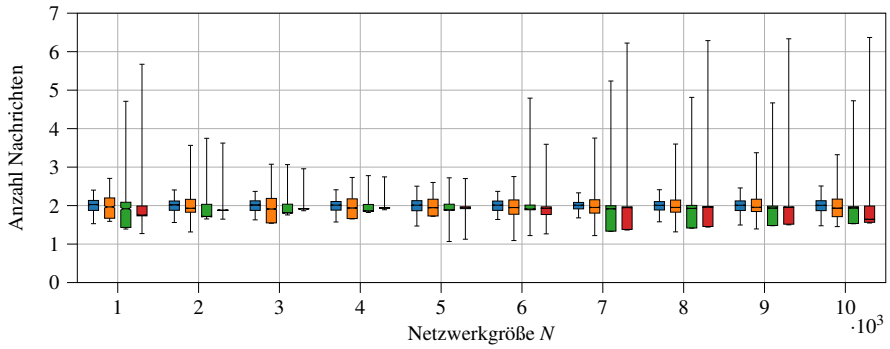
Netzwerkgröße N	$m = 2$	$m = 4$	$m = 8$	$m = 16$
1.000	(4:15)	(1:3)	(0:0)	(0:0)
2.000	(4:15)	(0:0)	(0:0)	(0:0)
3.000	(4:15)	(0:0)	(1:5)	(1:11)
4.000	(4:15)	(1:3)	(1:7)	(1:15)
5.000	(5:31)	(2:14)	(0:0)	(0:0)
6.000	(4:15)	(0:0)	(0:0)	(0:0)
7.000	(1:1)	(1:1)	(0:0)	(0:0)
8.000	(4:15)	(0:0)	(0:0)	(0:0)
9.000	(5:31)	(0:0)	(0:0)	(1:1)
10.000	(5:31)	(0:0)	(1:1)	(1:1)

Abb. 5.27b umfasst dabei den Median (Kerbe) und das obere bzw. das untere Quantil (Boxen). Der obere Whisker entspricht dem 97,5 % Quantil und der untere Whisker dem 2,5 % Quantil. Somit decken die Whisker 95 % aller Knoten in dem Netzwerk ab. Die Ausreißer, welche ebenfalls die maximale Anzahl an Nachrichten die ein Knoten versenden muss, sind nicht dargestellt. In Abb. 5.27c ist lediglich die maximale Anzahl an Nachrichten dargestellt, welche ein Knoten in dem experimentellen Setup versenden muss. Die meisten Knoten innerhalb des Netzwerks haben eine Grundlast von ca. 2 Nachrichten, wenn eine Nachricht im Netzwerk mithilfe des Broadcasts verteilt wird. Liegt der Median bzw. der Durchschnitt unterhalb von 2, so bedeutet dies, dass die meisten Knoten lediglich empfangen und keine Nachricht versenden. Mit der Erhöhung des Fanouts erhöht sich auch die Anzahl an Links in einer RT. Daher kann ein Knoten, welcher die Daten in einer frühen Runde erhält, an viele Knoten innerhalb seiner RT und der Intervalle versenden.

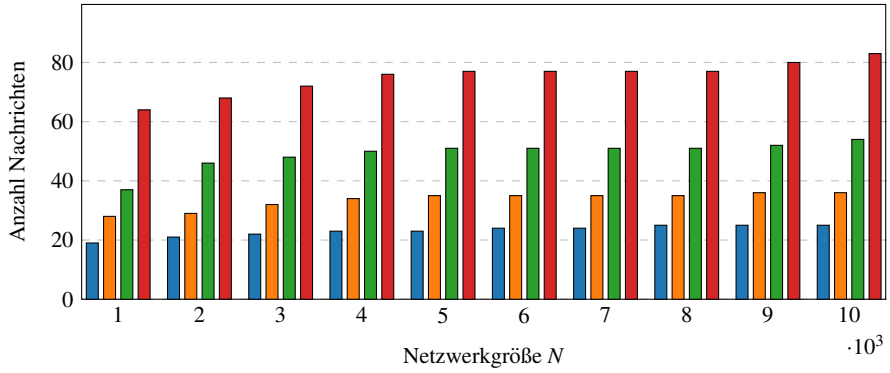
In Abb. 5.27c ist die maximale Anzahl an Nachrichten dargestellt, welcher ein Knoten bei der Verteilung einer Broadcast-Nachricht innerhalb des Netzwerks versenden muss. Da mit zunehmender Runde die Anzahl an verbleibenden Knoten geringer wird, welche die Nachricht noch nicht erhalten haben, wird die maximale Anzahl an Nachrichten primär in den ersten Runden erreicht. Auch hier gilt, dass mit der Erhöhung des Fanouts die maximale Anzahl an Links innerhalb einer RT steigt, jedoch auch stark von der Position innerhalb des Baumes abhängt. Ein Knoten, welcher innerhalb des Levels, z. B. an der Position $m^l/2$ liegt hat Links in seiner RRT und LRT. Ein Knoten ganz links auf einem Level an Position 0 bzw. ganz rechts auf einem Level $m^l - 1$ hat nur Verbindungen in seiner RRT bzw. LRT. Hinzu kommt noch die Höhe des Baumes. Je höher das Level und der Fanout, desto höher ist der Grad der Vernetzung bzw. die maximale Anzahl an Links, die ein Knoten hat.



(a) Minimale, maximale und durchschnittliche Anzahl an ein- und ausgehenden Nachrichten



(b) Anzahl an ein- und ausgehenden Nachrichten. Der obere Whisker entspricht dem 97,5 % Quantil, untere Whisker dem 2,5 % Quantil



(c) Maximale Anzahl an Nachrichten die ein Knoten zu versenden hat

Abb. 5.27: Anzahl an ein- und ausgehenden Nachrichten je Knoten in Abhängigkeit von der Netzwerkgröße N und dem Fanout m für ES2

Ausbreitung einer Nachricht

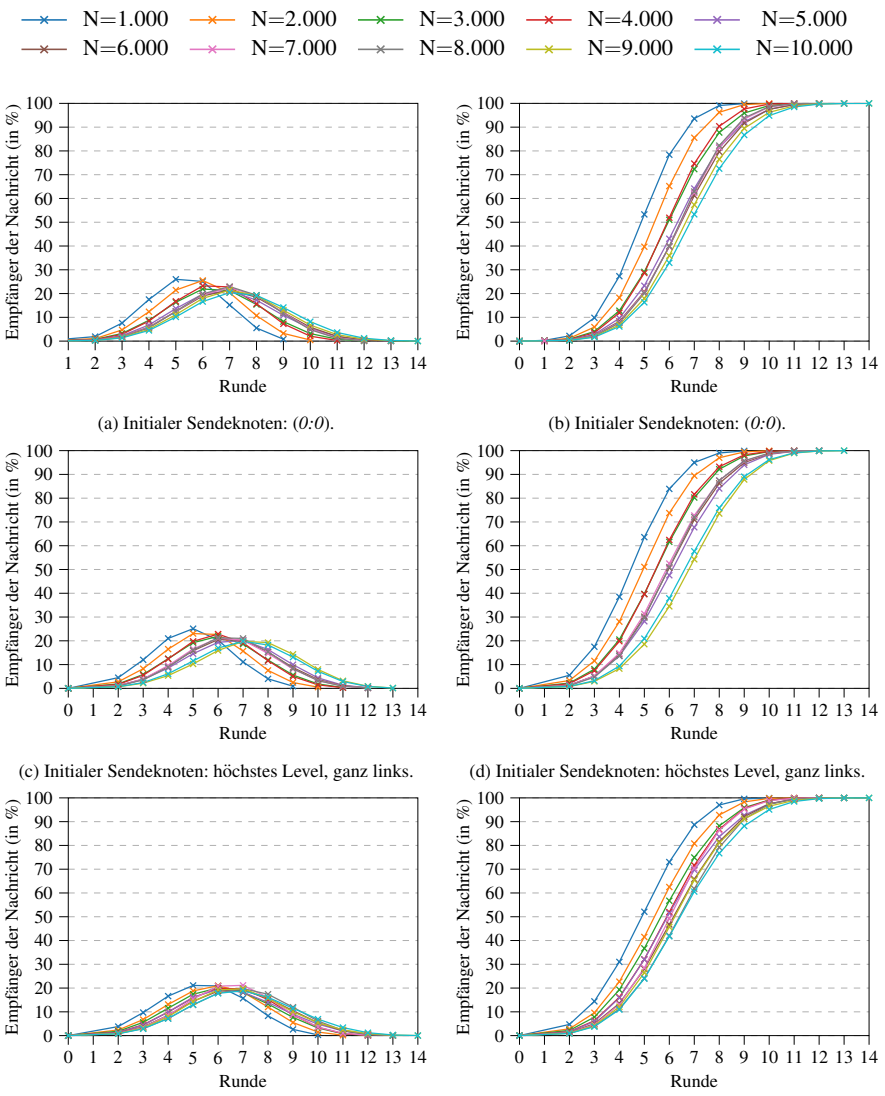
Im Folgenden wird nun betrachtet, wie sich eine Nachricht im Netzwerk verteilt. In Abb. 5.28 und Abb. 5.29 sind die Ausbreitungsgeschwindigkeiten für Fa-

nouts $m = 2$ und $m = 16$ in Abhängigkeit von der Netzwerkgröße N dargestellt. Die genauen logischen IDs der Knoten und weitere Abb. für Fanouts $m = 4$ und $m = 8$ zur Vollständigkeit sind im Anhang D dargestellt. Im linken Teilbereich der Abb. 5.28 (a, c und e) bzw. in Abb. 5.29 (a, c und e) ist die prozentuale Anzahl an Knoten je Runde dargestellt, die Daten erhalten. Während für Fanout $m = 2$ sich das Ausbreiten fast gleich verteilt, so ist die Anzahl an erreichten Knoten pro Runde für Fanout $m = 16$ größer. Für Fanout $m = 2$ wächst die Anzahl an verfügbaren Einträgen bzw. Nachbaren in der RT wesentlich kleiner als für Fanout $m = 16$.

In Abb. 5.29a ist der Wurzelknoten der initiale Sender. Dieser teilt den Baum vertikal auf und leitet die Daten an Level 1 und an das höchste $(h-1)$ bzw. zweithöchste Level $(h-2)$ weiter. Welches Level von beiden erreicht wird, hängt von der Höhe des Baumes bzw. des Füllgrades ab. Während zwischen $N = 1.000$ bis $N = 4.000$ lediglich die ersten drei Level gefüllt werden, wird ab dem Knoten 4.369 das vierte Level gefüllt. Die Weiterleitung der Nachricht an einen Adjazenten erreicht hier Level $h-2$, da für einen Sprung auf $h-1$ auf Level 4 mindestens $m^{16}/2$ Knoten existieren müssten (vgl. Abb. 5.25).

Eine fallende Anzahl an Empfängern hängt mit der Erreichung eines neuen Levels zusammen und ebenfalls mit der Fülle des höchsten Levels. Ein starker Anstieg resultiert durch die horizontale Verteilung, bei der bereits Knoten die Nachricht erstmalig erhalten haben und diese in der zweiten horizontalen Runde weiterleiten können. Bei der Betrachtung des Knotens ganz links auf dem höchsten Level als initialer Sender (vgl. Abb. 5.29c) wird mit jeder Weiterleitung ein neues Level betreten, da hier eine vertikale Teilung des Baumes nicht erfolgt und vertikale Verteilung immer über den Elternknoten erfolgt. In diesem Fall kann über die benötigte Anzahl der Runden auch ein Rückschluss auf die Höhe des Baumes erfolgen. Wird nun der Knoten ganz rechts auf dem zweithöchsten Level betrachtet (vgl. Abb. 5.29e), so kann hier kein Rückschluss auf die Höhe erfolgen. Auch hier wird der Baum vertikal nicht geteilt, sodass jeweils die Knoten ganz rechts auf einem Level als erstes die Nachricht erhalten und horizontal nach links zum ersten Knoten verteilen. Eine gesunkene Anzahl an Knoten je Runde resultiert zum einen aus der verbleibenden Anzahl an Knoten auf einem Level, während der horizontalen Ausbreitung, zum anderen kann auch hier ein neues Level eröffnet worden sein.

Im rechten Teilbereich der Abb. 5.28 (b, d und f) bzw. in Abb. 5.29 (b, d und f) sind die kumulierten prozentualen Werte der Ausbreitung dargestellt. Hierbei handelt es sich lediglich um eine andere Form der Darstellung.



(a) Initialer Sendeknoten: (*0:0*).

(b) Initialer Sendeknoten: (*0:0*).

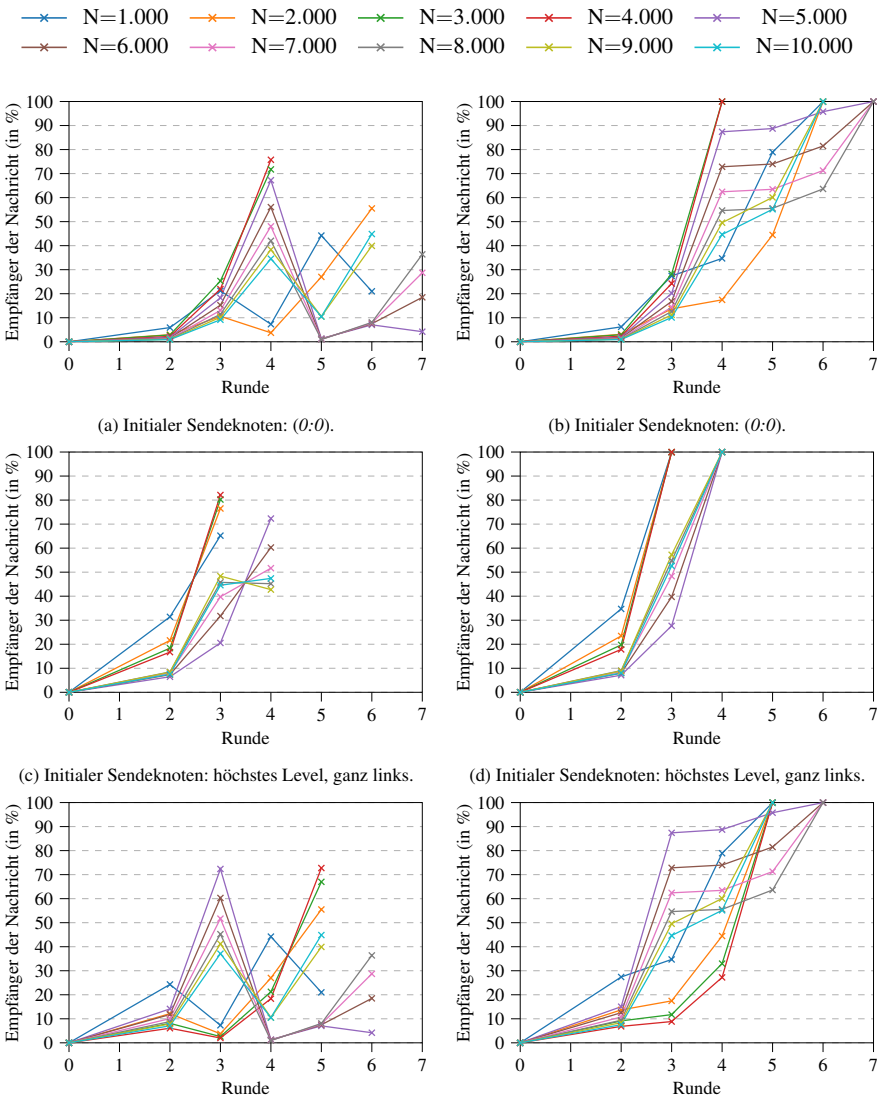
(c) Initialer Sendeknoten: höchstes Level, ganz links.

(d) Initialer Sendeknoten: höchstes Level, ganz links.

(e) Initialer Sendeknoten: zweithöchstes Level, ganz rechts.

(f) Initialer Sendeknoten: zweithöchstes Level, ganz rechts.

Abb. 5.28: Prozentuale Ausbreitung einer Nachricht je Runde (a, c und e) bzw. die kumulierte prozentuale Ausbreitung einer Nachricht bis zur Runde (b, d und f) in Abhängigkeit von der Netzwerkgröße N, dem Fanout $m = 2$ und der Baumhöhe h für unterschiedliche initiale Sendeknoten



(a) Initialer Sendeknoten: (*0:0*).

(b) Initialer Sendeknoten: (*0:0*).

(c) Initialer Sendeknoten: höchstes Level, ganz links.

(d) Initialer Sendeknoten: höchstes Level, ganz links.

(e) Initialer Sendeknoten: zweithöchstes Level, ganz rechts.

(f) Initialer Sendeknoten: zweithöchstes Level, ganz rechts.

Abb. 5.29: Prozentuale Ausbreitung einer Nachricht je Runde (a, c und e) bzw. kumulierte prozentuale Ausbreitung einer Nachricht bis zur Runde (b, d und f) in Abhängigkeit von der Netzwerkgröße N, dem Fanout $m = 16$ und der Baumhöhe h für unterschiedliche initiale Sendeknoten

Kapitel 6
Evaluation

"*Boy, that escalated quickly. I mean, that really got out of hand fast!*"

— Roy Burgundy

In den vorangegangenen Kapiteln wurde ein Konzept zur virtuellen Abbildung des Materialflusses und einer selbstorganisierenden Kommunikation vorgestellt. Um jedoch die Anwendbarkeit der Ansätze darzustellen und den Nutzen aufzuzeigen, erfolgt eine Realisierung der Konzepte. Daher widmet sich dieses Kapitel der empirischen und simulativen Evaluation von SOLA, mit der das Potenzial aufgezeigt wird. Zunächst wird die Evaluationsumgebung beschrieben, mit der das Konzept des selbstorganisierenden Materialflusses und der damit verbundenen Selbstorganisation der Kommunikation evaluiert wird. Im Anschluss erfolgt die Umsetzung von zwei Anwendungsfällen, bei denen sowohl logistische Kenngrößen als auch die Kommunikation betrachtet werden. Der erste Anwendungsfall befasst sich mit einer Simulationsstudie, bei der die Leistung eines Sortiersystems unter Verwendung Fahrerloser Transportfahrzeuge untersucht wird, mit der Randbedingung, dass über die Fahrtwege dezentral verhandelt wird. Der zweite Anwendungsfall umfasst die Simulation des selbstorganisierenden Materialflusses innerhalb eines CPPS. Sowohl der Materialfluss als auch die Fahrerlosen Transportfahrzeuge repräsentieren sich eigenständig und verhalten sich dabei egoistisch bzw. *greedy*. Dies bedeutet, dass jeder Teilnehmer versucht, das für sich beste Ergebnis zu erzielen, um so ein lokales Optimum zu erreichen.

6.1 Die Evaluationsumgebung

Eine Netzwerksimulation ist die Implementierung einer Simulation, welche versucht, das reale Verhalten eines Computernetzwerks und seiner Eigenschaften zu imitieren, sodass die erfassten Informationen und übertragenen Daten analysiert werden können [Ram13]. Der network simulator 3 (ns-3) ist ein diskreter Ereignis-

simulator, das bedeutet, dass jedem Ereignis in der Simulation eine Simulationszeit zugeordnet ist und Ereignisse nacheinander ausgelöst werden. Hierbei bewegt sich die Simulationszeit in diskreten Sprüngen von Ereignis zu Ereignis [RH10].

SOLA hat seinen Ursprung in der Logistik, ist allerdings nicht auf diese beschränkt und kann daher auch in anderen Domänen eingesetzt werden. SOLA ermöglicht die Vernetzung von Entitäten in einem CPPS auf der Anwendungsebene. Durch die Verwendung von SOLA erscheint die Sammlung autonomer und dezentraler Anwendungen dem Benutzer als ein einziges, kohärentes System. Anwendungen können ihre Dienste veröffentlichen bzw. anbieten oder nach anderen Diensten suchen und deren Statusänderungen abonnieren. Zusätzlich besteht die Möglichkeit der Suche nach Informationen und der Filterung. Darüber hinaus wird keine zusätzliche dedizierte Hardware für die Kommunikation wie z. B. ein Broker benötigt. Stattdessen kommunizieren die Teilnehmer selbstständig miteinander. Somit begünstigt SOLA die Selbstorganisation der Applikation in Bezug auf Konnektivität und Verfügbarkeit.

6.1.1 Integration von SOLA in ns-3

SOLA wurde unter der Verwendung von ns-3 entwickelt. Neben einer Vielzahl an möglichen einzustellenden Parametern ermöglicht ns-3 die Verwendung von verschiedenen Kommunikationstechnologien auf der physikalischen Schicht, z. B. 5*G* oder *IEEE802.11* für WLAN. Eine Übersicht der wichtigsten Komponenten und die Integration von SOLA in ns-3 ist in Abb. 6.1 dargestellt.

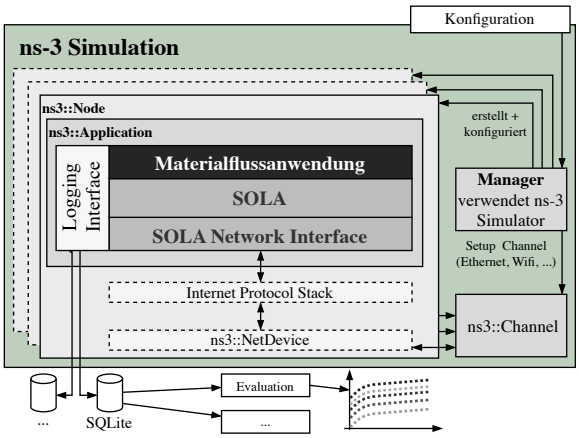


Abb. 6.1: Übersicht der Integration von SOLA innerhalb von ns-3

Das zu simulierende Szenario wird mithilfe einer **Konfiguration** beschrieben. Dabei umfasst das Szenario u. a. die Umgebungsinformationen über die Fabrikhal-

le, die Anzahl an FTF oder die Auftragsperiode. Der Einfachheit halber wird für die Fabrikhalle eine Fläche verwendet, welche durch eine Breite und eine Höhe beschrieben wird. Die Parametrisierung der FTF beinhaltet u. a. die Anzahl, den Startpunkt ihrer Verfügbarkeit und die Fahrzeugeigenschaften, bspw. der verwendeten Ladungsträger, die maximale Nutzlast oder die Kinematik. Die Auftragsfunktion beschreibt, welche Transporte in welcher Häufigkeit erzeugt werden. Sowohl die Erstellung der Transporte als auch die Verteilung, was genau transportiert werden soll, kann mithilfe von stetigen Wahrscheinlichkeitsverteilungen beschrieben werden. Ein anderer wichtiger Parameter der Konfiguration ist die Festlegung der physikalischen Schicht und deren Eigenschaften.

Der **Manager** verarbeitet die Konfigurationsdatei und erstellt das zuvor definierte Szenario. Dazu gehört das Erzeugen von ns-3-Knoten und die Installation der Anwendung auf den Knoten. Jedem Knoten wird eine IPv4-Adresse und ein Port zugewiesen. Darüber hinaus wird jeder Knoten um eine IP/TCP/UDP-Funktionalität mithilfe des *ns3::InternetStack* erweitert. Der Manager konfiguriert auch den *ns3::Channel*, welcher für die Kommunikation benötigt wird und auf dem physikalische Effekte wie Verzögerung, Rauschen oder Ausbreitung simuliert werden, z. B. für kabelgebundene (Ethernet, ...) und drahtlose (WiFi, 5G, ...) Medien. Im Rahmen dieser Arbeit wird sowohl eine drahtgebundene als auch eine drahtlose Kommunikation benötigt. Mobile Geräte wie FTF benötigen eine drahtlose Kommunikation, während ein Transportauftrag über eine drahtgebundene Kommunikation realisiert wird.

Die ***ns-3::Node*** ist der Basis-Container, welcher mit Funktionalität zur Laufzeit erweitert wird. Innerhalb der *ns-3::Node* existiert die *ns-3::Application*, welche auf die anderen Ressourcen zugreift, bspw. auf *ns-3::Channel* oder den *ns-3::InternetStack*. Innerhalb der *ns-3::Application* werden die eigens entwickelten Anwendungen installiert und verwendet. Jede dieser Anwendungen kann ein FTF oder einen Transportauftrag darstellen. Innerhalb der Simulation stellen diese Anwendungen eigenständige Entitäten in einem CPPS dar. Jedes FTF verfügt über mindestens einen Ladungsträger, welcher zum Be-/Entladen verwendet werden kann und Informationen über die physische Abmessung, das maximal unterstützte Nutzlastgewicht oder die Kinematik in Bezug auf Beschleunigung, Verzögerung und Geschwindigkeit. Die Kinematik wird verwendet, um die Bewegung eines FTF in der Fabrikhalle nachzubilden. Hierfür wurde ein eigenes *ns-3::MobilityModel* implementiert, das die Position eines FTF in einem kartesischen Koordinatensystem und dessen Bewegung repräsentiert. Dazu wird eine Trajektorie berechnet, die in drei Phasen aufgeteilt ist. Diese umfasst die Beschleunigung, den Bremsweg und, sofern die Distanz lang genug ist, die konstante Geschwindigkeitsphase. Der TA definiert, welches bestimmte Gut von einem Ort (Abholung) zu einem anderen (Abgabe) bewegt wird.

Mit dem **Logging Interface** kann jede Materialflussanwendung Daten in einer Datenbank protokollieren. Diese Daten umfassen dabei Aktionen und Zustände der Entitäten oder auch Ereignisse. Letztere können entweder durch die Umgebung initiiert werden, wie bei der Erstellung eines neuen Transportauftrags, oder aus eigenem Interesse einer Entität. Die Schnittstelle für das automatische Protokollieren

von Daten unterstützt verschiedene Arten von Datenbanken und Formaten, welche für eine reale Anwendung von Bedeutung sein können. Innerhalb von ns-3 ist dies eine zentrale, portable SQLite[1]-Datenbank. Die protokollierten Daten werden während der Entwicklung zur Überprüfung und Bewertung von Algorithmen verwendet oder für eine Bewertung des Systems.

Das **SOLA Netzwerk Interface** dient als Abstraktion der realen Netzwerkschnittstelle, welche der Materialflussanwendung suggeriert, dass diese in einer realen Umgebung ausgeführt wird.

Netzwerkinfrastruktur

In einem CPPS existieren mobile und stationäre Teilnehmer (vgl. Abschn. 3.1), die miteinander kommunizieren können. Die Infrastruktur des Netzwerks, bestehend aus Netzwerk Switches und WLAN Access Points, ist in Abb. 6.2 dargestellt. Mobile Teilnehmer wie Fahrerlose Transportfahrzeuge bewegen sich frei im Raum und sind über den WLAN Access Point verbunden. Stationäre Teilnehmer wie Transportaufträge oder Abhol- bzw. Lieferstationen befinden sich im kabelgebundenen Netzwerk.

Um eine realistische Abbildung einer Netzwerkinfrastruktur zu haben, existieren diverse Switches mit einer begrenzten Anzahl an Ports, welche miteinander über einen Router vernetzt sind. Je nach Größe des Netzwerks und der Anzahl an Teilnehmern kann es hierbei allerdings zu Herausforderungen kommen. Eine Herausforderung stellt dabei das Address Resolution Protocol (ARP) Flooding zur Auflösung einer IP-Adresse zu einer physikalischen Adresse dar [KAK14]. Daher wurde für die Entitäten, die an einem Switch hängen, das ARP deaktiviert und eine statische Zuweisung vorgenommen [Mad19]. Dies reduziert innerhalb eines Switchs den initialen Kommunikationsaufwand für die Auflösung, bevor eine Entität mit einer anderen Entität kommuniziert, die am selben Switch angeschlossen ist. Eine

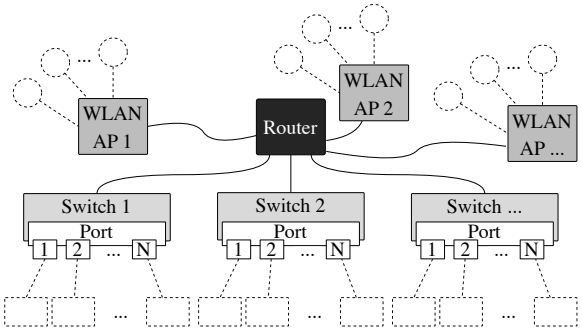


Abb. 6.2: Aufbau der Netzwerkinfrastruktur

[1] SQLite: https://www.sqlite.org/

Auflösung einer IP–Adresse auf eine MAC-Adresse erfolgt lediglich, wenn letztere nicht bekannt ist und ein Paket an eine Entität geschickt wird, welche an einem anderen Netzwerk-Switch angeschlossen ist. In Tabelle 6.1 sind die Parameter für die drahtlose und die drahtgebundene Kommunikation aufgelistet, welche innerhalb der Simulationen verwendet wurden. Weicht ein Wert für eine Simulation ab, so wird dies nachfolgend explizit erwähnt.

Tabelle 6.1: Parameter der Vernetzung für die Evaluation in ns-3

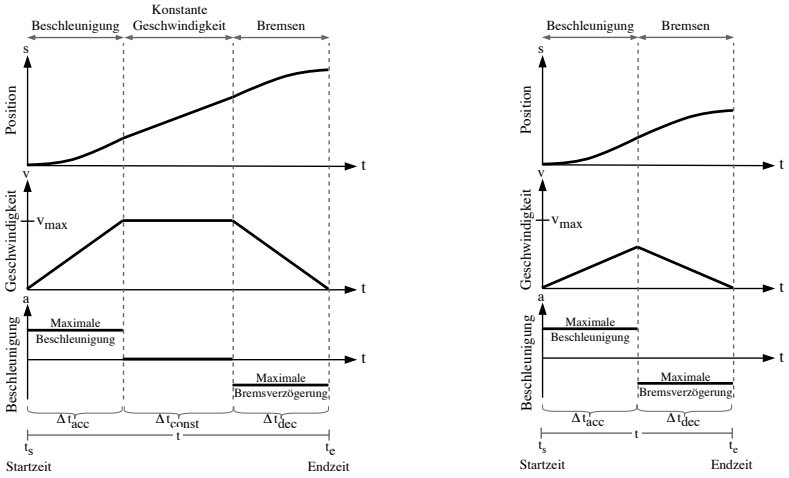
Drahtlose Vernetzung (CSMA/CA)			
MAC	IEEE 802.11n	**Frequenzband**	2,400 GHz - 2,4835 GHz
Kanäle	13	**Kanalbreite**	20 MHz
Sendeleistung	20 dBm [Bun19]	**Empfangssensibilität**	-85 dBm
RtsCtsThreshold	100 Byte	**ActiveProbing**	Deaktiviert
Drahtgebundene Vernetzung (CSMA/CD)			
Ports pro Switch	48	**CSMA Queue Size**	1000
CSMA Link	10 Gbps		

Das Fahrprofil eines Fahrerlosen Transportfahrzeugs

Um die Bewegung eines FTF in ns-3 nachzubilden, existieren zwei verschiedene Fahrprofile, die Standardfahrrampe und die spitze Rampe [HSD18, S. 295 ff.]. Beide Fahrprofile sind in der nachfolgenden Abb. 6.3 dargestellt, bestehend aus der Wegstrecke s, der Geschwindigkeit v und der Beschleunigung a. In der Standardfahrrampe, auch Trapez genannt, beschleunigt ein FTF mit a_{acc} auf seine maximale Geschwindigkeit v_{max} (vgl. Abb. 6.3a). Diese Geschwindigkeit wird für den Zeitraum Δt_{const} gehalten, bevor die Bremsverzögerung mit a_{dec} eingeleitet wird. Hier ist zu beachten, dass die Beschleunigung a_{acc} nicht der Bremsverzögerung a_{dec} entsprechen muss, sodass sich unterschiedliche Zeiten für die Beschleunigung Δt_{acc} und den Bremsweg Δt_{dec} ergeben können. Im zweiten Fahrprofil, der spitzen Rampe, erreicht ein FTF nicht die maximale Geschwindigkeit v_{max} und das Fahrzeug muss den Bremsvorgang einleiten (vgl. Abb. 6.3b). Grundlage für die Berechnung der Fahrprofile ist das Weg-Zeit-Gesetz:

$$s(t) = \frac{a}{2}t^2 + v_0 t + s_0$$

Unter der Annahme $a_{acc} \neq a_{dec}$ lassen sich hieraus die entsprechenden Fahrzeiten für die Phasen der Standardfahrrampe und der spitzen Rampe in Gl. 6.1 bestimmen:



(a) Standardfahrrampe

(b) Spitze Fahrrampe

Abb. 6.3: Fahrprofile für ein Fahrerloses Transportfahrzeug

$$t = \begin{cases} \Delta t_{acc} + \Delta t_{const} + \Delta t_{dec} & s \geq \frac{1}{2}\left(\frac{v_{max}^2}{a_{acc}} + \frac{v_{max}^2}{a_{dec}}\right) \text{ Standardfahrrampe} \\ \Delta t_{acc} + \Delta t_{dec} & s < \frac{1}{2}\left(\frac{v_{max}^2}{a_{acc}} + \frac{v_{max}^2}{a_{dec}}\right) \text{ spitze Fahrrampe} \end{cases} \tag{6.1}$$

Unter der Annahme, dass die Beschleunigung der Bremsverzögerung entspricht, ergeben sich folgende Zeiten für die unterschiedlichen Fahrprofile:

$$t = \begin{cases} \Delta t_{acc} + \Delta t_{const} + \Delta t_{dec} & s \geq \frac{v_{max}^2}{a_{acc}} \text{ Standardfahrrampe} \\ \Delta t_{acc} + \Delta t_{dec} & s < \frac{v_{max}^2}{a_{acc}} \text{ spitze Fahrrampe} \end{cases} \tag{6.2}$$

Andere Fahrprofile, bei denen ein FTF bis zu einem gewissen Punkt beschleunigt und dann abbremst oder angehalten wird, sind nicht Gegenstand dieser Arbeit.

Randbedingungen, Einschränkungen und Parametrisierung der Simulation

Bei einer Simulation muss berücksichtigt werden, dass diese lediglich versucht, das reale Verhalten zu imitieren bzw. abzuschätzen. Daher ist eine Simulation lediglich ein Werkzeug, welches die Entwicklung vereinfacht, jedoch auch Einschränkungen mit sich bringt und bei dem Randbedingungen zu beachten sind. Im Rahmen dieser Arbeit existieren folgende Randbedingungen und Einschränkungen:

- Verarbeitungszeiten von Ereignissen innerhalb einer Entität sind nicht berücksichtigt. Empfängt eine Entität ein Datenpaket, so entspricht die Verarbeitungszeit in der Simulation 0 s.

- Überlagerungen von Radiofrequenzen (Interferenzen) anderer kabelloser Geräte sind nicht berücksichtigt.
- Es erfolgt keine Übergabe einer Verbindung zwischen zwei oder mehreren WLAN Access Points (WLAN Handover). Folglich existieren in der Simulation mehre WLAN Access Points, welche die gesamte Fabrikhalle funktechnisch ausleuchten, sodass sich der WLAN Access Point in der Mitte der Versuchshalle befindet.
- Um eine entsprechende Funkausleuchtung zu erhalten, ist die Sendeleistung der WLAN Access Points auf das in Deutschland zugelassene Maximum konfiguriert [Bun19].
- Pro WLAN-Kanal existiert nur eine begrenzte Anzahl an Teilnehmern, sodass der Kanal nicht überlastet wird.
- Alle Teilnehmer sind zeitsynchron.
- Kein Teilnehmer fällt aus.
- Alle Nachrichten, die versendet werden, kommen beim Empfänger an. Die Daten innerhalb der Nachricht sind zu jedem Zeitpunkt korrekt.
- Jeder Teilnehmer ist eindeutig identifizierbar mithilfe einer UUID.

6.2 Dezentrale Verhandlung der Pfadplanung

In einem System mit mobilen Robotern wie z. B. Fahrerlosen Transportfahrzeugen ist die kollisionsfreie Bewegung, auch Pfadplanung genannt, eine wichtige Eigenschaft. In einer Umgebung mit mehreren unabhängigen mobilen Fahrerlosen Transportfahrzeugen wird bei der Pfadplanung zwischen einer globalen und einer lokalen Pfadplanung unterschieden [DRVD20, RP12, BNM11]. Die globale Pfadplanung bestimmt eine Route von einem Start- zu einem Zielpunkt im Raum, unter Berücksichtigung von statischen Hindernissen oder der Berücksichtigung von bereits temporär belegten Routen. Zur Reduktion der Wartezeit eines FTF kann durch die globale Pfadplanung eine alternative Route geplant werden. Dabei kann der Raum bspw. durch einen Graphen repräsentiert werden, bestehend aus Knoten (Orten im Raum) und Kanten (Wege zwischen den Orten) [SLS21], oder durch den idealen logistischen Raum, eine offene Teilmenge des euklidischen Raums [Roi22]. Wird auf einem Graphen geroutet, so kann die kürzeste Route u. a. durch den Dijkstra-Algorithmus bestimmt werden [Dij59], unter Berücksichtigung von freien Zeitfenstern [MZW07]. Trifft ein FTF auf einer Route auf ein unbekanntes Hindernis, so muss dieses umfahren werden. Dies geschieht durch eine lokale Pfadplanung. Je mehr Stationen und Fahrerlose Transportfahrzeuge existieren, desto größer wird der Suchraum für eine kollisionsfreie globale Pfadplanung. Insbesondere zentralisierte Ansätze stoßen hier schnell an ihre Grenzen [Reg08, Abschn. 2.4], [DRVD20]. Da der Fokus dieser Arbeit die Dezentralität ist, wird nachfolgend eine dezentrale Pfadplanung mithilfe des bereits in dieser Arbeit vorgestellten dezentral organisierten Kommunikationsframeworks SOLA realisiert (vgl. Abschn. 5). Dazu wird als Szenario ein Paketsortiersystem mit Loop-Struktur verwendet [HSD18, S. 169], für das eine klassische Leistungsberechnung existiert [Roi22, Abschn.6.3]. Teile dieses Abschnitts basieren auf den Vorarbeiten des Autors in [LDE+22, Sto22, Tön22, Göd23].

In Abb. 6.4 ist ein Versuchsaufbau dargestellt, bestehend aus Abhol- und Lieferstationen. In diesem Szenario existieren Unstetigförderer in Form von Fahrerlosen Transportfahrzeugen, welche sich innerhalb des idealen logistischen Raums

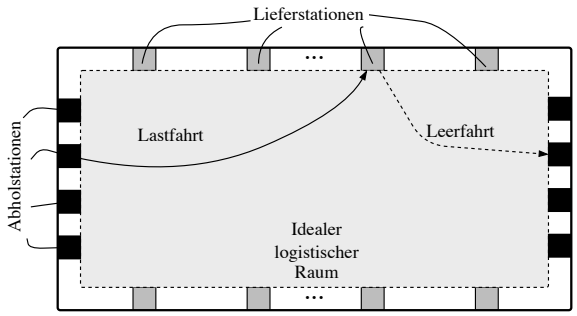


Abb. 6.4: Sortiersystem mit Fahrerlosen Transportfahrzeugen

bewegen können. Waren werden von einer Quelle (Abholstation) zu einer Senke (Lieferstation) transportiert. Befindet sich ein Paket auf einem FTF, während dieses von einer Abholstation zu einer Lieferstation fährt, so handelt es sich um eine Lastfahrt. Bewegt sich ein FTF ohne Ware von einer Station zu einer anderen, so handelt es sich um eine Leerfahrt.

Eine Dezentralisierung des vorliegenden Anwendungsszenarios erfolgt durch eine konsequente Modularisierung (vgl. Abschn. 2.3) wie in der Abb. 6.5 dargestellt. Das ERP-Modul erzeugt die Aufträge unter Verwendung der MFDL (vgl. Abschn. 4.2). Dieser Auftrag wird einer Abholstation zugewiesen. Jede Abholstation ist eine eigenständige Entität, welche sich selbst verwaltet und Trajektorien für Fahrerlose Transportfahrzeuge basierend auf deren Fahrprofil plant (vgl. Abschn. 6.1.1). Eine Trajektorie T_i eines Fahrerlosen Transportfahrzeugs f_i kann durch die Menge von Punkten $p \in \mathscr{R}^2$ und je einem dazugehörigen Zeitpunkt t_p beschrieben werden. Eine Kollision liegt vor, wenn die Trajektorien T_{f_1} und T_{f_2} zum selben Zeitpunkt t_p in zwei Punkten $p_{f_1} = p_{f_2}$ übereinstimmen. Im Umkehrschluss bedeutet dies, dass zwei Trajektorien kollisionsfrei sind, wenn zu jedem Zeitpunkt t_p $p_{f_1} \neq p_{f_2}$ gilt. Eine Verringerung der Geschwindigkeiten oder gar ein Stillstand eines FTF, sobald dieses die Höchstgeschwindigkeit erreicht hat, ist nicht vorgesehen. Eine Abholstation versucht nach Erhalt des Transportauftrags, die Verzögerung des Startzeitpunkts der Lastfahrt minimal zu halten mit $t_{start} \rightarrow 0$ und plant dabei auf dem vorhandenen Wissen.

Ein FTF mit einem omnidirektionalen Antrieb kann eine Rotation entgegen seiner Fahrtrichtung durchführen. Dies ist besonders vorteilhaft, wenn die Trajektorie einer Spline entspricht (vgl. Abb. 6.6a). Zur Abstraktion und zur Vereinfachung werden allerdings nachfolgend die Fahrtwege als Geraden angenommen (vgl. Abb. 6.6b). Die hieraus entstehenden Schnittpunkte werden für einen

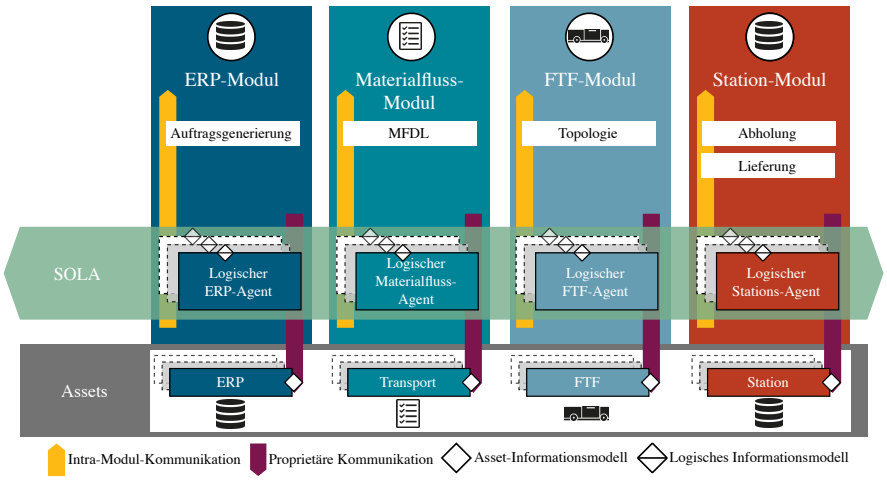
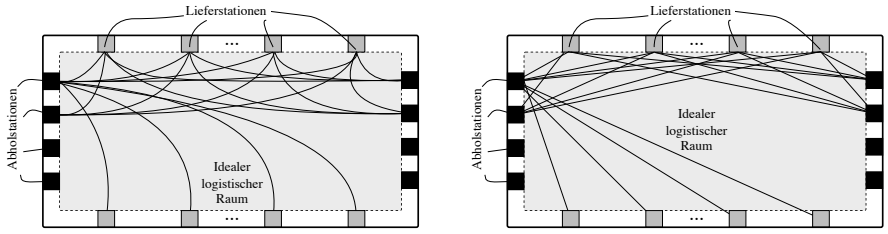


Abb. 6.5: Architektur zur Dezentralisierung der Pfadplanung für Fahrerlose Transportfahrzeuge



(a) Fahrtwege dargestellt als Splines (b) Fahrtwege dargestellt als Geraden

Abb. 6.6: Beispiel der Abstraktion der Fahrtwege und Kollisions- bzw. Schnittpunkte

Zeitpunkt t_n für $\pm\Delta$ als belegt markiert. Die Schnittpunkte der Geraden können vor Beginn des Szenarios berechnet werden und somit als global verfügbares Wissen angenommen werden, sodass jede (Abhol- bzw. Liefer-)Station diese kennt.

Bevor ein FTF einen Transportauftrag ausführen kann, muss eine Belegung des entsprechenden Weges mit den anderen Stationen verhandelt werden. Zunächst plant eine Abholstation a_i die Trajektorie für ein Fahrzeug, sodass eine nur ihr bekannte lokale Sicht entsteht. Im Anschluss wird diese lokale Sicht, mit der geplanten Trajektorie einer Abholstation, unter allen Abholstationen verhandelt, damit die lokal gespeicherten Informationen der Abholstation a_i den anderen Abholstationen bekannt ist. Dies geschieht durch die Verwendung von Paxos [Lam98, Lam01], einem *voting-based* Konsensus-Algorithmus zur Einigung auf eine gemeinsame Informationsbasis [BMT+20] bzw. es wird sich primär durch Abstimmungen geeinigt, wessen Daten als nächstes übernommen werden. Das vereinfachte Verfahren ist in Abb. 6.7 dargestellt. In der *Propose*-Phase kündigt eine Abholstation gegenüber den anderen Abholstationen an, dass sie einen neuen Wert bzw. eine Trajektorie vorschlagen will.

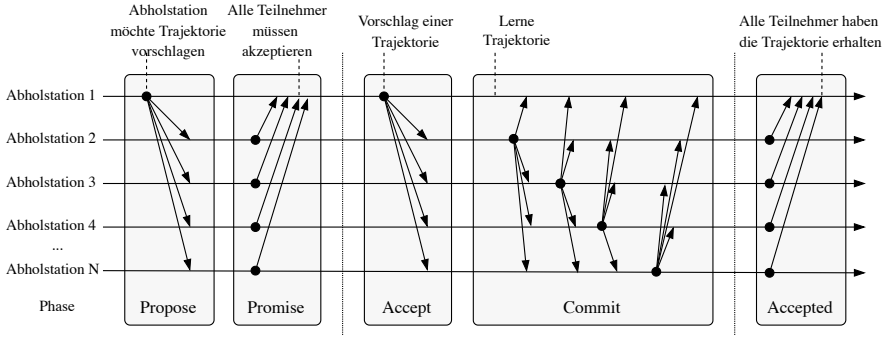


Abb. 6.7: Vereinfachte Darstellung des Paxos-Algorithmus [Lam98], angewendet auf die Verhandlung einer einzelnen Trajektorie zwischen den Abholstationen

Sobald die Abholstation in der Phase *Promise* eine Bestätigung von den anderen Abholstationen erhalten hat, sodass eine Mehrheit (Quorum) entstanden ist, kann diese Abholstation in der Phase *Accept* eine neue Trajektorie mit den entsprechenden Belegungszeiten der Schnittpunkte vorschlagen. Nun verhandeln alle Abholstationen in der Phase *Commit* miteinander, in der sich auf diesen Wert geeinigt wird. Hieraus resultiert, dass bei N Abholstationen insgesamt $N \cdot (N-1)$ Unterverhandlungen erfolgen. Hier teilen auch alle Abholstationen den anderen Abholstationen mit, dass sie den neuen Zustand akzeptieren, sodass ein globales Wissen über anstehende geplante Trajektorieren entsteht. In der Phase *Accepted* senden alle verbleibenden Abholstationen der zuvor gewählten Abholstation ihren Wert zu, sodass die Abholstation dem Fahrerlosen Transportfahrzeug den Fahrbefehl schicken kann.

Die *Propose*-Phase kann von jeder Abholstation zu jedem Zeitpunkt initiiert werden, auch wenn die *Propose*-Phase einer anderen Abholstation aktiv ist. Zur Vermeidung eines gegenseitigen Blockierens können Abholstationen erst nach einer bestimmten Zeit wieder senden. Diese Zeit kann für jede Abholstation unterschiedlich gewählt werden, z. B. durch exponentielle Verzögerung. In diesem Verfahren handelt es sich um eine vollständige dezentrale Aushandlung der Trajektorien bzw. der Bestimmung der Abholstation, deren Trajektorie als nächster Wert bei den verbleibenden Abholstationen gültig wird, sodass ein *Multi-Paxos* durchgeführt wird [VRA15]. Eine Vereinfachung des Systems ist, dass eine vollständig dezentrale Wahl eines zentralen Planers für Trajektorien erfolgt, bspw. durch die Verwendung von Raft [OO14], was u. a. die Ausfallerkennung einzelner Teilnehmer ermöglicht.

In Abb. 6.8 ist die vereinfachte Interaktion zwischen den einzelnen Teilnehmern inkl. der verwendeten Topics dargestellt. Diese berücksichtigt sowohl die dezentrale Verhandlung, welche zwischen den Abholstationen erfolgt, als auch die Auftragszuweisung zwischen einer Abholstation und einem FTF. Transportaufträge werden von einem ERP-System erzeugt und einer Abholstation zugewiesen. Sobald eine Station einen Auftrag erhält, erfolgt die Durchführung der dezentralen Verhandlung bzgl. der Belegung der Trajektorien. Ist ein Konsens gefunden, so wird der Fahr-

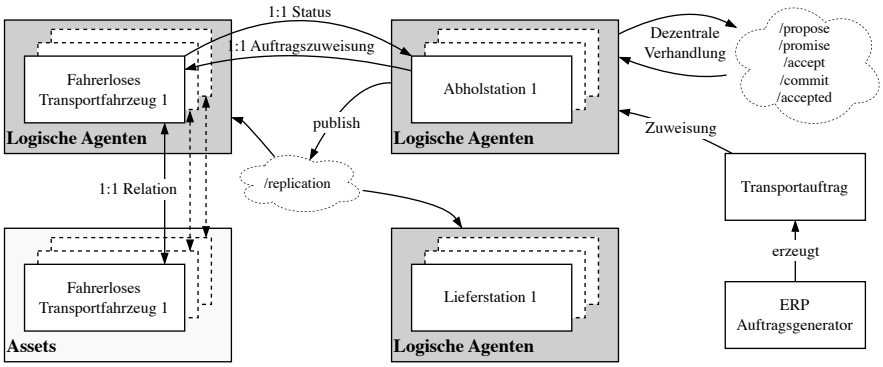


Abb. 6.8: Topics und Interaktionen zwischen den logischen Agenten

auftrag dem logischen Agenten des Fahrerlosen Transportfahrzeugs zugewiesen, welcher das physikalische Asset bzw. das Fahrerlose Transportfahrzeug verwaltet. Darüber hinaus wird der gemeinsame Konsens an alle Teilnehmer, bestehend aus der Menge von Fahrerlosen Transportfahrzeugen sowie Abhol- und Lieferstationen, verteilt. Somit liegt allen Teilnehmern eine Kopie der Transportaufträge vor, was einem vereinfachten Distributed Ledger (DL) entspricht. Dieser DL kann u. a. zur Abrechnung der Transporte verwendet werden. Die logischen Agenten der Fahrerlosen Transportfahrzeuge werden auf dem Fahrzeug selbst ausgeführt. Dies bedeutet, dass die Kommunikation mit den logischen Agenten durch SOLA über WLAN erfolgt. Nach Abschluss einer Lastfahrt wird im Anschluss über eine Leerfahrt verhandelt, welche der Fahrt von der aktuellen Lieferstation zur nächsten Abholstation entspricht. Nachdem ein FTF bei der Abholstation angekommen ist, beginnt das Verfahren über die Verhandlung eine Trajektorie für die nächste Lastfahrt von vorn. Im Management Overlay sind alle Teilnehmer vertreten. In den Topics zur dezentralen Verhandlung sind lediglich die Abholstationen involviert. Hierfür existiert für jedes Topic ein eigener MINHTON-Baum. In dem Topic *Replication*, für den ebenfalls ein eigener MINHTON-Baum existiert, sind alle Teilnehmer des Szenarios vertreten.

6.2.1 Simulation des Anwendungsszenarios

Zu Beginn der Simulation werden alle FTF gleichmäßig auf die Abholstationen verteilt. Bei der Planung einer neuen Trajektorie wird eine zufällige Lieferstation mithilfe einer Gleichverteilung als Zielpunkt für die Lastfahrt festgelegt. Zusätzlich wird die nächste Abholstation bestimmt. Dies geschieht dadurch, dass die Seite mit Abholstationen ausgewählt wird, welche nach euklidischer Distanz am nächsten ist. Im Anschluss wird eine der vier Abholstation zufällig über eine Gleichverteilung ausgewählt. Damit verteilen sich die FTF im Verlauf der Simulation gleichmäßig auf die Abholstationen. Mit einem Transportauftrag wird dem FTF die nächste Abholstation mitgeteilt, sodass sich das FTF, wenn es bei der Lieferstation angekommen ist, bei seiner bisherigen Abholstation abmelden und bei der neuen anmelden kann. Die neue Abholstation ist auch für die Planung der Trajektorie der Leerfahrt von einer Lieferstation zur Abholstation zuständig. Eine Abholstation kann beliebig viele FTF verwalten, daher werden mögliche Kollisionen innerhalb der Abholstationen nicht betrachtet. Die Planung der Trajektorie erfolgt erst, wenn ein FTF an einer Lieferstation bzw. an einer (neuen) Abholstation angekommen ist. Da zu jedem Zeitpunkt bei allen Abholstationen dieselben Informationen vorliegen, versucht eine Abholstation, den nächstmöglichen freien Zeitpunkt für eine Trajektorie zum Zielort zu planen. Treffen in der Zwischenzeit neue Informationen ein, so werden diese berücksichtigt, was zu einer Verzögerung des Transports führen kann bzw. führt.

In Tabelle 6.2 sind die Parameter der Simulation dargestellt. Während die Abhol- und Lieferstationen nicht verändert werden, wurde die Anzahl der Fahrerlosen

Transportfahrzeuge variiert. Anzumerken gilt, dass die gesamte Simulationszeit ungefähr 900 s dauert, allerdings 150 s zu Beginn und am Ende zur Stabilisierung des Systems entfernt werden, sodass die betrachtete Simulationszeit ungefähr 600 s entspricht. Die nachfolgenden Ergebnisse, sofern diese sich auf eine Stunde beziehen, wurden ausgehend von den 600 s auf eine Stunde hochgerechnet. Jedes Experiment wurde 10-mal – jeweils mit verschiedenen Startwerten für einen Zufallsgenerator – durchgeführt und für die nachfolgenden Abbildungen gemittelt.

Tabelle 6.2: Übersicht der Parametrisierung zur simulativen Evaluation der dezentralen Pfadplanung

Nr.	Parameter	Wert
1	Fanout $m \in \mathbb{N}$	$m = 2$
2	Anzahl an FTF $N \in \mathbb{N}$	$N \in \{10,\ 20,\ 30, \ldots,\ 100\}$
3	Anzahl an Abholstationen	8
4	Anzahl an Lieferstationen	50
5	Simulationszeit	900 s
6	Stabilisierungszeit	je 150 s am Anfang und am Ende
8	Anzahl an Wiederholungen	10
7	Belegungszeit Schnittpunkte	$\Delta = 0{,}5$ s

In Abb. 6.9 ist ein Beispiel des Szenarios mit 40 Fahrerlosen Transportfahrzeugen zur Ausführungszeit dargestellt. Die Gesamtfläche beträgt 113 m × 25 m. Insgesamt existieren 50 Lieferstationen, verteilt auf beide Längsseiten mit jeweils 25 Stationen. Die Breite der Lieferstationen ist 1,2 m, der Abstand zwischen den Lieferstationen jeweils 2,8 m, der initiale Abstand am Anfang bzw. finale Abstand am Ende ist 6,5 m. Zusätzlich existieren 8 Abholstationen, verteilt auf beide Stirnseiten mit jeweils 4 Stationen. Der Abstand zwischen den Stationen entspricht 4 m.

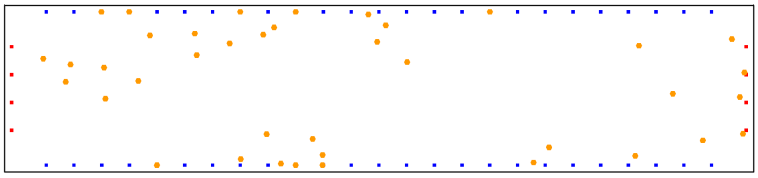


Abb. 6.9: Screenshot des Basisszenarios mit 40 Fahrerlosen Transportfahrzeugen während der Simulation

Betrachtung des System – Logistische Kennzahlen

Aus logistischer Sicht wurde die theoretische obere Schranke der Leistung für dieses Szenario simulativ bestimmt (vgl. Abschn. 6.3.1 [Roi22]). Bei der Bestimmung der oberen Leistungsgrenze wurde ein zentraler Ansatz verwendet, welcher ebenfalls dezentralisierbar ist. Zusätzlich wurden die Trajektorien auf der Basis homotoper Bahnkurven geplant, sodass zwischen einem Start- und einem Zielpunkt mehrere Pfade möglich sind.

In dem hier vorliegenden simulierten Szenario erfolgt die Umkehrung des zentralisierten Ansatzes, sodass jede Abholstation eigenständig mit den verbleibenden Abholstationen über eine Belegung der Schnittpunkte mithilfe einer Paxos-Konsensus-Findung verhandelt. Dies entspricht aus kommunikationstechnischer Sicht, sofern für den Kollisionsabstand und die Ausdehnung des Fahrzeugs ein Wert von 1 angenommen wird, bei der dezentralen Planung auch der theoretischen oberen Grenze. Je nachdem, welcher Abholstation ein Auftrag zugewiesen wird, und je nach Optimalität der Verhandlung der Abholstationen untereinander kann ein anderer Durchsatz erreicht werden.

In Abb. 6.10 ist zum einen die theoretische Leistung aus [Roi22] und zum anderen die Leistung bzw. die obere Grenze der hier vorgestellten dezentralen Planung dargestellt. Als Referenz wird die obere Schranke der Leistung ohne Kollisionsvermeidung genommen, was bedeutet, das die FTF im Konfliktfall durch einander hindurchfahren. Beide Verfahren verwenden eine maximale Beschleunigung von $4\,\mathrm{m/s^2}$ und einen Kollisionsabstand von 0 cm. Ein weiterer Unterschied, neben dem, *wie* und *wo* die Entscheidung getroffen wird, ist, dass das hier simulierte Verfahren eine Fahrzeugausdehnung von 1 betrachtet. Es ist ersichtlich, dass mit einer Einführung eines Sicherheitsabstandes zur Kollisionsvermeidung bzw. einer Ausdehnung des Fahrzeugs die Leistung des Systems reduziert wird, gegeben durch die Tatsache, dass die Schnittpunkte für einen längeren Zeitpunkt belegt sind. Mit steigender Anzahl an Fahrerlosen Transportfahrzeugen nimmt die Gesamtanzahl

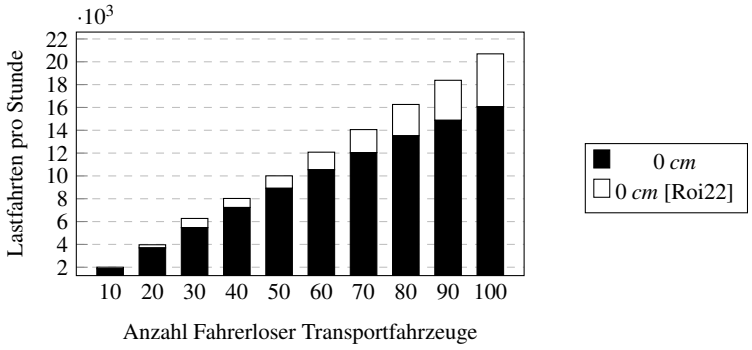
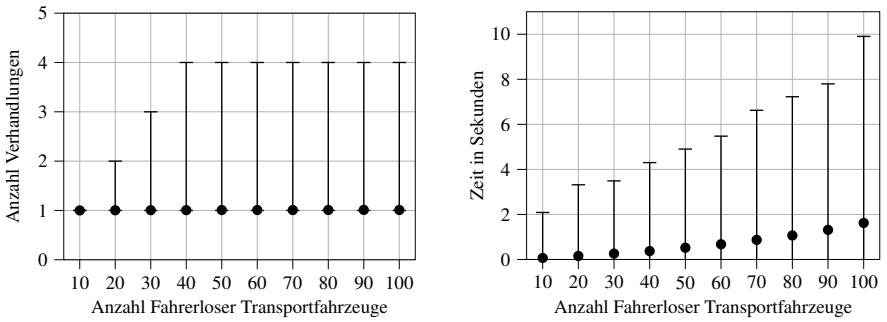


Abb. 6.10: Leistungswerte eines Sortiersystems, realisiert mit Fahrerlosen Transportfahrzeugen, unter Berücksichtigung eines Kollisionabstands bzw. Sicherheitsabstands von 0 cm bei $4\,\mathrm{m/s^2}$. Die Werte in weiß sind entnommen aus [Roi22], die Werte in schwarz sind in dieser Arbeit entstanden.

an transportierten Paketen pro Stunde zu, die Anzahl an durchgeführten Lastfahrten je FTF sinkt hingegen. Dies wird auch in Abb. 6.11 und Abb. 6.12 ersichtlich. So ist in Abb. 6.11a die minimale, die maximale und die durchschnittliche Anzahl an benötigten Verhandlungen für eine Trajektorie abgebildet. Während der Durchschnitt nahezu konstant bleibt, wächst die maximale Anzahl an Verhandlungen entsprechend. Diese Trajektorien sind allerdings nur Ausreißer. In Abb. 6.11b hingegen ist die Zeit dargestellt, welche benötigt wird, bis ein FTF nach erfolgreicher Beladung die Abholstation verlassen kann. Die durchschnittlich benötigte Wartezeit steigt bei einer zunehmenden Anzahl an FTF an. Dies hängt zum einen mit der Anzahl der vorhandenen Schnittpunkte zusammen, zum anderen damit, dass für einen Weg von einer Abholstation zur Lieferstation und umgekehrt nur eine mögliche Trajektorie existiert und somit die Wahrscheinlichkeit dafür steigt, dass die gewählte Route durch ein anderes FTF blockiert ist.



(a) Benötigte Verhandlungen (b) Benötigte Zeit

Abb. 6.11: Minimale, maximale und durchschnittliche benötigte Anzahl an Verhandlungen für eine erfolgreiche Planung der Trajektorie (a) bzw. die benötigte Zeit (b), bis ein Fahrerloses Transportfahrzeug losfahren kann

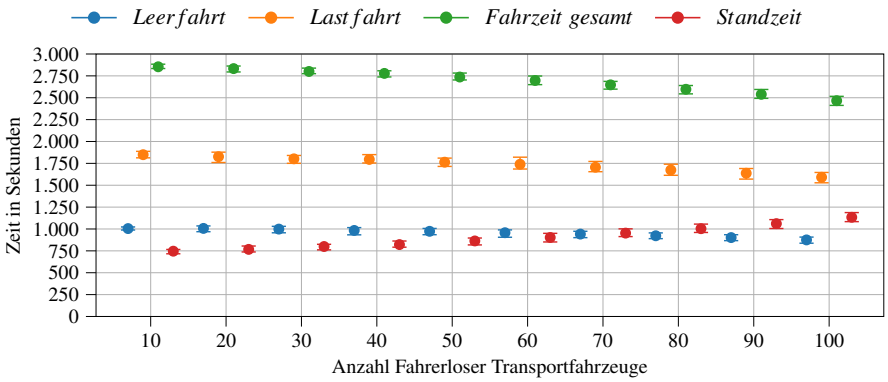


Abb. 6.12: Minimale, maximale und durchschnittliche Fahr- und Standzeit pro Stunde für ein Fahrerloses Transportfahrzeug

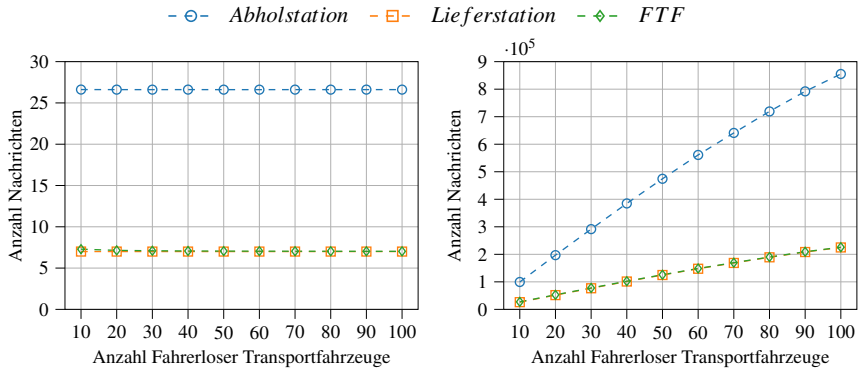
In Abb. 6.12 ist die minimale, die maximale und die durchschnittliche Nutzungsdauer eines FTF pro Stunde dargestellt. Die Fahrzeit selbst lässt sich nochmals in Zeiten für eine Leerfahrt und eine Lastfahrt unterteilen. Die Lastfahrt beinhaltet auch die Zeit für die Be- bzw. Entladung eines Fahrzeugs. Auch hier wird ersichtlich, dass die durchschnittliche Fahrzeit mit steigender Anzahl an FTF im System sinkt, die Standzeit für jedes FTF hingegen steigt. Dies resultiert u. a. aus der leicht steigenden Zeit, bis ein FTF losfahren kann (vgl. Abb. 6.11b). Die Auslastung der Fahrzeuge ist jedoch relativ ausgeglichen, gegeben durch die Tatsache, dass die Minima bzw. Maxima nah beieinander liegen.

Betrachtung der Kommunikation

In jedem einzelnen simulierten Szenario ändert sich die Anzahl an Teilnehmern nicht. Die ersten N Knoten in MINHTON für ein Topic entsprechen den Fahrerlosen Transportfahrzeugen, gefolgt von den Abhol- und Lieferstationen. Somit sind die mobilen Teilnehmer auf den niedrigsten Leveln und haben weniger Verbindungen zu anderen Knoten als Knoten auf höheren Leveln (vgl. Abschn. 5.2.1). Anschließend treten die Teilnehmer den entsprechenden Topics bei (vgl. Abb. 6.8). Im Folgenden wird die Kommunikation untersucht, welche durch die Interaktion der Abholstationen während der dezentralen Pfadplanung entsteht. Der initiale Kommunikationsaufwand bei der Erstellung der Topic-Bäume oder des Management Overlays wird hier vernachlässigt.

In Abb. 6.13 ist die durchschnittliche empfangene Anzahl an Nachrichten für die drei unterschiedlichen Typen an Teilnehmern dargestellt. Abb. 6.13a berücksichtigt die durchschnittliche Anzahl an empfangenen Nachrichten für eine Verhandlung. Die Anzahl der empfangenen Nachrichten für eine Abholstation ist konstant und unabhängig von der Anzahl an Fahrerlosen Transportfahrzeugen bzw. Lieferstationen. Dies resultiert daraus, dass die Verhandlung lediglich unter den Abholstationen und somit ohne Kommunikation mit den Transportfahrzeugen und Lieferstationen erfolgt. Die Abholstationen empfangen auf allen Topics Nachrichten, die Lieferstationen und FTF empfangen lediglich auf dem Topic *Replication*. Die empfangenen Fahrbefehle zu einer Abholstation bzw. zu einer Lieferstation sind hier nahezu vernachlässigbar mit steigender Anzahl an Fahrerlosen Transportfahrzeugen.

Das Versenden einer Nachricht durch eine Abholstation auf dem Topic *Prepare* bzw. *Accept* bedeutet, dass diese von $(N-1)$ Abholstationen empfangen wird. Folglich empfangen durchschnittlich $\frac{7}{8}$ bzw. $\frac{N-1}{N}$ Abholstationen eine Nachricht pro initial versendeter Nachricht. Auf den Topics *Promise*, *Commit* und *Accepted* beantworten die $(N-1)$ verbleibenden Abholstationen die jeweilige vorherige Phase. Gegeben durch die Interaktion des Verfahrens werden auf diesen Topics jeweils $(N-1)$ Nachrichten versendet, welche von $(N-1)$ Abholstationen empfangen werden, was in $\frac{(N-1)\cdot(N-1)}{N}$ bzw. hier in $\frac{49}{8}$ empfangenen Nachrichten resultiert. Auf dem Topic *Replication* werden ebenfalls jeweils $(N-1)$ Nachrichten nach Erhalt der ersten Nachricht auf diesem Topic versendet, sodass die Anzahl an empfangenen Nachrichten hier ebenfalls $\frac{(N-1)\cdot(N-1)}{N}$ bzw. $\frac{49}{8}$ entspricht. Konkret bedeutet dies,



(a) Je Verhandlung (b) Je Stunde

Abb. 6.13: Durchschnittliche Anzahl an Nachrichten, die ein Teilnehmer des angegebenen Typs empfängt

dass insgesamt 210 Nachrichten auf den Topics empfangen wurden. Hinzu kommen noch für die Abholstation, welche das Fahrzeug beauftragt hat, Nachrichten auf dem Applikation hin zu, bei der das FTF seinen Status übermittelt. Im Verhältnis zum Paxos-Verfahren ist dieser fast vernachlässigbar. Die FTF und Lieferstationen erhalten hauptsächlich auf dem Topic *Replication* Nachrichten, bei der alle Teilnehmer im Netzwerk die Informationen erhalten, sodass diese ein nahezu aktuelles globales Wissen haben. Auch hier sind die Nachrichten auf der Applikation, wie bspw. Fahrbefehle zum Fahrzeug aus Sicht des Fahrzeug fast vernachlässigbar.

Eine Hochrechnung der zu empfangenden Nachrichten auf eine Stunde ist in Abb. 6.13b dargestellt. Diese berücksichtigt sowohl Last- als auch Leerfahrten, sodass jeweils die doppelte Anzahl an Fahrten je Stunde aus Abb. 6.10 mit der Anzahl an empfangenen Nachrichten je Verhandlung multipliziert wird.

In Abb. 6.14 sind die minimale, die maximale und die durchschnittliche Anzahl an initial gesendeten Nachrichten und die dazu empfangenen Nachrichten für eine Verhandlung dargestellt. Auf den Topics *Prepare* und *Accept* sendet jeweils die Abholstation eine Nachricht, welche eine Verhandlung über eine Trajektorie initiiert. Empfangen wird diese von den verbleibenden $(N-1)$ Abholstationen. Auf den Topics *Promise*, *Commit*, *Accepted* und *Replication* versenden aufgrund des Verfahrens die verbleibenden $(N-1)$ Abholstationen Nachrichten, welche von $(N-1)\cdot(N-1)$ Abholstationen insgesamt empfangen werden. Dieser Graph spiegelt nicht die Weiterleitungen je Knoten in MINHTON gemäß dem Verfahren aus Abschn. 5.5 wider. Da auch hier für die Verhandlung der Trajektorien lediglich die Abholstationen zuständig sind, bleibt der Wert der initial versendeten Nachrichten konstant. Die Anzahl an empfangenen Nachrichten steigt folglich mit der Anzahl an Fahrerlosen Transportfahrzeugen an. Anzumerken gilt, dass auf den Topics *Promise* und *Accepted* die verbleibenden $(N-1)$ Abholstationen, welche der initiierenden Abholstation antworten, an alle Teilnehmer in MINHTON eine Nachricht schicken.

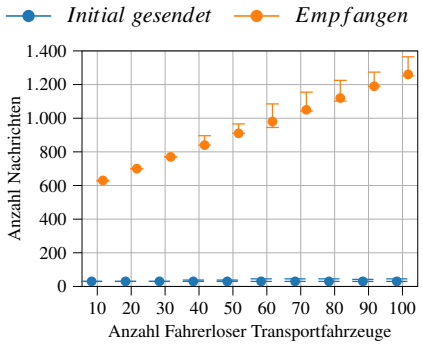


Abb. 6.14: Minimale, maximale und durchschnittliche Anzahl initial gesendeter Nachrichten und empfangener Nachrichten für die Verhandlung einer Fahrt

Dies ist auf die deterministische Verteilung des Broadcasts zurückzuführen, sodass insgesamt $(N-1)\cdot(N-1)$ Nachrichten versendet werden, obwohl nur $(N-1)$ Nachrichten notwendig sind. Daher ist eine (1:1)-Kommunikation bei steigender Anzahl an Abholstationen sinnvoller. Die Maxima resultieren aus der Anzahl an benötigten Verhandlungen für eine Trajektorie (vgl. Abb. 6.11a).

In Abb. 6.15 ist die gesamte Anzahl der empfangenen Nachrichten in Abhängigkeit von der Anzahl an Fahrerlosen Transportfahrzeugen für die unterschiedlichen Topics dargestellt. Mit steigender Anzahl an Fahrerlosen Transportfahrzeugen steigt auch die Anzahl an empfangenen Nachrichten, was mit der Anzahl an durchgeführten Transporten bzw. Verhandlungen korreliert. Hinzu kommt, dass auf den Topics *Promise*, *Commit* und *Accepted* $(N-1)$ Abholstationen Nachrichten initial verschicken. Auf dem Topic *Replication* versenden ebenfalls nur $(N-1)$ Abholstationen Nachrichten, allerdings werden diese auch an die Lieferstationen und die Fahrerlosen Transportfahrzeuge versendet, sodass die Anzahl an empfangenen Nachrichten die Anzahl der empfangenen Nachrichten auf den anderen Topics übersteigt.

Die Position eines Knotens in MINHTON und ihr Einfluss auf die Ausbreitungsgeschwindigkeit

Die logische Kommunikation zwischen allen Teilnehmern bzw. den Knoten in MINHTON ist dezentral organisiert, die physikalische Kommunikation hingegen zentral organisiert (vgl. Abb. 6.2). Wird eine Broadcast-Nachricht durch eine der Abholstationen auf dem Topic *Replication* initiiert, so müssen sich alle Knoten an der Verbreitung der Nachricht beteiligen, indem Datenpakte von einem Knoten empfangen und zu einem oder mehreren anderen Knoten gesendet werden. In dem hier vorliegenden Szenario existieren Knoten sowohl im kabelgebundenen als auch im kabellosen Medium. Wird ein Datenpaket an einen Knoten im WLAN geschickt, so geschieht der Medienwechsel über den WLAN Access Point. Soll nun ein FTF die

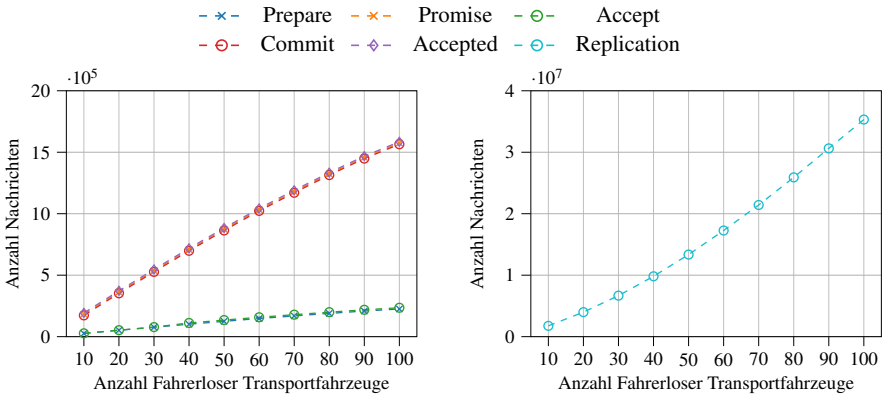


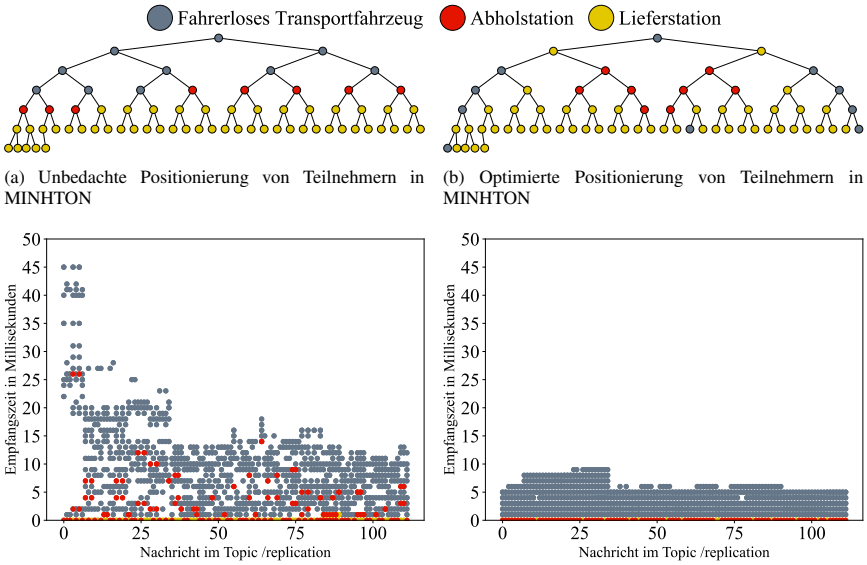
Abb. 6.15: Durchschnittliche Anzahl an empfangenen Nachrichten innerhalb einer Stunde, aufgeteilt je Topic, in Abhängigkeit von der Anzahl an Fahrerlosen Transportfahrzeugen

Nachricht an einen anderen Knoten weiterleiten, so wird diese Nachricht ebenfalls über WLAN an den Access Point geschickt. Befindet sich der Zielknoten auch im WLAN, so wird dieses erneut im Medium Luft über WLAN versendet. Folglich wird die Übertragungslatenz eines Datenpakets bei einer Weiterleitung im Medium Luft zweimal benötigt zwischen zwei Knoten.

Der Einfluss der Positionierung der Knoten in MINHTON bei der Verwendung von Ethernet und WLAN ist in Abb. 6.16 dargestellt. In den bisher betrachteten Szenarien waren die ersten N Knoten in MINHTON die Fahrerlosen Transportfahrzeuge, welche sich im kabellosen Medium Luft befinden, gefolgt von 8 Abholstationen und 50 Lieferstationen im kabelgebundenen Medium (vgl. Abb. 6.16a). In Abb. 6.16c sind die dazugehörigen Empfangszeiten für die Knoten zu Beginn der Simulation dargestellt. Empfängt ein Knoten Datenpakete von anderen Knoten, welche sich im kabelgebundenen Netzwerk befinden, so ist die dafür benötigte Zeit nahezu 0 s. Erfolgt ein Medienwechsel von kabelgebunden zu kabellos bzw. umgekehrt, so belaufen sich Empfangszeiten im einstelligen ms-Bereich. Kommunizieren bspw. die Knoten (*1:0*) und (*1:1*) miteinander, so werden Datenpakete lediglich über das kabellose Medium Luft versendet. Die durchschnittlichen Empfangszeiten liegen im Bereich von 10 ms bis 15 ms. Die zeitlichen Ausreißer am Anfang der Simulation hängen mit der Auflösung von einer IP-Adresse zu einer physikalischen Adresse mittels des ARP-Protokolls zusammen.

Werden die Knoten der FTF in MINHTON so positioniert, dass diese im Idealfall nur noch das Paket empfangen und nicht weiterleiten müssen, so hat dies einen Einfluss auf die Ausbreitungsgeschwindigkeit bzw. die Empfangszeiten aller Knoten. In Abb. 6.16b sind die Knoten, welche ein FTF in MINHTON repräsentieren, so gewählt worden, dass diese die Nachricht nur noch empfangen und nicht weiterleiten müssen. Die Empfangszeiten sind hierfür in Abb. 6.16d dargestellt. Hier leiten lediglich Knoten aus dem kabelgebundenen Netzwerk Datenpakete weiter. Teilnehmer im kabellosen Netzwerk empfangen diese nur noch.

Mit steigender Anzahl an FTF steigt die Wahrscheinlichkeit bei einem Broadcast in MINHTON, dass ein FTF ein Datenpaket neben dem Empfangen auch noch weiterleiten muss. Allerdings können die Positionen für Teilnehmer in MINHTON so gewählt werden, dass die Anzahl an Weiterleitungen aus dem kabellosen Netzwerk minimal ist. Sollen keine Weiterleitungen aus dem kabellosen Netzwerk erfolgen, so kann dies durch das Hinzufügen von Knoten im kabelgebundenen Netzwerk erreicht werden, sodass die FTF nur noch Datenpakete empfangen und nicht weiterleiten müssen, bei der hier vorgegebenen Anzahl an Sendern. Ist die Anzahl der Knoten bzw. der Teilnehmer im Netzwerk bekannt, so kann die Bestimmung der Positionen der Knoten mit den wenigsten Weiterleitungen dadurch erfolgen, dass für jeden Teilnehmer in MINHTON ein DAG berechnet wird, bei dem der Teilnehmer als initialer Sender angenommen wird. Die Blattknoten eines DAG sind die Knoten, welche Datenpakete nur noch erhalten und nicht weiterleiten. Die Schnittmenge aller DAG und deren Blattknoten bestimmt die Menge an Positionen, bei der die Anzahl an Weiterleitungen minimal wird. Zwar können mobile Teilnehmer in einem CPPS nicht ganz vermieden werden, allerdings kann die Anzahl an sendenden Teilnehmern reduziert werden, sodass WLAN, welches als limitierendes Medium in einem Netzwerk angesehen wird [PZC+16], nicht häufiger benutzt wird als nötig.



(a) Unbedachte Positionierung von Teilnehmern in MINHTON

(b) Optimierte Positionierung von Teilnehmern in MINHTON

(c) Empfangszeiten von Datenpakten bei unbedachter Positionierung der Teilnehmer in MINHTON

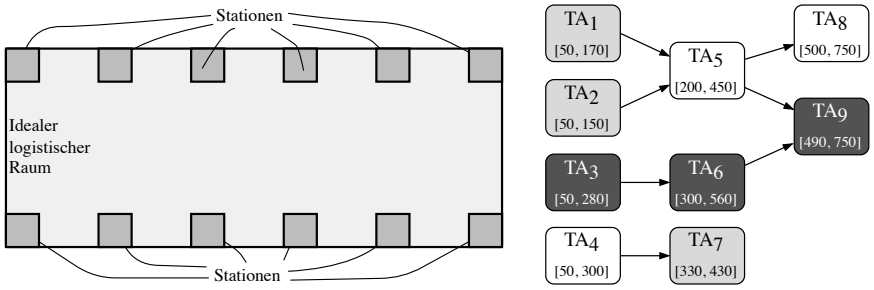
(d) Empfangszeiten von Datenpakten bei optimierter Positionierung der Teilnehmer in MINHTON

Abb. 6.16: Einfluss der Position von Knoten in MINHTON auf die Empfangszeit von Nachrichten bei der Verwendung von kabelgebundenen und kabellosen Medien

6.3 Der selbstorganisierende Materialfluss in einem CPPS

In diesem Abschnitt ist das Szenario des selbstorganisierenden Materialflusses an eine Produktion eines Automobilzulieferers angelehnt. Die Stationen verteilen sich auf eine Fläche von 1000 m^2 (vgl. Abb. 6.17a), bei der der Grundriss 50 $m \cdot 20$ m entspricht und sich an der Forschungshalle *LivingLab Zellulare Transportsysteme*[2] des Fraunhofer-Instituts für Materialfluss und Logistik IML in Dortmund orientiert. Teile dieses Abschnitts basieren auf den Vorarbeiten des Autors in [LDE+22, Tön22, Göd23].

Der Materialfluss für einen Produktionsauftrag besteht insgesamt aus neun Transportaufgaben (TA) (vgl. Abb. 6.17b). Innerhalb des Materialflusses existieren Abhängigkeiten, so darf bspw. TA_7 erst ausgeführt werden, wenn TA_4 abgeschlossen ist. Hierbei orientiert sich die Größe bzw. Tiefe des Vorranggraphen bzw. die Anzahl der Fertigungsschritte, welche innerhalb des Szenarios durchlaufen werden müssen, orientiert sich an einem großen Unternehmen (vgl. Abb. 3.3b). Zusätzlich sind für die Transportaufgaben temporale Randbedingungen definiert, zu welchem Zeitpunkt der Transport frühestens starten darf bzw. bis zu welchem Zeitpunkt der Transport abgeschlossen sein muss. Diese Zeitpunkte sind relativ zu der Erstellung des Materialflusses. Eine weitere Randbedingung ist, dass bestimmte Transportaufgaben nur von bestimmten FTF durchgeführt werden können. Als Optimierungskriterium wird die Bearbeitungsdauer (engl.: *makespan*) eines Materialflusses minimiert. In diesem Fall umfasst die Bearbeitungsdauer eines Materialflusses die Zeitspanne vom frühesten Start der ersten Transportaufgabe bis zu der Fertigstellung der letzten Transportaufgabe. Für die vorliegenden neun Transportaufgaben werden drei verschiedene Typen an FTF benötigt. Drei Transportaufgaben können nur von FTF des Typs *1* ausgeführt werden, drei Transportaufgaben können nur von FTF



(a) Verteilung der Stationen auf die Lagerhalle

(b) Verketteter Materialfluss von Transportaufgaben als Vorranggraph. Farbe spiegelt den Typen des benötigten Fahrerlosen Transportfahrzeugs wieder

Abb. 6.17: Exemplarische Darstellung des Szenarios und des damit verbundenen Materialflusses

[2] Fraunhofer-Institut für Materialfluss und Logistik IML – LivingLab Zellulare Transportsysteme: https://www.iml.fraunhofer.de/de/unser-institut/forschungshallenlabore/zft-halle.html

des Typs *2* ausgeführt werden, während die verbleibenden drei Transportaufgaben von FTF des Typs *2* oder *3* ausgeführt werden können. Die verwendeten Fahrerlosen Transportfahrzeuge unterscheiden sich neben der Kinematik auch in ihrem Ladungsträger und dem maximalen Ladungsgewicht (vgl. Tabelle 6.3).

Tabelle 6.3: Übersicht der verwendeten FTF

Parameter	**FTF1**	**FTF2**	**FTF3**
Ladungsträger	Box	KLT	KLT
Anzahl Ladungsträger	1	1	1
Max. Ladegewicht	30 kg	50 kg	25 kg
Ladezeit	2 s	10 s	5 s
Entladezeit	3 s	10 s	5 s
Kinematik	$a_{acc} = 4\,\mathrm{m/s^2}$ $a_{dec} = 4\,\mathrm{m/s^2}$ $v_{max} = 10\,\mathrm{m/s}$	$a_{acc} = 0{,}5\,\mathrm{m/s^2}$ $a_{dec} = 0{,}5\,\mathrm{m/s^2}$ $v_{max} = 1\,\mathrm{m/s}$	$a_{acc} = 1\,\mathrm{m/s^2}$ $a_{dec} = 1\,\mathrm{m/s^2}$ $v_{max} = 1\,\mathrm{m/s}$
Positionsaktualisierung	10 Hz		

Die Dezentralisierung des vorliegenden Anwendungsszenarios erfolgt durch eine konsequente Modularisierung (vgl. Abschn. 2.3), wie in Abb. 6.18 dargestellt. Das ERP-Modul erzeugt die Aufträge für den Materialfluss (MF) unter Verwendung der MFDL (vgl. Abschn. 4.4). Der hieraus entstehende logische MF-Agent ist ein eigenständiger Teilnehmer im CPPS und repräsentiert sich eigenständig. Die Auftragszuweisung erfolgt dezentral, sodass jeder logische MF-Agent dafür zuständig ist, sich selbstorganisierend fertigzustellen, und daher individuell mit jedem für ihn relevanten logischen FTF-Agenten verhandelt. Sowohl die logischen Agenten des Materialflusses als auch die logischen Agenten der FTF verhalten sich egoistisch bzw. *greedy*. Für den logischen MF-Agenten bedeutet dies, dass dieser sich schnellstmöglich fertigstellen will, während ein logischer FTF-Agent die Anzahl an abgearbeiteten Transportaufträgen maximieren will. Jeder logische FTF-Agent bietet seine Fähigkeiten und Dienste in Form von Services an, welche von den anderen logischen Agenten aufgefunden und verwendet werden. Grundlage für die logischen FTF-Agenten ist hierfür das Informationsmodell (vgl. Abschn. 4.1). Das physische FTF-Asset erhält lediglich Fahrbefehle bzw. die Steuerbefehle. Die Verwaltung der Auftragswarteschlange erfolgt im logischen FTF-Agenten unter der Verwendung eines Simple Temporal Network (STN). Im STN stellen die Knoten Zeitpunktvariablen bzw. Zeitereignisse dar und die gewichteten Kanten Ungleichheitsbedingungen zwischen Knoten [DMP91]. Die Gültigkeit der Randbedingungen kann innerhalb Polynomialzeit verifiziert werden. Auch das Hinzufügen von neuen Zeitpunkten und deren Randbedingungen ist ebenfalls in Polynomialzeit möglich, insbesondere dann, wenn neue Aufgaben hinzukommen bzw. entfernt werden können.

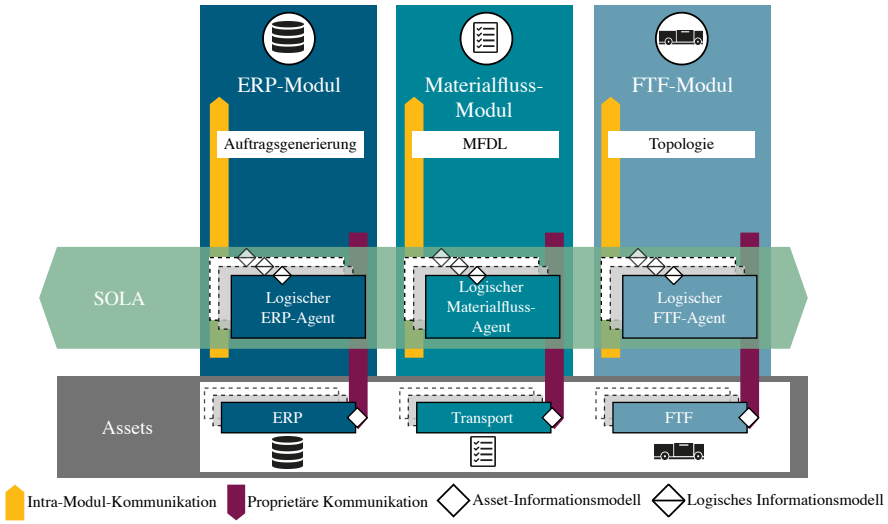


Abb. 6.18: Architektur zur Dezentralisierung des selbstorganisierenden Materialflusses innerhalb eines CPPS.

So hat jeder logische FTF-Agent nur das Wissen über seine vorliegenden Aufgaben [NMG16].

Für die Auftragsvergabe bestehen unterschiedliche Möglichkeiten [KHE15]. In dieser Arbeit basiert die Auftragsvergabe auf MURDOCH [GM02], einer Variation des marktbasierten Contract Net Protocol (CNP) [Smi80], welches eine (m:n)-Kommunikation mittels Publish-Subscribe und eine (1:1)-Kommunikation zwischen zwei Teilnehmern verwendet. Die Interaktion zwischen einem logischen MF-Agenten und einer Menge an logischen FTF-Agenten umfasst fünf Phasen:

1. **Call-for-Proposals**: Der logische MF-Agent schickt eine Anfrage mit den entsprechenden temporalen Randbedingungen an die Menge der logischen FTF-Agenten hinsichtlich der Fähigkeiten (z. B. Ladungsträgertyp oder Gewicht) auf das entsprechende Topic über die (m:n)-Kommunikation. Es erhalten daher nur solche FTF die Anfrage, welche diese Transportaufgabe erfüllen können. Innerhalb einer Anfrage können gleichzeitig mehrere Transportaufträge enthalten sein.
2. **Proposal**: Jeder logische FTF-Agent prüft, ob die Transportanfrage unter Berücksichtigung der Randbedingungen durchführbar ist. Unabhängig davon, ob der Transport durchgeführt werden kann oder nicht, wird diese Anfrage von einem logischen FTF-Agenten in einer (1:1)-Kommunikation an den logischen MF-Agenten beantwortet.
3. **Evaluation**: In dieser Phase evaluiert der logische MF-Agent die erhaltenen Angebote anhand der für den Transport relevanten Kriterien. In dieser Phase kann es einen, keinen oder mehrere Gewinner geben.

4. **Accept/Reject Proposals**: Der logische MF-Agent sendet dem Gewinner bzw. den Gewinnern in einer (1:1)-Kommunikation den Zuschlag zu. Diese antworten ebenfalls in einer (1:1)-Kommunikation, ob der Auftrag angenommen wird oder nicht. Die Verlierer der Auktion werden über die (m:n)-Kommunikation über den Status der Auktion bzw. des Transportauftrags im dazugehörigen Topic benachrichtigt. Im Folgenden wird ein bereits geplanter Transport nicht mehr hinsichtlich seiner Ausführungszeit verändert.
5. **Execution**: Nach Erhalt des Zuschlags ist ein logischer FTF-Agent für die Ausführung des Transports zuständig. Die Ausführung des Transports bzw. die Zustandsänderung eines FTF ist in Abb. 2.3 dargestellt.

In Abb. 6.19 ist die vereinfachte Interaktion zwischen den einzelnen Teilnehmern inkl. der verwendeten Topics dargestellt. Diese berücksichtigt sowohl die dezentrale Verhandlung, welche zwischen den logischen MF-Agenten und den logischen FTF-Agenten stattfindet, als auch die Auftragszuweisung und die Statusaktualisierungen. Für jede Klasse von FTF existiert jeweils ein Topic (‘/loadcarriertype/box/30’, ‘/loadcarriertype/klt/25’und ‘/loadcarriertype/klt/50’), auf welchem verhandelt wird. Auf diesen Topics initiiert jeder logische MF-Agent die *Call-for-Proposals*-Phase. Die Angebote (*Proposals*) der logischen FTF-Agenten werden in einer direkten Nachricht an den logischen MF-Agenten geschickt. Der logische FTF-Agent des Gewinners wird ebenfalls mit einer direkten Nachricht über die Auftragszuweisung informiert. Die Ablehnung bzw. die Abweisungen an die verbleibende Menge an logischen FTF-Agenten erfolgt erneut über das Topic für die entsprechende Klasse der FTF (vgl. Anhang E, Tabelle E.1). Statusänderungen bzw. Updates des Transports schickt der logische FTF-Agent direkt an den logischen MF-Agenten (vgl. Anhang E, Tabelle E.2). Für jeden logischen FTF-Agenten bzw. MF-Agenten existiert ein eigens Topic (*agv/status/uuid* bzw. *mf/status/uuid*), auf dem die Teilnehmer ihre Status für andere interessierte Teilnehmer informieren können. Dies ist nur der Vollständigkeit halber aufgeführt und wird nachfolgend nicht näher betrachtet.

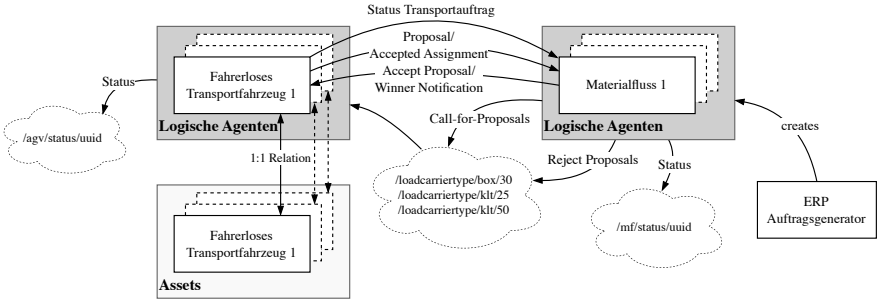


Abb. 6.19: Verwendete Topics und Interaktion zwischen den logischen Agenten

6.3.1 Simulation des Anwendungsszenarios

Die Parametrisierung der Simulation ist in der nachfolgenden Tabelle 6.4 dargestellt. Der Fanout wird auf $m = 2$ gesetzt, während die Anzahl an FTF im Bereich von $N = 4$ bis $N = 150$ variiert wird. Jedes Experiment wurde 5-mal – jeweils mit verschiedenen Startwerten für einen Zufallsgenerator – durchgeführt. Die Auftragsperiode wird mithilfe einer Normalverteilung im Bereich des Erwartungswerts μ erstellt, während die Standardabweichung $\sigma = 0,1 \cdot \mu$ entspricht. Der Umfang jeder Simulation entspricht 100 durchgeführten Materialflüssen, bei denen jeder Materialfluss neun Transportaufgaben enthält (vgl. Abb. 6.17b). Die Auswahl der Stationen erfolgt über eine Gleichverteilung, sodass jede Station im Durchschnitt gleich oft als Abholort bzw. Lieferort gewählt wird.

Tabelle 6.4: Übersicht der Parametrisierung zur simulativen Evaluation des selbstorganisierenden Materialflusses

Nr.	Parameter	Wert
1	Fanout $m \in \mathbb{N}$	$m = 2$
2	Anzahl an FTF $N \in \mathbb{N}$	$N \in \{4, 6, 8, \ldots, 150\}$
3	Anzahl an Wiederholungen	5
4	Anzahl an Stationen	12
5	Simulationszeit	100 Materialflüsse mit je 9 Transportaufgaben
6	Auftragsperiode T	Normalverteilung $\mu \in \{1,25\,\text{s}, 2,5\,\text{s}, 3,75\,\text{s}, ..., 80\,\text{s}\}, \sigma = 0,1 \cdot \mu$
7	Peer Discovery Antwortzeit	2 s
8	Zeit zur Abgabe eines Gebots	1 s
9	Abbruchkriterium der Simulation	Randbedingungen können nicht eingehalten werden
10	Zeit zur Annahme einer Transportanfrage	150 ms

Die Anzahl der Fahrerlosen Transportfahrzeuge (N) wird auf die unterschiedlichen FTF-Typen FTF1, FTF2 und FTF3 mithilfe des Wahrscheinlichkeitsvektors $\vec{w} = \begin{pmatrix} w_{\text{FTF1}} \\ w_{\text{FTF2}} \\ w_{\text{FTF3}} \end{pmatrix} = \begin{pmatrix} 0,2 \\ 0,5 \\ 0,3 \end{pmatrix}$ verteilt. Mit $\vec{z} = N \cdot \vec{w} = N \cdot \begin{pmatrix} w_{\text{FTF1}} \\ w_{\text{FTF2}} \\ w_{\text{FTF3}} \end{pmatrix} = \begin{pmatrix} N \cdot 0,2 \\ N \cdot 0,5 \\ N \cdot 0,3 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$ ergibt sich die Anzahl der verteilten FTF auf den jeweiligen Typ. Da nur ganze FTF existieren und verwendet werden, müssen die Werte jeweils abgerundet werden. Aufgrund dessen ergibt sich eine Anzahl an noch verbleiben-

den und noch nicht verteilten FTF (N_{Rest}), die mit $N_{\text{Rest}} = N - (\lfloor z_1 \rfloor + \lfloor z_2 \rfloor + \lfloor z_3 \rfloor)$ bestimmt werden kann. Um auf die vorgesehene Anzahl an Fahrerlosen Transportfahrzeugen N zu kommen, müssen zwei Fälle unterschieden werden. Ist $N_{\text{Rest}} = 1$, so wird die Anzahl der FTF für Typ FTF1 um 1 erhöht. Ist $N_{\text{Rest}} = 2$, so wird die Anzahl der FTF für Typ FTF1 und FTF2 jeweils um 1 erhöht. Es werden nur FTF der Typen FTF1 und FTF2 hinzugefügt, da diese die geringste Anzahl an Fahrzeugen stellen, welche Transporte für den entsprechenden Ladungsträger durchführen können. Fahrerlose Transportfahrzeuge des Typs FTF2 können auch Transporte des Typs FTF3 durchführen.

Zu Beginn der Simulation werden alle FTF gleichmäßig auf die Abholstationen verteilt. Es stehen maximal 24 FTF in einer Reihe mit einem Abstand von 1,5 m horizontal zueinander. Der vertikale Abstand der nächsten Reihe beträgt ebenfalls 1,5 m. Nach Ausführung eines Transports verweilt das FTF an seinem Ort. Die Versuchsdurchführung berücksichtigt hierbei die Bestimmung der Anzahl an Fahrerlosen Transportfahrzeugen, die benötigt werden, um die Materialflüsse mit den entsprechenden Transportaufgaben unter Einhaltung von deren temporalen Randbedingungen und Abhängigkeiten realisieren zu können. Ist eine Einhaltung der temporalen Randbedingungen nicht möglich, so wird die Simulation abgebrochen. Eine vereinfachte theoretische Betrachtung, wie viele Fahrerlose Transportfahrzeuge N in Abhängigkeit von der Auftragsperiode bzw. dem Zeitintervall T benötigt werden, ohne Berücksichtigung der Randbedingungen, sondern nur nach der durchschnittlichen Auslastung (DA), ist gegeben durch:

$$f(N,T) = \frac{DA_{\text{FTF}}}{T \cdot N} \tag{6.3}$$

In Abb. 6.20 ist die Gl. 6.3 dargestellt, welche die gültigen bzw. ungültigen Lösungen mit einer durchschnittlichen Auslastung eines FTF von 300 s bestimmt. Bei gleichbleibender Anzahl an Fahrerlosen Transportfahrzeugen und einer kürzeren Auftragsperiode T werden schnell ungültige Lösungen erzeugt. Ungültig bedeutet hier, dass die Menge an Fahrerlosen Transportfahrzeugen die benötigte Transportleistung nicht erbringen kann. Eine kürzere Auftragsperiode T hat zur Folge, dass die Anzahl an konkurrierenden Transportaufträgen des gleichen Typs steigt. Allerdings kann ein FTF nur einen Transport zu einem Zeitpunkt ausführen kann, die Auftragswarteschlange kann jedoch größer als 1 sein. Wird nun die Anzahl an Fahrerlosen Transportfahrzeugen erhöht, so besteht die Möglichkeit, dass die Anzahl an konkurrierenden Transporten des gleichen Typs möglich ist und somit eine gültige Lösung entsteht.

Insgesamt wurden 23.680 Simulationen gemäß der Parametrisierung aus Tabelle 6.4 durchgeführt. In Abb. 6.21 ist eine visuell aufbereitete grafische Darstellung abgebildet, welche den Lösungsraum basierend auf den für diese Arbeit durchgeführten Simulationen dargestellt. Die Auftragsperiode T wurde in einem Abstand von 1,25 s je Simulation mithilfe einer Normalverteilung und einer Standardabweichung von $\sigma = 0,1 \cdot \mu$ reduziert, die Anzahl an Fahrerlosen Transportfahrzeugen jeweils um 2 erhöht. In Abb. 6.21a sind somit diskrete Ergebnisse der Simulationen

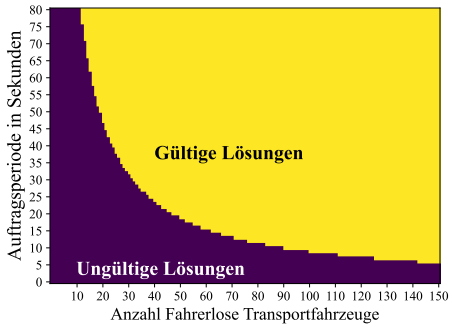
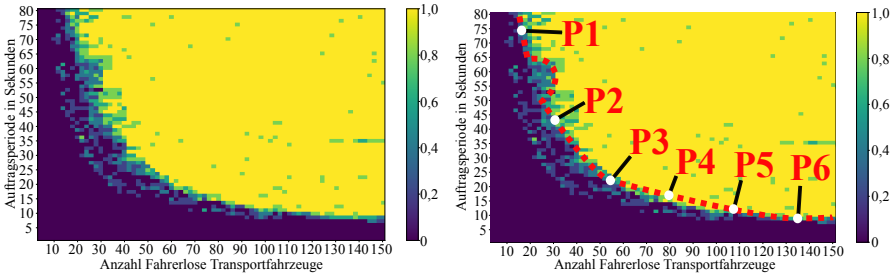


Abb. 6.20: Theoretisches Modell zur Bestimmung der benötigten Fahrerlosen Transportfahrzeuge nach Gl. 6.3 in Abhängigkeit von der Auftragsperiode für einen Materialfluss mit einer durchschnittlichen Auslastung von 300 s

und ihre Erfolgswahrscheinlichkeit dargestellt. Die Schwankungen bzw. die Fragmente in der Abbildung resultieren aus der Verwendung der Standardabweichung σ.

Im Folgenden wird sich auf einen kleinen Teil der Experimente des Lösungsraums beschränkt. Dafür ist eine Isolinie in Abb. 6.21b bei einer Erfolgsrate von $p = 0,8$ eingezeichnet, was vier erfolgreich durchgeführten Simulationen (und einer ungültigen Simulation) entspricht. Auf dieser Isolinie wurden sechs Punkte ausgewählt, dargestellt in Tabelle 6.5. Zur Durchführung von 100 Materialflüssen, welche mit einer Auftragsperiode T und der Hilfe einer Normalverteilung erstellt werden, auch eine bestimmte Anzahl an Fahrerlosen Transportfahrzeugen (Parameter 3–5) auch einige logische MF-Agenten benötigt. Bei der Ausführung des ersten Materialflusses wird erst ein logischer MF-Agent benötigt. Sobald der nächste Materialfluss abgearbeitet werden soll, wird geprüft, ob ein logischer MF-Agent verfügbar ist. Ist dem nicht der Fall, so wird ein neuer logischer MF-Agent dem System hinzugefügt. Dies führt dazu, dass die Anzahl an parallel existierenden logischen MF-Agenten variieren kann, aber in den hier vorliegenden Experimenten nach oben beschränkt ist (Parameter 6). Die Anzahl der logischen MF-Agenten hängt u. a. von der Standardabweichung der Normalverteilung bei der Auftragsperiode und der Durchlaufzeit eines Materialflusses ab. Beendet ein logischer MF-Agent seinen Materialfluss, so verlässt dieser nicht das Netzwerk, sondern wird den nächsten Materialfluss abarbeiten. Zur Verdeutlichung der Systemkomplexität beinhaltet diese Tabelle eine Auflistung, wie viele Fahrerlose Transportfahrzeuge je FTF-Typ existieren und wie viele logische MF-Agenten parallel bzw. konkurrierend auf die Fahrerlosen Transportfahrzeuge zugreifen möchten. Sowohl die Anzahl der Fahrerlosen Transportfahrzeuge als auch die Anzahl der parallel existierenden Materialflüsse haben einen Einfluss darauf, wie viele Teilnehmer in einem Topic jeweils existieren (Parameter 8–10).



(a) Diskrete Simulationsergebnisse

(b) Diskrete Simulationsergebnisse mit einer Isolinie bei $p = 0,8$ und den betrachteten Experimenten P1 bis P6

Abb. 6.21: Ergebnisse der Simulation

Tabelle 6.5: Übersicht der ausgewerteten experimentellen Setups auf der Isolinie aus Abb. 6.21b

Nr.	**Parameter**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
1	Auftragsperiode T in Sekunden	78,75	63,75	37,5	26,25	17,5	8,75
2	Gesamtanzahl FTF N	16	30	54	78	108	138
3	Anzahl Typ FTF1	3	6	11	16	22	28
4	Anzahl Typ FTF2	8	15	27	39	54	69
5	Anzahl Typ FTF3	5	9	16	23	32	41
6	maximale Anzahl paralleler Materialflüsse	8	10	17	24	34	66
7	maximale Anzahl an Entitäten im Management Overlay	24	40	71	102	142	204
8	maximale Anzahl an Entitäten im Topic '/loadcarriertype/box/30'	11	16	28	40	56	94
9	maximale Anzahl an Entitäten im Topic '/loadcarriertype/klt/50'	16	25	44	63	58	135
10	maximale Anzahl an Entitäten im Topic '/loadcarriertype/klt/25'	13	19	33	47	66	107

Betrachtung des Systems/ Logistische Kennzahlen

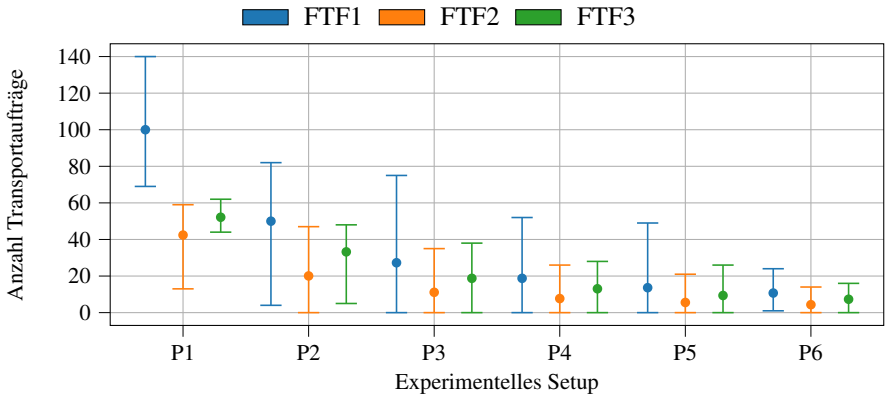
In Tabelle 6.6 ist die minimale, die maximale und die durchschnittliche Durchlaufzeit eines einzigen Materialflusses dargestellt. Dass sich die Werte zu den unterschiedlichen Experimenten kaum unterscheiden, hängt zum einen damit zusammen, dass immer der gleiche Materialfluss bestehend aus neun Transportaufträgen erstellt wird und zum anderen damit, dass die Auswahl der Abhol- und Lieferstationen gleichverteilt ist. Eine Schätzung für die Anzahl an benötigten logischen MF-Agenten kann mit $\frac{\text{maximale Durchlaufzeit Materialfluss}}{\text{Auftragsperiode } T}$ berechnet werden.

Tabelle 6.6: Minimale, maximale und durchschnittlichen Durchlaufzeit je Materialfluss

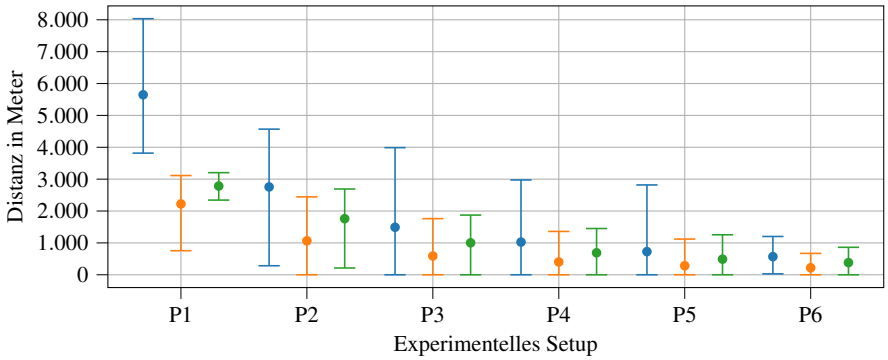
Materialfluss	**P1-P6**
durchschnittliche Durchlaufzeit	549 s
minimale Durchlaufzeit	532 s
maximale Durchlaufzeit	576 s

In Abb. 6.22 sind logistische Kennzahlen, bezogen auf die Fahrerlosen Transportfahrzeuge, dargestellt. Abb. 6.22a stellt die durchgeführten Transportaufträge je Fahrzeug da. Wenn die Anzahl an Fahrerlosen Transportfahrzeugen steigt und die Auftragsperiode T sinkt, dann sinkt auch die durchschnittliche Anzahl an durchgeführten Transporten je FTF bzw. je FTF-Typ. Sinkt das Minimum für einen FTF-Typ auf 0, so ist dies ein Indikator dafür, dass in dieser Kategorie zu viele Fahrzeuge verwendet werden. Dies hat auch Einfluss auf die zurückgelegte Distanz eines FTF (vgl. Abb. 6.22b) und dessen Standzeit (vgl. Abb. 6.22c). Die gefahrene Distanz beinhaltet sowohl die Leerfahrten als auch die Lastfahrten. Es ist anzumerken, dass es sich hier um eine dezentrale Zuweisung der Transportaufgaben handelt. In einer zentralisierten Lösung, bei der eine Heuristik wie bspw. *Simulated Annealing* oder *Tabu Search* [VRM+16] verwendet wird, kann dies ggf. zu einem besseren Ergebnis führen [FdS+21]. Besser bedeutet in diesem Kontext, dass die Anzahl der Fahrerlosen Transportfahrzeuge eines Typs geringer ausfällt und somit die Standzeit je FTF reduziert wird.

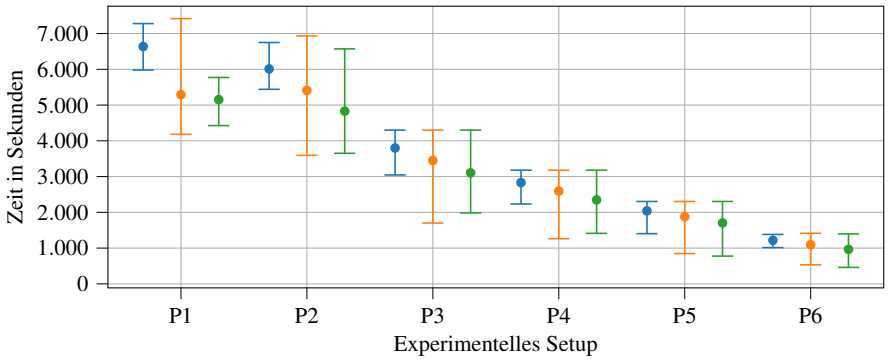
Die durchschnittliche Standzeit der FTF in den Experimenten P1–P6 nimmt ab. Dies hängt damit zusammen, dass in den Experimenten die Abarbeitung von 100 Materialflüssen betrachtet wird. Steigt die Anzahl an Fahrerlosen Transportfahrzeugen oder fällt die Auftragsperiode T in einem Experiment, so reduziert sich die Durchlaufzeit der Simulation, weil die darin enthaltenen Transporte parallel abgearbeitet werden.



(a) Anzahl ausgeführter Transportaufträge je FTF-Typ



(b) Gefahrene Distanz je FTF-Typ



(c) Standzeit je FTF-Typ

Abb. 6.22: Minimale, maximale und durchschnittliche Leistungswerte eines Fahrerlosen-Transportfahrzeugs-Typs

Betrachtung der Kommunikation

Bevor der aus diesem Szenario entstandene Datenverkehr evaluiert wird, erfolgt eine genauere Betrachtung, in welche nachfolgenden Kategorien dieser aufgeteilt ist und welche Kosten darin enthalten sind.

- **Management Overlay und Erstellung der Topic-Bäume**:
  - Aufbau (Join) und Abbau (Leave) des Management Overlays (vgl. Abschn. 5.3).
  - Peer Discovery: Neben den Informationen, welche Peers welche Fähigkeiten und Eigenschaften besitzen (vgl. Abschn. 5.4), wird diese Funktionalität dafür benutzt, um die Einstiegspunkte für die jeweiligen drei Topics zu finden.
  - Die Erstellung des jeweiligen Topic-Baums: Hierfür wird eine vereinfachte Form von MINHTON verwendet, da die Funktionalität der Peer Discovery nicht benötigt wird.
- **Nachrichtenverteilung in einer Gruppe – (m:n)-Kommunikation**
  - In einer Gruppe von Teilnehmern wird eine Nachricht unter allen Teilnehmern des entsprechenden Baumes verteilt (vgl. Abschn. 5.5). Für jedes Topic, also für jeden FTF-Typen, existiert ein eigener Baum. Hier wird die Peer-Discovery-Funktionalität nicht benötigt, sondern nur der Broadcast. Die Verwendung des Broadcasts erfolgt in den Interaktionsphasen *Call-for-Proposals* und *Reject Proposals* (vgl. Abschn. 6.3).
- **Applikation – (1:1)-Kommunikation**
  - Der hier entstandene Datenverkehr in der (1:1)-Kommunikation resultiert aus der (m:n)-Kommunikation bzw. Interaktion zwischen den logischen MF-Agenten und den logischen FTF-Agenten. Die (1:1)-Kommunikation beinhaltet die Statusaktualisierung des ausgeführten Transportauftrags (vgl. Anhang E.1) und die Interaktionsphasen *Proposal* und *Accept Proposal* (vgl. Anhang E.2).

Anzumerken gilt, dass der hier verursachte Datenverkehr abhängig von Applikation und dessen Interaktion ist. Ändert sich die Applikation und die darin enthaltene Interaktion, so hat dies auch Einfluss auf den Kommunikationsaufwand.

Kommunikationsaufwand im Management Overlay und zur Erstellung der Topic-Bäume

In dem hier vorliegenden Szenario treten die logischen Agenten dem Management Overlay in einer bestimmten Reihenfolge bei. Begonnen wird mit den logischen FTF-Agenten des Typs FTF1, gefolgt von den logischen FTF-Agenten des Typs FTF2 und des Typs FTF3. Dementsprechend bedeutet dies, dass die logischen MF-Agenten zuletzt dem Management Overlay beitreten. Diese Reihenfolge ist nicht zwingend notwendig, wird aber in den hier untersuchten Experimenten verwendet, um das Verhalten nachvollziehen zu können. In Abb. 6.23 ist die minimale, die maximale und die durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten dargestellt, die bei dem Aufbau des Management Overlays und dem Aufbau des jeweiligen Topic-Baums benötigt wird. In einem Topic-Baum erfolgt die Verhandlung der Transportaufträge zwischen den logischen MF-Agenten und logischen FTF-Agenten des entsprechenden FTF-Typs.

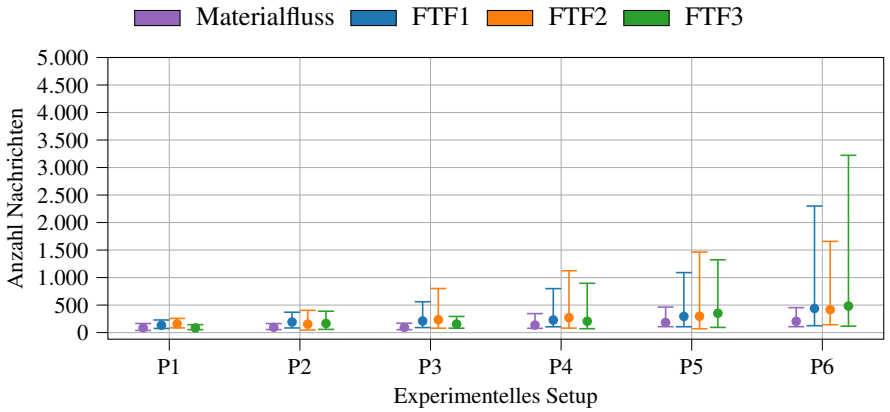


Abb. 6.23: Minimale, maximale und durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten für den Aufbau des Management Overlays und aller Topic-Bäume, aufgeteilt nach Typ des logischen Agenten

Im Folgenden werden die versendeten und empfangenen Nachrichten aufgeteilt und separat betrachtet, wie in den nachfolgenden Abbildungen Abb. 6.24 und vgl. Abb. 6.25 dargestellt.

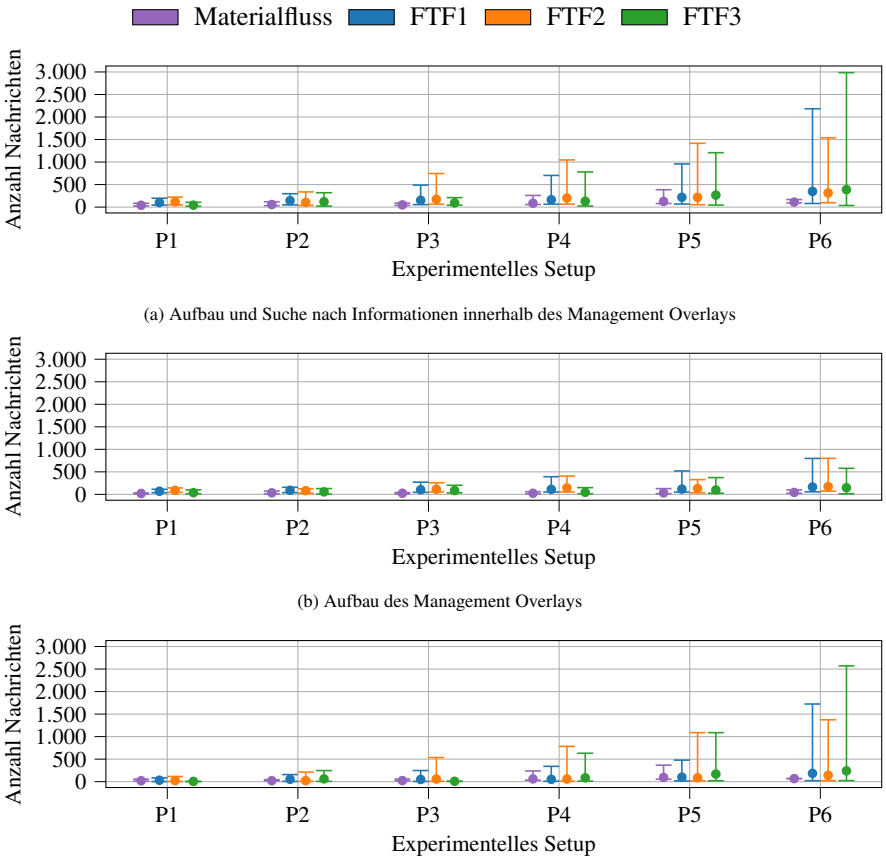
In Abb. 6.24a sind die gesamten Kosten für den Aufbau des Management Overlays und die Kosten der Suche nach den Einstiegspunkten dargestellt (vgl. Abb. 6.24a). In den anderen zwei Teilbildern sind die Kosten für die jeweilige Teilfunktionalität dargestellt, die Kosten für den Aufbau des Management Overlays (vgl. Abb. 6.24b) und die Kosten für Suche nach den Einstiegspunkten für die entsprechenden Topics (vgl. Abb. 6.24c).

Alle logischen Agenten sind Teilnehmer des Management Overlays, welches die Baumstruktur MINHTON verwendet. Bei dem Aufbau des Management Overlays

sendet ein neuer Knoten seine Anfrage an den Wurzelknoten. In einem Szenario mit N Teilnehmern erhält der Wurzelknoten auch $N-1$ Anfragen zum Betreten des Netzwerks. Um die Last des Wurzelknotens zu reduzieren, könnte ein Knoten, welcher dem Management Overlay beitreten will, eine oder mehrere *bootstrapping nodes* kontaktieren. Diese besonderen Knoten sind bspw. über eine statische IP erreichbar und könnten die Anfrage an Knoten nahe dem nächsten freien Platz weiterleiten. Da es sich bei MINHTON um einen nullbalancierten, vollständigen Baum handelt, der nach einer festen Struktur aufgebaut wird, werden auch immer dieselben Knoten an den jeweiligen Positionen in Abhängigkeit von der Anzahl der Teilnehmer im Baum kontaktiert. Das bedeutet, dass der Wurzelknoten eine Anfrage zum Betreten des Netzwerks immer an seine Adjazenten weiterleitet. Je mehr Knoten in MINHTON existieren, desto größer ist die Baumhöhe, was wiederum bedeutet, dass die Adjazenten vom Wurzelknoten sich auf einem höheren Level befinden. Die größten Kosten beim Aufbau des Management Overlays sind die Kosten für die Aktualisierung der Nachbarn (vgl. Abb. 6.24b bzw. Abschn. 5.3.2).

Bevor ein Baum für ein Topic aufgebaut werden kann, muss zunächst nach dem Einstiegspunkt gesucht werden. Dies geschieht über eine Peer-Discovery-Anfrage im Management Overlay und der Adressierung der CDSs bzw. der darin enthaltenen DSNs. Zu berücksichtigen ist, dass die Wahl eines DSN von der Position in MINHTON abhängt. Im Experiment P6 existieren maximal 204 Knoten gleichzeitig im Management Overlay. Erst nach dem alle logischen FTF-Agenten dem Management Overlay beigetreten sind, wird nach dem zugehörigen Topic gesucht. Somit ist Knoten (*6:34*) der Knoten, welcher am meisten Nachrichten in dieser Teilfunktionalität versendet und empfängt. Je nach Konfiguration des Experiments variiert neben der maximalen Anzahl an existierenden DSN innerhalb des Management Overlays auch, welcher logische FTF-Agent des jeweiligen Typs diese Position belegt. Die Kosten für die Suche nach Informationen sind im Vergleich zu den Kosten für den Aufbau des Management Overlays hoch. Allerdings werden die Kosten für das Betreten und Verlassen für einen logischen Agent im Netzwerk vernachlässigbar, je länger dieser im Management Overlay verbleibt.

In Abb. 6.25 sind die Kosten für die Erstellung der verschieden Topic-Bäume dargestellt. Während in den jeweiligen Topic-Bäumen nur logische FTF-Agenten eines FTF-Typs existieren und somit der Baum homogen in Bezug auf den FTF-Typ ist, existieren darüber hinaus noch die logischen MF-Agenten. Sowohl die durchschnittliche Anzahl an gesendeten und empfangen Nachrichten als auch die maximale Anzahl sind beim Typ FTF2 am größten (vgl. Abb. 6.25b). Dies hängt damit zusammen, dass von diesem Typ am meisten Fahrzeuge existieren.

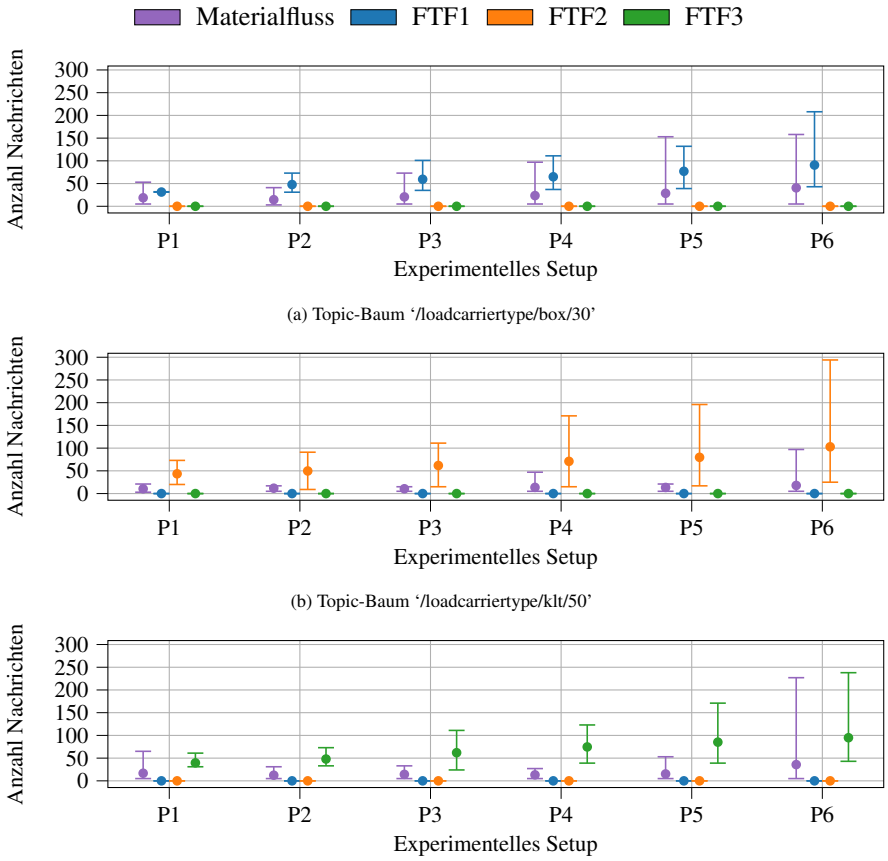


(a) Aufbau und Suche nach Informationen innerhalb des Management Overlays

(b) Aufbau des Management Overlays

(c) Suchen nach Informationen innerhalb des Management Overlays

Abb. 6.24: Minimale, maximale und durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten für den Aufbau das Management Overlay und der Suche nach Informationen innerhalb des Management Overlays, aufgeteilt nach Typ des logischen Agenten



(a) Topic-Baum '/loadcarriertype/box/30'

(b) Topic-Baum '/loadcarriertype/klt/50'

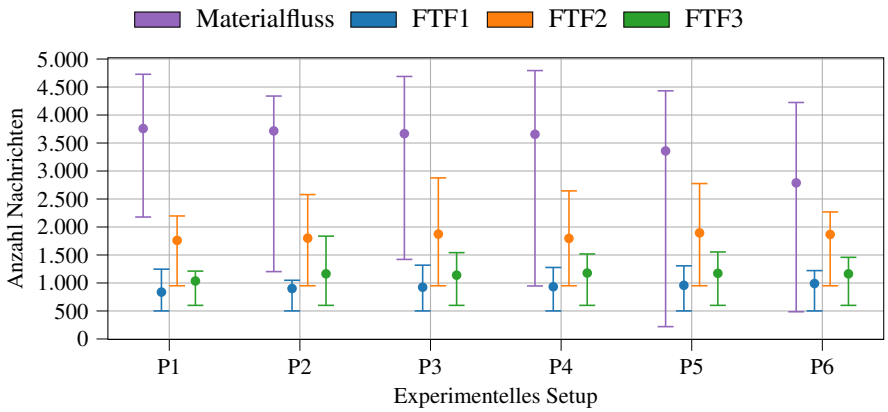
(c) Topic-Baum '/loadcarriertype/klt/25'

Abb. 6.25: Minimale, maximale und durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten für den Aufbau eines Topic-Baums, aufgeteilt nach Typ des logischen Agenten

Nachrichtenverteilung in einer Gruppe – (m:n)-Kommunikation

Jeder logische MF-Agent ist in jedem der drei Topics vertreten. Wird nun von einem logischen MF-Agenten ein Transportauftrag auf einem Topic ausgeschrieben, so erhalten auch alle anderen verbleibenden logischen MF-Agenten die Transportanfrage. Dies gilt auch, wenn ein logischer MF-Agent die logischen FTF-Agenten darüber informiert, dass diese den Transportauftrag nicht erhalten haben.

Die Anzahl der gesendeten und empfangenen Nachrichten der logischen FTF-Agenten ist nahezu konstant und somit unabhängig von der Anzahl der vorhandenen Fahrzeuge (vgl. Abb. 6.26). Der Einfluss ist gegeben durch die Anzahl an durchzuführenden Materialflüssen bzw. der darin verhandelten Anzahl an Transportaufträgen, also die Interaktion zwischen den Teilnehmern.



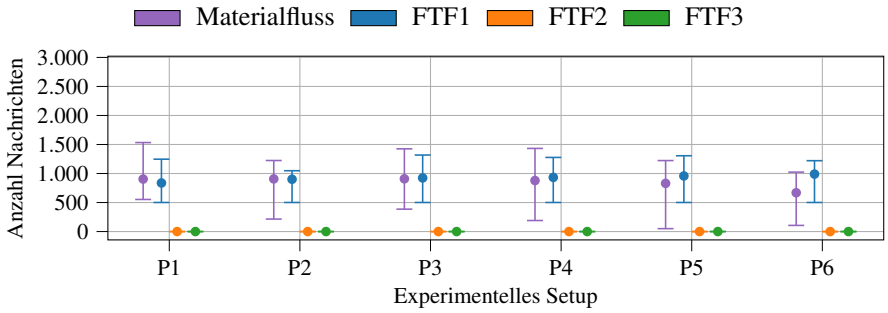
(a) Verteilung von Statusinformationen – (m:n)-Kommunikation

Abb. 6.26: Minimale, maximale und durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten für die (m:n)-Kommunikation, aufgeteilt nach Typ des logischen Agenten

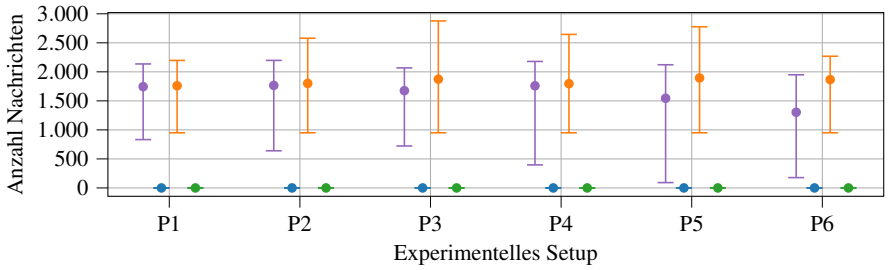
Die initialen Sender auf den Topics sind lediglich die logischen MF-Agenten. Je nach Position im Baum werden einige logische Agenten die Nachrichten nur empfangen und nicht an andere logische Agenten im Baum weiterleiten. Dies sind unter anderem die Knoten, welche sich am Rand des Baumes befinden, und keine initialen Sender sind. Folglich helfen die logischen FTF-Agenten bei der Verteilung der Nachrichten mit, auch die, welche keinen Transportauftrag ausführen. Hinzu kommt, dass die logischen MF-Agenten, deren letzter Transport des Materialflusses fertiggestellt wurde, sowohl im Management Overlay als auch im Topic-Baum verbleiben. Hierdurch steigen der Mittelwert und Maximalwert über die Zeit. Diese logischen MF-Agenten könnten das Management Overlay und die Topic-Bäume verlassen, würden jedoch kurze Zeit später, sofern mehr als 100 Materialflüsse abgearbeitet werden, erneut benötigt werden. Dies hätte zur Folge, dass neue logische MF-Agenten dem Management Overlay und den Topic-Bäumen beitreten müssten

und ggf. höhere Kosten dadurch entstehen. Denn je nach Position des Knoten im Management Overlay müsste ein Nachfolger bestimmt werden (vgl. Abschn. 5.3).

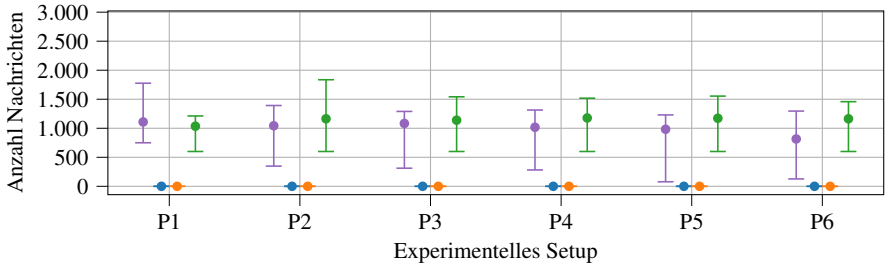
In Abb. 6.27 ist die minimale, die maximale und die durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten für das jeweilige Topic dargestellt. Auch hier ist zu erkennen, dass die durchschnittliche Last nahezu konstant ist und auf welchen Topics welche logischen Agenten miteinander kommunizieren, unabhängig von der Anzahl an vorhandenen logischen Agenten.



(a) Topic '/loadcarriertype/box/30'



(b) Topic '/loadcarriertype/klt/50'



(c) Topic '/loadcarriertype/klt/25'

Abb. 6.27: Minimale, maximale und durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten für die (m:n)-Kommunikation für das jeweilige Topic, aufgeteilt nach Typ des logischen Agenten

Applikation – (1:1)-Kommunikation

In Abb. 6.28 ist die minimale, die maximale und die durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten dargestellt, welche durch die direkte Interaktion zwischen den logischen Agenten entsteht.

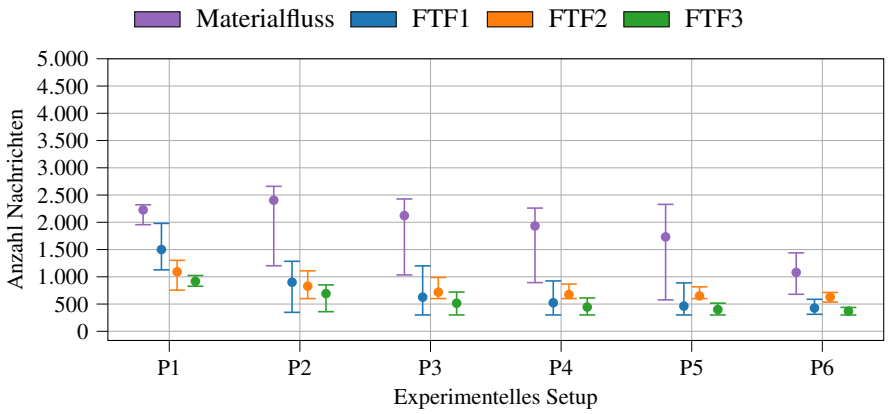


Abb. 6.28: Minimale, maximale und durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten auf Applikationsschicht, bei der eine (1:1)-Kommunikation verwendet wird, aufgeteilt nach Typ des logischen Agenten

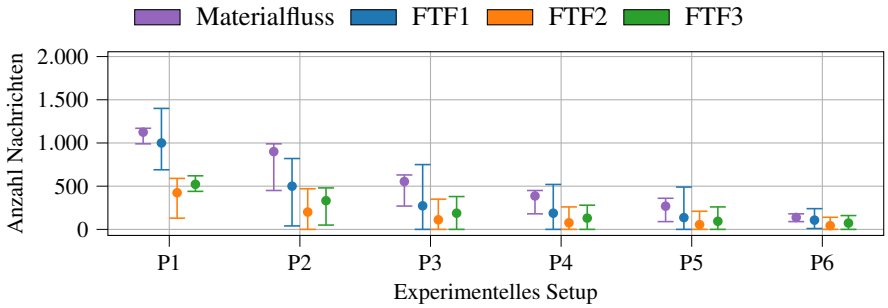
Je näher das Minimum und das Maximum der logischen MF-Agenten beieinander liegen, desto früher ist die maximale Anzahl an logischen MF-Agenten erreicht. Im Umkehrschluss bedeutet dies, dass je weiter diese Werte auseinander liegen, desto später wird ein weiterer logischer MF-Agent benötigt. Entsprechend ist die Wiederverwendungsrate hoch und die Verteilung der Materialflüsse auf die einzelnen MF-Agenten ausgeglichen. In P2 wird bspw. erst zur Mitte der Simulationszeit ein weiterer logischer MF-Agent benötigt, sodass dieser in der verbleibenden Simulationszeit insgesamt weniger Materialflüsse abarbeiten kann. Auch die Auftragsperiode T inkl. ihrer Standardabweichung und die Auslastung der jeweiligen logischen FTF-Agenten haben einen Einfluss auf die Anzahl der gesendeten und empfangenen Nachrichten bzw. auf die Anzahl an benötigten logischen MF-Agenten. Je häufiger Materialflüsse erstellt werden, desto mehr logische Agenten werden hierfür benötigt.

Mit einer steigenden Anzahl an Fahrerlosen Transportfahrzeugen sinkt auch die Anzahl an gesendeten und empfangenen Nachrichten je logischen FTF-Agenten. Das hängt zum einen damit zusammen, dass sich die Last der Transportaufträge über die verfügbaren FTF verteilt und zum anderen damit, dass nicht mehr alle Fahrzeuge aufgrund ihrer Auslastung auf alle Transportanfragen reagieren. Auch hier gilt, je weiter das Minimum und das Maximum voneinander entfernt liegen, desto ungleichmäßiger sind die Transportaufträge verteilt. Auch wenn ein Fahrzeug keinen

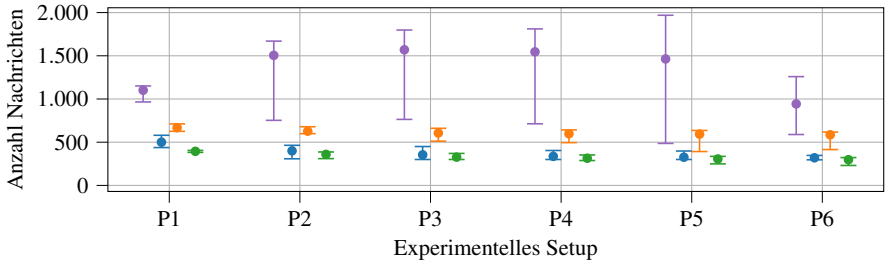
einzigen Transportauftrag erhält, so kann es dennoch auf eine Transportanfrage ein Gebot abgeben.

In Abb. 6.29 sind die Kosten der gesendeten und empfangenen Nachrichten auf Applikationsschicht nach ihrer Teilfunktionalität dargestellt. Die Statusänderungen eines Transportauftrags (vgl. AnhangE.1) werden lediglich von einem logischen FTF-Agenten an den logischen MF-Agenten gesendet (vgl. Abb. 6.29a). Somit sind in dieser Abbildung die Nachrichten dargestellt, die ein logischer FTF-Agent versendet und ein logischer MF-Agent empfangen hat. Werden keine Nachrichten von einem logischen FTF-Agenten versendet, so liegt das daran, dass dieser keinen einzigen Transportauftrag abgearbeitet hat. Mit der steigenden Anzahl an logischen FTF- bzw. MF-Agenten sinkt die durchschnittliche Anzahl an gesendeten bzw. empfangen Nachrichten. Wie bei der Auslastung der Fahrzeuge gilt auch hier, je näher das Maximum und das Minimum beieinander liegen, desto ausgeglichener ist die Verteilung der Transportaufträge (vgl. Abb. 6.22a).

In Abb. 6.29b sind die Nachrichten dargestellt, die bei der Interaktion zwischen einem logischen FTF-Agenten und den logischen MF-Agenten versendet und empfangen werden. Der Durchschnitt an gesendeten und empfangen Nachrichten für die logischen MF-Agenten steigt, je länger die maximale Anzahl an logischen MF-Agenten erreicht ist. Je später ein logischer MF-Agent dem Szenario beitritt, desto niedriger liegt das jeweilige Minimum. Auch wenn ein Fahrzeug keinen Transportauftrag zugewiesen bekommt, so nimmt diese stets an den Verhandlungen teil.



(a) Statusänderungen eines Transportauftrags, welche nur von den logischen FTF-Agenten an die logischen MF-Agenten gesendet werden



(b) Verhandlung über die Zuweisung eines Transportauftrags

Abb. 6.29: Minimale, maximale und durchschnittliche Anzahl an gesendeten und empfangenen Nachrichten auf Applikationsschicht, bei der eine (1:1)-Kommunikation verwendet wird, aufgeteilt nach Typ des logischen Agenten

Zeitlicher Verlauf der Kommunikation

Die Anzahl an gesendeten und empfangenen Nachrichten je logischem FTF-Agenten bleibt mit steigender Anzahl an logischen Agenten konstant. Die Gesamtanzahl an gesendeten und empfangenen Nachrichten hingegen wird mit der Anzahl an logischen Agenten steigen und über die Zeit steigen. In Abb. 6.30 ist zeitliche Verlauf der entstandenen Kommunikation für ein Experiment mit 30 Fahrerlosen Transportfahrzeugen und einer Auftragsperiode von 63,75 s dargestellt. Hierfür wurde die Zeit quantisiert, sodass die gesamte Simulation in der Abbildung darstellbar ist. Daher entspricht eine Zeiteinheit in der Abbildung 60 s in der Simulation. Die Anzahl an Nachrichten ist daher die kumulierte Summe der Nachrichten in diesem kurzen Zeitraum.

Zu Beginn der Simulation treten die logischen Agenten der Fahrerlosen Transportfahrzeuge und der Materialflüsse dem Netzwerk bei (grün). Ab dem Moment, in dem der erste logische Materialfluss-Agent erstellt wird und dieser dem Netzwerk beitritt, beginnt die Abarbeitung der Transportaufträge. Die Verhandlung der Transportaufträge erfolgt über die (m:n)-Kommunikation (orange) im jeweiligen Topic-Baum und über die direkte (1:1)-Kommunikation (blau). Sind nicht genügend logische MF-Agenten vorhanden, so wird ein neuer logischer MF-Agent dem System hinzugefügt, wie bspw. bei ca. 3800 s. Findet keine (m:n)-Kommunikation mehr statt, sondern lediglich eine (1:1)-Kommunikation, so sind alle Verhandlungen abgeschlossen und die Transportaufträge zugewiesen. Am Ende der Simulation, bei ca. 6.800 s, sind die Transportaufträge des letzten Materialflusses verhandelt und es erfolgt lediglich die Abarbeitung.

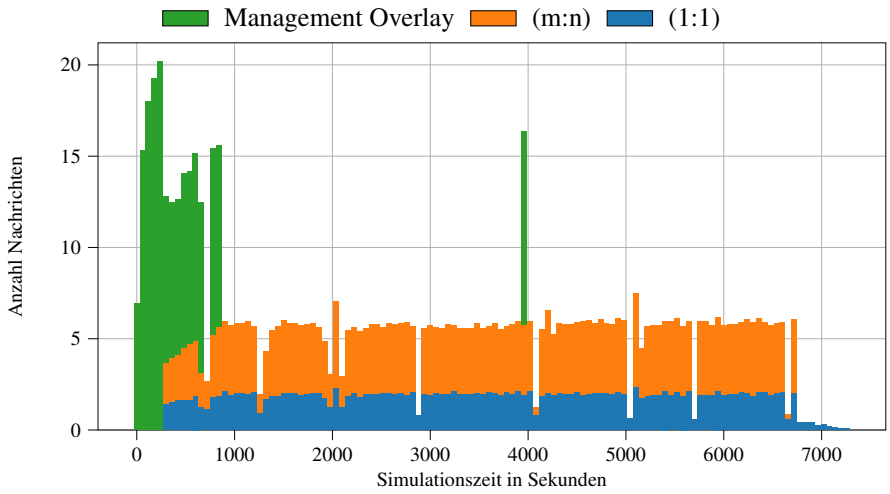


Abb. 6.30: Zeitlicher Verlauf der Interaktion und Kommunikation

Kapitel 7
Zusammenfassung und Ausblick

Der Einsatz von Fahrerlosen Transportfahrzeugen für unterschiedliche Aufgabengebiete rückt mehr und mehr in den (industriellen) Fokus, bei dem diese mehr und mehr Autonomie erhalten sollen. Diese Arbeit erforschte daher den dezentral organisierten Transport in einem cyberphysischen Produktionssystem, um zur Selbstorganisation in der Intralogistik beizutragen.

In Kapitel 1 erfolgte zunächst eine Einführung in die Thematik der Selbstorganisation in der Intralogistik, indem die Arbeit motiviert und wissenschaftlich eingeordnet wurde. Das Kapitel 2 befasste sich zum einen damit, wie der allgegenwärtige Begriff Industrie 4.0 in der vorliegenden Arbeit zu verstehen ist und zum anderen mit dem Fördern und Transportieren mittels Fahrerloser Transportfahrzeuge. Anschließend erfolgte eine Zusammenführung beider vorangegangener Themen, woraus ein Konzept für einen selbstorganisierenden Materialfluss entstanden ist, der auf einer konsequenten Dezentralisierung basiert und von einer konsequenten Modularisierung geprägt ist. Dieses umgesetzte Konzept ermöglichte die Dekomposition der starren monolithischen Automationspyramide in eine Menge von lose gekoppelten Entitäten.

Kapitel 3 widmete sich der Vernetzung von Entitäten in einem cyberphysischen Produktionssystem. Hierbei wurde zunächst eine Definition für eine Entität und eine Quantifizierung in produzierenden Unternehmen herausgearbeitet. Ein weiterer Schwerpunkt in diesem Kapitel war die Systemorganisation der Kommunikation und die Möglichkeiten zur Vernetzung innerhalb des cyberphysischen Produktionssystems.

In Kapitel 4 erfolgte die Erstellung eines ganzheitlichen Informationsmodells für die Intralogistik, welches aus einer Selbstbeschreibung des Betriebsmittels, in Form eines Fahrerlosen Transportfahrzeugs, und eines Prozesses besteht. Hierfür wurde der Transport als Bindeglied zwischen einem Fahrerlosen Transportfahrzeug und einem Materialfluss formalisiert und die MFDL entwickelt, eine domänenspezifische Sprache.

In Kapitel 5 wurde das dezentral organisierte Kommunikationsframework SOLA vorgestellt, eine Peer-to-Peer-basierte Plattform. Dieses Framework besteht aus drei funktionalen Komponenten: einer nullbalancierten, vollständigen Baumstruktur na-

mens MINHTON, einem Peer-Discovery-Algorithmus für das Auffinden von spezifischen Peers oder Informationen über diese, und einer Gruppenkommunikation zur Verteilung von Status- und Informationsänderungen (Broadcast). Des Weiteren verwendet SOLA die inhärent bestehende Kommunikationsfähigkeit der Teilnehmer, sodass kein zusätzlicher Server zur Kommunikation benötigt wird, und tritt nach außen hin als eine kohärente Plattform auf. Bei dem Betreten oder Verlassen eines Teilnehmers in MINHTON sind nur eine geringe Anzahl an Knoten involviert, so dass die Kosten in Bezug auf die gesendeten und empfangenen Nachrichten logarithmisch wachsen. Der Peer-Discovery-Algorithmus in SOLA bietet den Teilnehmern die Möglichkeit, gezielt nach Informationen zu suchen. Dabei können diese Informationen z. B. einen oder mehrere Teilnehmer darstellen oder als Einstiegspunkt zu einem Topic-Baum genutzt werden. Zur Erschaffung eines globalen Wissens ist eine Aggregation lokaler Informationen bestimmter Knoten notwendig. Die Anzahl der Knoten, welche in der lokalen Sicht abgedeckt sind, ist nach oben beschränkt. Das Verfahren skaliert automatisch mit der Anzahl der Teilnehmer und Informationen können in $O(1)$ in die dafür vorgesehenen Knoten eingetragen, aktualisiert oder gelöscht werden. Für die Verteilung von Informationsänderungen innerhalb einer Gruppe werden dedizierte Instanzen von MINHTON kreiert, die sogenannten Topic-Bäume. Für jede Komponente wurde sowohl eine analytische Betrachtung als auch eine simulative Evaluation durchgeführt.

Die Evaluation der Anwendbarkeit von SOLA erfolgte in Kapitel 6 mithilfe zweier disjunkter Anwendungsfälle. Der erste Anwendungsfall umfasste die dezentrale Verhandlung der Pfadplanung für ein Sortiersystem mittels Fahrerloser Transportfahrzeuge. Jede Abholstation agierte dabei als eine eigenständige Entität und verhandelt mit den verbleibenden Abholstationen über eine Belegung der Schnittpunkte, sodass ein Fahrerloses Transportfahrzeug eine Last- oder Leerfahrt durchführen kann. Trotz der dezentralen Organisation der Anwendung und der Kommunikation wurden Ergebnisse erreicht, welche sich zentralen Ansätzen annähern. Der zweite Anwendungsfall entspricht dem des selbstorganisierenden Materialflusses. Hierfür existieren logische Materialfluss-Agenten, welche mit den logistischen Betriebsmitteln, den logischen Agenten der Fahrerlosen Transportfahrzeugen, über die Ausführung von Transportaufgaben verhandeln. Dabei sind alle logischen Agenten eigenständig und handeln egoistisch, indem sie versuchen, das für sich beste Ergebnis zu erzielen. Für die logischen Materialfluss-Agenten war somit das Ziel, die Durchlaufzeit der Transporte zu reduzieren, während die logischen Fahrerlosen-Transportfahrzeug-Agenten hingegen versuchten die Anzahl an Transportaufträgen zu maximieren. Zusätzlich wurde die Anzahl an benötigten Fahrerlosen Transportfahrzeugen in Abhängigkeit von der Auftragsperiode T bestimmt, um einen Materialfluss sicherzustellen. Mit den beiden Anwendungsfällen wurde gezeigt, dass sich dezentral organisierte verteilte Applikationen in einem cyberphysischen Produktionssystem unter Verwendung einer serverlosen Kommunikation realisieren lassen.

Die Realisierung und Erprobung hat gezeigt, dass die hier entwickelten Konzepte und Verfahren auf verschiedene Anwendungsfälle angewendet werden können. Auch das dezentral organisierte Kommunikationframework SOLA ist folglich da-

zu geeignet, einen selbstorganisierenden Materialfluss umzusetzen, was zu einer Selbstorganisation in der Intralogistik beiträgt. Dennoch bestehen zahlreiche Anknüpfungspunkte für zukünftige Forschungsarbeiten, allerdings sei an dieser Stelle darauf hingewiesen, dass nur auf einige nachgehend eingegangen wird:

Aus Sicht der Applikation sind die Algorithmen der dezentralen Verhandlungsstrategien bereits Gegenstand der Technik und weiterer Forschung, mit dem Ziel bessere oder schnellere Ergebnisse zu liefern. Häufig werden allerdings die kommunikationstechnischen Auswirkungen vernachlässigt, weil davon ausgegangen wird, dass die Kommunikation aufgrund von technologischen Weiterentwicklungen gegeben ist. Hier sollte beachtet werden, dass das Medium Luft bspw. nur einmal vorhanden ist und daher sollten die Verfahren auch die Auswirkungen auf die Kommunikation bei der Interaktion berücksichtigen.

Aus Sicht der Validierung von SOLA sind neben den Randbedingungen und Einschränkungen aufgrund der Simulation (vgl. Abschn. 6.1.1) auch die Reduktion der Kosten einzelner Funktionsblöcke mögliche Anknüpfungspunkte. Bei der Peer Discovery sind die Kosten für eine Aggregation lokaler Informationen zur Erstellung einer globalen Sicht hoch im Vergleich zu den Kosten für das Eintragen, Aktualisieren oder Löschen von Informationen, Bei der Verteilung der Informationsänderungen in MINHTON könnte die Anzahl der Topic-Bäume reduziert werden, indem die Positionen der Knoten so gewählt werden, dass die Routingtabellen mehr Überschneidung erhalten. In Abschnitt 6.2.1 wurde der Einfluss der Position auf die Ausbreitungsgeschwindigkeit gezeigt. Hier wäre die Optimierung der Positionen zur Laufzeit denkbar, sodass maximal wenige Medienwechsel erfolgen und die mobilen Teilnehmer Nachrichten nur noch empfangen und nicht weiterleiten müssen.

Abkürzungsverzeichnis

Alle im Rahmen der vorliegenden Dissertationsschrift verwendeten Abkürzungen, die nicht zum allgemeinen Sprachgebrauch gehören, werden im Folgenden aufgelistet. Dabei handelt es sich vor allem um Abkürzungen von längeren Fachbegriffen meist mit einem technischen Hintergrund.

0MQ	ZeroMQ
AMQP	Advanced Message Queuing Protocol
AMR	Autonomer mobiler Roboter
AO	ActionOrder
ARP	Address Resolution Protocol
AST	Abstract Syntax Tree
BATON	BAlanced Tree Overlay Network
BATON*	BAlanced m-ary Tree Overlay Network
CA	Cover Area
CDS	Connected Dominating Set
CNP	Contract Net Protocol
CPPS	Cyberphysisches Produktionssystem
CPS	Cyberphysisches System
DAG	Directed Acyclic Graph
DDS	Data Distribution Service
DHT	Distributed Hash Table
DL	Distributed Ledger
DSL	Domain-specific language
DSN	Dominating Set Node
EBT	Epidemic Broadcast Tree
EE	Execution Engine
ES	Experimentelles Setup
ERP	Enterprise Resource Planning
FMS	Flottenmanagementsystem
FTF	Fahrerloses Transportfahrzeug
FTS	Fahrerloses Transportsystem

GPL	General-Purpose Language
I4.0	Industrie 4.0
IIRA	Industrial Internet Reference Architecture
IoT	Internet of Things
IP	Internet Protocol
KMU	Kleine und mittlere Unternehmen
KVP	Key-Value-Pair
L4MS	Logistics for Manufacturing SMEs
LTL	Lineare Temporale Logik
LoTLan	Logistic TAsk LANguage
LRT	Linke Routingtabelle
M2M	Machine-to-Machine
MF	Materialfluss
MFDL	Material Flow Description Language
MINHTON	Minimal Height Tree Overlay Network
MO	MoveOrder
MQTT	Message Queuing Telemetry Transport
nBATON*	Null-Balanced m-ARY Tree Overlay Network
ns-3	Network simulator 3
OCP	Order-Controlled Production
OPC UA	Open Platform Communications Unified Architecture
OPIL	Open Platform for Innovations in Logistics
P2P	Peer-to-Peer
PD	Peer Discovery
PFDL	Production Flow Description Language
PN	Petri-Netz
QoS	Quality of Service
RAMI 4.0	Referenzarchitekturmodell Industrie 4.0
REST	Representational State Transfer
ROS	Robot Operating System
ROS2	Robot Operating System 2
RT	Routingtabelle
RRT	Rechte Routingtabelle
RTC	Routing Table Children
SOA	Service-oriented Architecture
SPOF	Single Point of Failure
STN	Simple Temporal Network
TA	Transportauftrag
TO	TransportOrder
TOS	TransportOrderStep
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Auflistungsverzeichnis

Literaturverzeichnis

ABB+11. Aghassi, Susanne; Bauhoff, Fabian; Brecher, Christian; Fuchs, Sascha; Jeschke, Sabina; Jooß, Claudia; Kozielski, Stefan; Orilski, Simon; Richert, Anja; Roderburg, Andreas; Schiffer, Michael; Schubert, Johannes; Schuh, Günther; Stiller, Sebastian; Welter, Florian; Arnoscht, Jens; Karmann, Oliver; Tönissen, Stefan: Integrative Produktionstechnik für Hochlohnländer. In: Brecher, Christian (Hrsg.): *Integrative Produktionstechnik für Hochlohnländer*. Berlin, Heidelberg: Springer, 2011 (VDI-Buch). – ISBN 978–3–642–20693–1, S. 17–81

AGD+06. Anceaume, E.; Gradinariu, M.; Datta, A.K.; Simon, G.; Virgillito, A.: A Semantic Overlay for Self- Peer-to-Peer Publish/Subscribe. In: *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. Lisboa, Portugal, Juli 2006. – ISSN 1063–6927, S. 22–22

AGL+15. Autili, Marco; Grunske, Lars; Lumpe, Markus; Pelliccione, Patrizio; Tang, Antony: Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar. In: *IEEE Transactions on Software Engineering* 41 (2015), Juli, Nr. 7, S. 620–638. – ISSN 1939–3520

AHK+18. Alawe, I.; Hadjadj-Aoul, Y.; Ksentini, A.; Bertin, P.; Darche, D.: On the Scalability of 5G Core Network: The AMF Case. In: *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2018. – ISSN 2331–9860, S. 1–6

AIM10. Atzori, Luigi; Iera, Antonio; Morabito, Giacomo: The Internet of Things: A Survey. In: *Computer Networks* 54 (2010), Oktober, Nr. 15, S. 2787–2805. – ISSN 1389–1286

ALG+09. Ashley-Rollman, Michael P.; Lee, Peter; Goldstein, Seth C.; Pillai, Padmanabhan; Campbell, Jason D.: A Language for Large Ensembles of Independently Executing Nodes. In: Hill, Patricia M. (Hrsg.); Warren, David S. (Hrsg.): *Logic Programming*. Berlin, Heidelberg: Springer, 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–02846–5, S. 265–280

All83. Allen, James F.: Maintaining Knowledge about Temporal Intervals. In: *Communications of the ACM* 26 (1983), November, Nr. 11, S. 832–843. – ISSN 0001–0782, 1557–7317

AMN+19. Ankele, Ralph; Marksteiner, Stefan; Nahrgang, Kai; Vallant, Heribert: Requirements and Recommendations for IoT/IIoT Models to Automate Security Assurance through Threat Modelling, Security Analysis and Penetration Testing. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. New York, NY, USA: Association for Computing Machinery, August 2019 (ARES '19). – ISBN 978–1–4503–7164–3, S. 1–8

AP07. Ardagna, Danilo; Pernici, Barbara: Adaptive Service Composition in Flexible Processes. In: *IEEE Transactions on Software Engineering* 33 (2007), Juni, Nr. 6, S. 369–384. – ISSN 1939–3520

APP19. Aceto, Giuseppe; Persico, Valerio; Pescapé, Antonio: A Survey on Information and Communication Technologies for Industry 4.0: State-of-the-Art, Taxonomies, Perspectives, and Challenges. In: *IEEE Communications Surveys Tutorials* 21 (Fourthquarter 2019), Nr. 4, S. 3467–3501. – ISSN 1553–877X

APR+18. Antão, Liliana; Pinto, Rui; Reis, João; Gonçalves, Gil: Requirements for Testing and Validating the Industrial Internet of Things. In: *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018, S. 110–115

Arn06. Arnold, Dieter (Hrsg.): *Intralogistik: Potentiale, Perspektiven, Prognosen*. Berlin Heidelberg: Springer-Verlag, 2006 (VDI-Buch). – ISBN 978–3–540–29657–7

AS04. Androutsellis-Theotokis, Stephanos; Spinellis, Diomidis: A Survey of Peer-to-Peer Content Distribution Technologies. In: *ACM Computing Surveys* 36 (2004), Dezember, Nr. 4, S. 335–371. – ISSN 0360–0300, 1557–7341

ASP+15. Anis, Anas; Schäfer, Wilhelm; Pines, Andrey; Niggemann, Oliver: CP3L: A Cyber-Physical Production Planning Language. In: *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, 2015. – ISSN 1946–0759, S. 1–4

ATA+21. Aliev, Khurshid; Traini, Emiliano; Asranov, Mansur; Awouda, Ahmed; Chiabert, Paolo: Prediction and Estimation Model of Energy Demand of the AMR with Cobot for the Designed Path in Automated Logistics Systems. In: *Procedia CIRP* 99 (2021), Januar, S. 116–121. – ISSN 2212–8271

Bas17. Bashir, Imran: *Mastering Blockchain*. Packt Publishing Ltd, 2017. – ISBN 978–1–78712–929–0

BCM+99. Banavar, G.; Chandra, T.; Mukherjee, B.; Nagarajarao, J.; Strom, R.E.; Sturman, D.C.: An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In: *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003)*, 1999. – ISSN 1063–6927, S. 262–272

Ben86. Bentley, Jon: Programming Pearls: Little Languages. In: *Communications of the ACM* 29 (1986), August, Nr. 8, S. 711–721. – ISSN 0001–0782

BFK+14. Brettel, Malte; Friederichsen, Niklas; Keller, Michael; Rosenberg, Marius: How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective. In: *International Journal of Information and Communication Engineering* 8 (2014), Nr. 1, S. 37–44. – ISSN 1307–6892

BFO+98. Bertocco, M.; Ferraris, F.; Offelli, C.; Parvis, M.: A Client-Server Architecture for Distributed Measurement Systems. In: *IEEE Transactions on Instrumentation and Measurement* 47 (1998), Oktober, Nr. 5, S. 1143–1148. – ISSN 1557–9662

BG88. Bond, Alan H.; Gasser, Les: *Readings in Distributed Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988. – ISBN 978–1–4832–1444–3

BHH21. Berger, Stephan; Häckel, Björn; Häfner, Lukas: Organizing Self-Organizing Systems: A Terminology, Taxonomy, and Reference Model for Entities in Cyber-Physical Production Systems. In: *Information Systems Frontiers* 23 (2021), April, Nr. 2, S. 391–414. – ISSN 1572–9419

BKR+16. Bauernhansl, Thomas; Krüger, Jörg; Reinhart, Gunther; Schuh, Günther: WGP-Standpunkt Industrie 4.0 / Wissenschaftliche Gesellschaft für Produktionstechnik WGP e. V. Darmstadt, Juni 2016. – Standpunktpapier. – 51 S. – ISBN 978–0–13–449416–6

BLS+17. Blesing, Christian; Luensch, Dennis; Stenzel, Jonas; Korth, Benjamin: Concept of a Multi-agent Based Decentralized Production System for the Automotive Industry. In: Demazeau, Yves (Hrsg.); Davidsson, Paul (Hrsg.); Bajo, Javier (Hrsg.); Vale, Zita (Hrsg.): *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection*. Cham: Springer International Publishing, 2017 (Lecture Notes in Computer Science). – ISBN 978–3–319–59930–4, S. 19–30

BMT+20. Bodkhe, Umesh; Mehta, Dhyey; Tanwar, Sudeep; Bhattacharya, Pronaya; Singh, Pradeep K.; Hong, Wei-Chiang: A Survey on Decentralized Consensus Mechanisms for

Cyber Physical Systems. In: *IEEE Access* 8 (2020), S. 54371–54401. – ISSN 2169–3536

BNM11. Buniyamin; Ngah, W.; Mohamad, Z.: A Simple Local Path Planning Algorithm for Autonomous Mobile Robots. In: *INTERNATIONAL JOURNAL OF SYSTEMS APPLICATIONS, ENGINEERING & DEVELOPMENT* 5 (2011), Nr. 2

BPS+08. Bray, Tim; Paoli, Jean; Sperberg-McQueen, C.N; Maler, Eve; Yergeau, François: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. https://www.w3.org/TR/2008/REC-xml-20081126/, November 2008. – letzter Zugriff: 10.07.2022

BQV09. Baldoni, Roberto; Querzoni, Leonardo; Virgillito, Antonino: Distributed Event Routing in Publish/Subscribe Communication Systems: A Survey. (2009), Februar, S. 27

BSS+07. Bauland, Michael; Schneider, Thomas; Schnoor, Henning; Schnoor, Ilka; Vollmer, Heribert: The Complexity of Generalized Satisfiability for Linear Temporal Logic. In: Seidl, Helmut (Hrsg.): *Foundations of Software Science and Computational Structures*. Berlin, Heidelberg: Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–71389–0, S. 48–62

BT12. Beckmann, Kai; Thoss, Marcus: A Wireless Sensor Network Protocol for the OMG Data Distribution Service. In: *Proceedings of the 10th International Workshop on Intelligent Solutions in Embedded Systems*, 2012, S. 45–50

BtV14. Bauernhansl, Thomas; ten Hompel, Michael; Vogel-Heuser, Birgit: *Industrie 4.0 in Produktion, Automatisierung Und Logistik: Anwendung, Technologien, Migration*. Wiesbaden: Springer Vieweg, 2014 (SpringerLink). – ISBN 978–3–658–04681–1

Bun19. Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen: WLAN Service Heft / Bundesnetzagentur. Bonn, März 2019. – Forschungsbericht. – 4 S.

Bun20. Bundesministerium für Wirtschaft und Energie: Leitfaden 5G-Campusnetze-Orientierungshilfe für kleine und mittelständische Unternehmen / Bundesministerium für Wirtschaft und Energie (BMWi). Berlin, April 2020. – Forschungsbericht. – 52 S.

Bus21. Busch, Emanuel: *Digitalisierung im Kontext der Industrie 4.0: Eine quantitative Erhebung*. Dortmund, TU Dortmund, Masterarbeit, November 2021

BVF+16. Backman, Jere; Väre, Janne; Främling, Kary; Madhikermi, Manik; Nykänen, Ossi: IoT-based Interoperability Framework for Asset and Fleet Management. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, S. 1–4

BYG03. Beverly Yang, B.; Garcia-Molina, H.: Designing a Super-Peer Network. In: *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, 2003, S. 49–60

Cas95. Castelfranchi, Cristiano: Guarantees for Autonomy in Cognitive Agent Architecture. In: *Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents*. Berlin, Heidelberg: Springer-Verlag, März 1995 (ECAI-94). – ISBN 978–3–540–58855–9, S. 56–70

CCK+18. Chowdhury, Mohammad Jabed M.; Colman, Alan; Kabir, Muhammad A.; Han, Jun; Sarda, Paul: Blockchain Versus Database: A Critical Analysis. In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018. – ISSN 2324–9013, S. 1348–1353

CG00. Cho, Junghoo; Garcia-Molina, Hector: Synchronizing a Database to Improve Freshness. In: *ACM SIGMOD Record* 29 (2000), Mai, Nr. 2, S. 117–128. – ISSN 0163–5808

CH13. Czirkos, Zoltán; Hosszú, Gábor: Solution for the Broadcasting in the Kademlia Peer-to-Peer Overlay. In: *Computer Networks* 57 (2013), Juni, Nr. 8, S. 1853–1862. – ISSN 1389–1286

Cha12. Cha, Sung-Hyuk: On Integer Sequences Derived from Balanced K-Ary Trees. In: *Proceedings of the 6th WSEAS International Conference on Computer Engineering*

and Applications, and Proceedings of the 2012 American Conference on Applied Mathematics. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), Januar 2012 (AMERICAN-MATH'12/CEA'12). – ISBN 978–1–61804–064–0, S. 377–381

CLZ00. Cabri, G.; Leonardi, L.; Zambonelli, F.: Mobile-Agent Coordination Models for Internet Applications. In: *Computer* 33 (Feb./2000), Nr. 2, S. 82–89. – ISSN 00189162

CMF+18. Chen, Hongzhi; Mi, De; Fuentes, Manuel; Vargas, David; Garro, Eduardo; Carcel, Jose L.; Mouhouche, Belkacem; Xiao, Pei; Tafazolli, Rahim: Pioneering Studies on LTE eMBMS: Towards 5G Point - to-Multipoint Transmissions. In: *2018 IEEE 10th Sensor Array and Multichannel Signal Processing Workshop (SAM)*, 2018. – ISSN 2151–870X, S. 1–4

CMT+07. Chockler, Gregory; Melamed, Roie; Tock, Yoav; Vitenberg, Roman: Constructing Scalable Overlays for Pub-Sub with Many Topics. In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing - PODC '07*. Portland, Oregon, USA: ACM Press, 2007. – ISBN 978–1–59593–616–5, S. 109

Coh03. Cohen, Bram: Incentives Build Robustness in BitTorrent. 2003. – Whitepaper. – 5 S.

Cro06. Crockford, D.: *The Application/Json Media Type for JavaScript Object Notation (JSON)*. https://www.ietf.org/rfc/rfc4627.txt, Juli 2006. – letzter Zugriff: 07.10.2022

CSRL+19. Cruz Salazar, Luis A.; Ryashentseva, Daria; Lüder, Arndt; Vogel-Heuser, Birgit: Cyber-Physical Production Systems Architecture Based on Multi-Agent's Design Pattern—Comparison of Selected Approaches Mapping Four Agent Patterns. In: *The International Journal of Advanced Manufacturing Technology* 105 (2019), Dezember, Nr. 9, S. 4005–4034. – ISSN 0268–3768, 1433–3015

CSW+01. Clarke, Ian; Sandberg, Oskar; Wiley, Brandon; Hong, Theodore W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System. In: Federrath, Hannes (Hrsg.): *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Berlin, Heidelberg: Springer, 2001 (Lecture Notes in Computer Science). – ISBN 978–3–540–44702–3, S. 46–66

DCJ+19. Dizdarević, Jasenka; Carpio, Francisco; Jukan, Admela; Masip-Bruin, Xavi: A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. In: *ACM Computing Surveys* 51 (2019), Februar, Nr. 6, S. 1–29. – ISSN 0360–0300, 1557–7341

Dec98. Decker, Stefan: On Domain-Specific Declarative Knowledge Representation and Database Languages. In: *Proc. of the 5th Knowledge Representation meets Databases Workshop (KRDB98)* (1998)

DEH+22. Detzner, Peter; Ebner, Andreas; Hörstrup, Maximilian; Kerner, Sören: PFDL: A Production Flow Description Language for an Order-Controlled Production. In: *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*, 2022, S. 1099–1106

DFS+13. Daubert, Jörg; Fischer, Mathias; Schiffner, Stefan; Mühlhäuser, Max: Distributed and Anonymous Publish-Subscribe. In: Lopez, Javier (Hrsg.); Huang, Xinyi (Hrsg.); Sandhu, Ravi (Hrsg.): *Network and System Security*. Berlin, Heidelberg: Springer, 2013 (Lecture Notes in Computer Science). – ISBN 978–3–642–38631–2, S. 685–691

DGB21. Detzner, Peter; Gödeke, Jana; Bondorf, Steffen: Low-Cost Search in Tree-Structured P2P Overlays: The Null-Balance Benefit. In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021. – ISSN 0742–1303, S. 613–620

DGB22. Detzner, Peter; Gödeke, Jana; Bondorf, Steffen: Peer Discovery in Tree-Structured P2P Overlay Networks by Means of Connected Dominating Sets. In: *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, 2022. – ISSN 0742–1303, S. 447–454

dGM+13. de Lemos, Rogerio; Giese, Holger; Müller, Hausi A.; Shaw, Mary; Andersson, Jesper; Litoiu, Marin; Schmerl, Bradley; Tamura, Gabriel; Villegas, Norha M.; Vogel,

Thomas; Weyns, Danny; Baresi, Luciano; Becker, Basil; Bencomo, Nelly; Brun, Yuriy; Cukic, Bojan; Desmarais, Ron; Dustdar, Schahram; Engels, Gregor; Geihs, Kurt; Göschka, Karl M.; Gorla, Alessandra; Grassi, Vincenzo; Inverardi, Paola; Karsai, Gabor; Kramer, Jeff; Lopes, Antónia; Magee, Jeff; Malek, Sam; Mankovskii, Serge; Mirandola, Raffaela; Mylopoulos, John; Nierstrasz, Oscar; Pezzè, Mauro; Prehofer, Christian; Schäfer, Wilhelm; Schlichting, Rick; Smith, Dennis B.; Sousa, Joao P.; Tahvildari, Ladan; Wong, Kenny; Wuttke, Jochen: Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In: de Lemos, Rogerio (Hrsg.); Giese, Holger (Hrsg.); Müller, Hausi A. (Hrsg.); Shaw, Mary (Hrsg.): *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Berlin, Heidelberg: Springer, 2013 (Lecture Notes in Computer Science). – ISBN 978–3–642–35813–5, S. 1–32

Dij59. Dijkstra, E. W.: A Note on Two Problems in Connexion with Graphs. In: *Numerische Mathematik* 1 (1959), Dezember, Nr. 1, S. 269–271. – ISSN 0945–3245

DKJ19. Detzner, P.; Kirks, T.; Jost, J.: A Novel Task Language for Natural Interaction in Human-Robot Systems for Warehouse Logistics. In: *2019 14th International Conference on Computer Science Education (ICCSE)*, 2019. – ISSN 2473–9464, S. 725–730

DKW+11. Dantu, Karthik; Kate, Bryan; Waterman, Jason; Bailis, Peter; Welsh, Matt: Programming Micro-Aerial Vehicle Swarms with Karma. In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11*. Seattle, Washington: ACM Press, 2011. – ISBN 978–1–4503–0718–5, S. 121

DMK+16. Draganjac, I.; Miklić, D.; Kovačić, Z.; Vasiljević, G.; Bogdan, S.: Decentralized Control of Multi-AGV Systems in Autonomous Warehousing Applications. In: *IEEE Transactions on Automation Science and Engineering* 13 (2016), Oktober, Nr. 4, S. 1433–1447. – ISSN 1558–3783

DMP91. Dechter, Rina; Meiri, Itay; Pearl, Judea: Temporal Constraint Networks. In: *Artificial Intelligence* 49 (1991), Mai, Nr. 1, S. 61–95. – ISSN 0004–3702

Dor15. Dorothy, Stanley: *IEEE 802.11 Multicast Capabilities*. Januar 2015

DPF+19. Detzner, P.; Pose, T.; Fumagalli, L.; Matteucci, M.: Towards a Plug and Play Architecture for a Materialflow Handling System. In: *2019 IEEE Conference on Open Systems (ICOS)*, 2019. – ISSN 2473–3660, S. 28–33

DPS+11. Dillon, Tharam; Potdar, Vidyasagar; Singh, Jaipal; Talevski, Alex: Cyber-Physical Systems: Providing Quality of Service (QoS) in a Heterogeneous Systems-of-Systems Environment. In: *5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011)*, 2011. – ISSN 2150–4946, S. 330–335

dRB14. del Val, E.; Rebollo, M.; Botti, V.: Enhancing Decentralized Service Discovery in Open Service-Oriented Multi-Agent Systems. In: *Autonomous Agents and Multi-Agent Systems* 28 (2014), Januar, Nr. 1, S. 1–30. – ISSN 1387–2532, 1573–7454

DRVD20. De Ryck, M.; Versteyhe, M.; Debrouwere, F.: Automated Guided Vehicle Systems, State-of-the-Art Control Algorithms and Techniques. In: *Journal of Manufacturing Systems* 54 (2020), Januar, S. 152–173. – ISSN 0278–6125

DS20. Detzner, Peter; Salhofer, Peter: Analysing FIWAREs Platform - Potential Improvements. In: *Proceedings of the 53rd Hawaii International Conference on System Sciences*. Hawaii, Januar 2020. – ISBN 978–0–9981331–3–3

Duf90. Duffie, Neil A.: Synthesis of Heterarchical Manufacturing Systems. In: *Computers in Industry* 14 (1990), Mai, Nr. 1, S. 167–174. – ISSN 0166–3615

DZD+03. Dabek, Frank; Zhao, Ben; Druschel, Peter; Kubiatowicz, John; Stoica, Ion: Towards a Common API for Structured Peer-to-Peer Overlays. In: Goos, Gerhard (Hrsg.); Hartmanis, Juris (Hrsg.); van Leeuwen, Jan (Hrsg.); Kaashoek, M. F. (Hrsg.); Stoica, Ion (Hrsg.): *Peer-to-Peer Systems II* Bd. 2735. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. – ISBN 978–3–540–40724–9 978–3–540–45172–3, S. 33–44

EAB+03. El-Ansary, Sameh; Alima, Luc O.; Brand, Per; Haridi, Seif: Efficient Broadcast in Structured P2P Networks. In: Kaashoek, M. F. (Hrsg.); Stoica, Ion (Hrsg.): *Peer-to-Peer Systems II*. Berlin, Heidelberg: Springer, 2003 (Lecture Notes in Computer Science). – ISBN 978–3–540–45172–3, S. 304–314

ECP+05. Eng Keong Lua; Crowcroft, J.; Pias, M.; Sharma, R.; Lim, S.: A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. In: *IEEE Communications Surveys & Tutorials* 7 (2005), Nr. 2, S. 72–93. – ISSN 1553–877X

EFG+03. Eugster, Patrick T.; Felber, Pascal A.; Guerraoui, Rachid; Kermarrec, Anne-Marie: The Many Faces of Publish/Subscribe. In: *ACM Computing Surveys* 35 (2003), Juni, Nr. 2, S. 114–131. – ISSN 0360–0300, 1557–7341

EM13. Esfahani, Naeem; Malek, Sam: Uncertainty in Self-Adaptive Software Systems. In: de Lemos, Rogerio (Hrsg.); Giese, Holger (Hrsg.); Müller, Hausi A. (Hrsg.); Shaw, Mary (Hrsg.): *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Berlin, Heidelberg: Springer, 2013 (Lecture Notes in Computer Science). – ISBN 978–3–642–35813–5, S. 214–238

Erb17. Erboz, Gizem: How to Define Industry 4.0: The Main Pillars Of Industry 4.0. In: *Managerial trends in the development of enterprises in globalization* (2017), November, S. 8

ETS20. ETSI: *Context Information Management (CIM); NGSI-LD Primer*. März 2020

Eur16a. European Comission: *Bringing FIWARE to the NEXT Step — FI-NEXT Project — H2020 — Grant Agreement ID 732851*. https://cordis.europa.eu/project/id/732851, 2016. – letzter Zugriff: 06.07.2021

Eur16b. European Comission: *FI-GLOBAL: Building and Supporting a Global Open Community of FIWARE Innovators and Users — FI-GLOBAL Project — H2020 — CORDIS — European Commission*. https://cordis.europa.eu/project/id/732056, November 2016. – letzter Zugriff: 06.07.2021

Eur17a. European Comission: *FI-WARE: Future Internet Core Platform — FI-WARE Project — FP7 — CORDIS — European Commission*. https://cordis.europa.eu/project/id/285248, April 2017. – letzter Zugriff: 06.07.2021

Eur17b. European Comission: France: Industrie Du Futur. 2017. – Forschungsbericht

Eur17c. European Telecommunications Standards Institute: 5G - Study on Scenarios and Requirements for Next Generation Access Technologies. 2017 (TR 38.913 14.2.0 Release 14). – Forschungsbericht. – 41 S.

Eur19. European Telecommunications Standards Institute: Context Information Management - NGSI-LD API. 2019 (1.1.1). – Forschungsbericht

Eur20. European Comission: User Guide to the SME Definition. (2020), S. 60. – ISSN 978–92–79–69909–2

FBE+16. Fathy, Y.; Barnaghi, P.; Enshaeifar, S.; Tafazolli, R.: A Distributed In-Network Indexing Mechanism for the Internet of Things. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016, S. 585–590

FdS+21. Fragapane, Giuseppe; de Koster, René; Sgarbossa, Fabio; Strandhagen, Jan O.: Planning and Control of Autonomous Mobile Robots for Intralogistics: Literature Review and Research Agenda. In: *European Journal of Operational Research* (2021), Januar. – ISSN 0377–2217

Fel18. Feldhorst, Sascha: *Automatische Aktivitäts- und Kontexterkennung zur Analyse des Kommissionierprozesses*. Dortmund, Verlax Praxiswissen, Diss., Dezember 2018

Fie00. Fielding, Roy T.: *REST: Architectural Styles and the Design of Network-Based Software Architectures*, University of California, Irvine, Diss., 2000

FIW. FIWARE Foundation: *The FIWARE Foundation*

FO82. Flajolet, Philippe; Odlyzko, Andrew: The Average Height of Binary Trees and Other Simple Trees. In: *Journal of Computer and System Sciences* 25 (1982), Oktober, Nr. 2, S. 171–213. – ISSN 0022–0000

FP11. Fowler, Martin; Parsons, Rebecca: *Domain-Specific Languages*. Addison-Wesley, 2011. – ISBN 978–0–321–71294–3

Fra01. Frankel, James: *The Gnutella Protocol Specification v0.4*. Juni 2001

GCF+19. González, Isaías; Calderón, Antonio J.; Figueiredo, João; M. C. Sousa, João: A Literature Survey on Open Platform Communications (OPC) Applied to Advanced Industrial Environments. In: *Electronics* 8 (2019), Mai, Nr. 5, S. 510

GD13. Government Office for Science; Department for Business, Innovation & Skills: *Future of Manufacturing*. https://www.gov.uk/government/collections/future-of-manufacturing, Oktober 2013 letzter Zugriff: 30.06.2021

GFC+20. Garro, E.; Fuentes, M.; Carcel, J. L.; Chen, H.; Mi, D.; Tesema, F.; Gimenez, J. J.; Gomez-Barquero, D.: 5G Mixed Mode: NR Multicast-Broadcast Services. In: *IEEE Transactions on Broadcasting* 66 (2020), Juni, Nr. 2, S. 390–403. – ISSN 1557–9611

GGB+18. Gupta, Varun; Gutterman, Craig; Bejerano, Yigal; Zussman, Gil: Experimental Evaluation of Large Scale WiFi Multicast Rate Control. In: *IEEE Transactions on Wireless Communications* 17 (2018), April, Nr. 4, S. 2319–2332. – ISSN 1558–2248

GHJ+94. Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John M.: *Design Patterns: Elements of Reusable Object-Oriented Software*. First. Addison-Wesley Professional, 1994. – ISBN 0–201–63361–2

GK94. Genesereth, Michael R.; Ketchpel, Steven P.: Software Agents. In: *Communications of the ACM* 37 (1994), Juli, Nr. 7, S. 48–53. – ISSN 0001–0782

GK98. Guha, S.; Khuller, S.: Approximation Algorithms for Connected Dominating Sets. In: *Algorithmica* 20 (1998), April, Nr. 4, S. 374–387. – ISSN 1432–0541

GM02. Gerkey, B.P.; Mataric, M.J.: Sold!: Auction Methods for Multirobot Coordination. In: *IEEE Transactions on Robotics and Automation* 18 (2002), Oktober, Nr. 5, S. 758–768. – ISSN 1042–296X

GM04. Gerkey, Brian P.; Matarić, Maja J.: A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. In: *The International Journal of Robotics Research* 23 (2004), September, Nr. 9, S. 939–954. – ISSN 0278–3649, 1741–3176

Göd23. Gödeke, Jana: *Scalable Decentralized Task Allocation for Cyber-Physical Production Systems*. Dortmund, TU Dortmund, Masterarbeit, Januar 2023

GPM+19. García, Sergio; Pelliccione, Patrizio; Menghi, Claudio; Berger, Thorsten; Bures, Tomas: High-Level Mission Specification for Multiple Robots. In: *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering - SLE 2019*. Athens, Greece: ACM Press, 2019. – ISBN 978–1–4503–6981–7, S. 127–140

Gra10. Graffi, Kalman: *Monitoring and Management of Peer-to-Peer Systems*. Darmstadt, Technische Universität Darmstadt, Diss., 2010

GSL+14. Gorecky, Dominic; Schmitt, Mathias; Loskyll, Matthias; Zühlke, Detlef: Human-Machine-Interaction in the Industry 4.0 Era. In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. Porto Alegre, Brazil, Juli 2014. – ISBN 978–1–4799–4905–2, S. 289–294

Gt10. Günthner, Willibald (Hrsg.); ten Hompel, Michael (Hrsg.): *Internet Der Dinge in Der Intralogistik*. Three hundred sixtieth. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. – ISBN 978–3–642–04895–1 978–3–642–04896–8

Gud10. Gudehus, Timm: *Logistik: Grundlagen – Strategien – Anwendungen*. Fourth. Berlin: Springer, 2010. – ISBN 978–3–540–89388–2

GVB16. Grefen, P.; Vanderfeesten, I.; Boultadakis, G.: Supporting Hybrid Manufacturing: Bringing Process and Human/Robot Control to the Cloud (Short Paper). In: *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*. Los Alamitos, CA, USA: IEEE Computer Society, Oktober 2016, S. 200–203

Hey99. Heylighen, Francis: The Science Of Self-Organization And Adaptivity. Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems, EOLSS (1999), Mai, S. 26

HH11. Hompel, Michael; Heidenblut, Volker; Hompel, Michael (Hrsg.): *Taschenlexikon Logistik: Abkürzungen, Definitionen und Erläuterungen der wichtigsten Begriffe aus Materialfluss und Logistik*. 3. Berlin Heidelberg: Springer-Verlag, 2011 (VDI-Buch). – ISBN 978–3–642–19944–8

HHH+19. Heidel, Roland; Hoffmeister, Michael; Hankel, Martin; Döbrich, Udo: *The Reference Architecture Model RAMI 4.0 and the Industrie 4.0 Component*. VDE Verlag, 2019. – ISBN 978–3–8007–4990–4 978–3–8007–4992–8

Hin13. Hintjens, Pieter: *ZeroMQ: Messaging for Many Applications*. O'Reilly Media, Inc., 2013. – ISBN 978–1–4493–3406–2

Hir15. Hirsch-Kreinsen, Hartmut: Entwicklungsperspektiven von Produktionsarbeit. In: Botthof, Alfons (Hrsg.); Hartmann, Ernst A. (Hrsg.): *Zukunft der Arbeit in Industrie 4.0*. Berlin, Heidelberg: Springer, 2015. – ISBN 978–3–662–45915–7, S. 89–98

HL18. Heinzemann, Christian; Lange, Ralph: vTSL - A Formally Verifiable DSL for Specifying Robot Tasks. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018. – ISSN 2153–0866, S. 8308–8314

HM10. Herrero-Perez, D.; Martinez-Barbera, H.: Modeling Distributed Transportation Systems Composed of Flexible Automated Guided Vehicles in Flexible Manufacturing Systems. In: *IEEE Transactions on Industrial Informatics* 6 (2010), Mai, Nr. 2, S. 166–180. – ISSN 1941–0050

Hol02. Holzmann, Gerard J.: The Logic of Bugs. In: *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, November 2002 (SIGSOFT '02/FSE-10). – ISBN 978–1–58113–514–5, S. 81–87

Hol03. Holzmann, Gerard: *Spin Model Checker, the: Primer and Reference Manual*. First. Addison-Wesley Professional, 2003. – ISBN 978–0–321–22862–8

Hör22. Hörstrup, Maximilian: *Entwurf und Entwicklung einer domänenspezifischen Sprache zur Beschreibung von Produktionsaufträgen in Cyber-Physischen Produktionssystemen*. Dortmund, TU Dortmund, Bachelorarbeit, 2022

Hos15. Hoste, Lode: *A Declarative Approach for Engineering Multimodal Interaction*. Vrije Universiteit Brussel, 2015. – ISBN 978–94–92312–00–6

HPO16. Hermann, M.; Pentek, T.; Otto, B.: Design Principles for Industrie 4.0 Scenarios. In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016. – ISSN 1530–1605, S. 3928–3937

HRC+18. Hejazi, Hamdan; Rajab, Husam; Cinkler, Tibor; Lengyel, László: Survey of Platforms for Massive IoT. In: *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, 2018, S. 1–8

HSD18. Hompel, Michael; Schmidt, Thorsten; Dregger, Johannes: *Materialflusssysteme: Förder- und Lagertechnik*. 4. Springer Vieweg, 2018 (VDI-Buch). – ISBN 978–3–662–56180–5

HW16. Happ, Daniel; Wolisz, Adam: Limitations of the Pub/Sub Pattern for Cloud Based IoT and Their Implications. In: *2016 Cloudification of the Internet of Things (CIoT)*, 2016, S. 1–6

HWW+16. Holfeld, Bernd; Wieruch, Dennis; Wirth, Thomas; Thiele, Lars; Ashraf, Shehzad A.; Huschke, Jorg; Aktas, Ismet; Ansari, Junaid: Wireless Communication for Factory Automation: An Opportunity for LTE and 5G Systems. In: *IEEE Communications Magazine* 54 (2016), Juni, Nr. 6, S. 36–43. – ISSN 1558–1896

HZW+08. Hu, S. J.; Zhu, X.; Wang, H.; Koren, Y.: Product Variety and Manufacturing Complexity in Assembly Systems and Supply Chains. In: *CIRP Annals* 57 (2008), Januar, Nr. 1, S. 45–48. – ISSN 0007–8506

IEE. IEEE 802.11 Working Group: *IEEE 802.11, The Working Group Setting the Standards for Wireless LANs*. https://www.ieee802.org/11/, . – letzter Zugriff: 02.07.2021

IEE22. IEEE 802.15 WPAN™ Task Group 4: *IEEE 802.15.4*. https://www.ieee802.org/15/pub/TG4.html, Oktober 2022. – letzter Zugriff: 10.07.2021

Ing70. Ingenieure, Verein D.: *VDI 2411 - Begriffe Und Erläuterungen Im Förderwesen*. VDI-Richtlinien, 1970 (2411)

Int81a. Internet Engineering Task Force: *Rfc768 - User Datagram Protocol*. https://datatracker.ietf.org/doc/html/rfc768, August 1981. – letzter Zugriff: 02.07.2021

Int81b. Internet Engineering Task Force: *Rfc793 - Transmission Control Protocol*. https://datatracker.ietf.org/doc/html/rfc793, September 1981. – letzter Zugriff: 02.07.2021

Int05. Internet Engineering Task Force: *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. https://datatracker.ietf.org/doc/html/rfc4180, Oktober 2005. – letzter Zugriff: 01.11.2021

Int20. Internet Engineering Task Force: *QUIC*. https://datatracker.ietf.org/wg/quic/about/, März 2020. – letzter Zugriff: 02.07.2021

Ish16. Ishiwatari, Yuji: Regarding Japan's Declaration to Be the World's Most Advanced IT Nation. (2016), S. 4

Jos21. Jost, Jana: *Beitrag Zur Maschinellen Integration in Sozio-Technische Systeme*. Dortmund: TU Dortmund, 2021. – ISBN 978–3–86975–169–6

JOT+06. Jagadish, H. V.; Ooi, Beng C.; Tan, Kian-Lee; Vu, Quang H.; Zhang, Rong: Speeding up Search in Peer-to-Peer Networks with a Multi-Way Tree Structure. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data - SIGMOD '06*. Chicago, IL, USA: ACM Press, 2006. – ISBN 978–1–59593–434–5, S. 1

JOV05. Jagadish, H. V.; Ooi, Beng C.; Vu, Quang H.: BATON: A Balanced Tree Structure for Peer-to-Peer Networks. In: *Proceedings of the 31st International Conference on Very Large Data Bases*. Trondheim, Norway: VLDB Endowment, August 2005 (VLDB '05). – ISBN 978–1–59593–154–2, S. 661–672

JOV+06. Jagadish, H.V.; Ooi, Beng C.; Vu, Quang H.; Zhang, Rong; Zhou, Aoying: VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes. In: *22nd International Conference on Data Engineering (ICDE'06)*, 2006. – ISSN 2375–026X, S. 34–34

KAK14. Kataoka, Kotaro; Agarwal, Nitin; Kamath, Aditya V.: Scaling a Broadcast Domain of Ethernet: Extensible Transparent Filter Using SDN. In: *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, 2014. – ISSN 1095–2055, S. 1–8

KAL+21. Krämer, Larissa; Ahlbäumer, Rico; Leveling, Jens; Detzner, Peter; Brehler, Marius; ten Hompel, Michael: Towards a Concept for Blockchain-based Cyber-Physical Production Systems. In: *Volume 2021* (2021), S. Issue 17

KÇP17. Köseoğlu, Murat; Çelik, Orkan M.; Pektaş, Ömer: Design of an Autonomous Mobile Robot Based on ROS. In: *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*. Malatya, Turkey: IEEE, September 2017, S. 1–5

KD20. Kassab, Wafa'a; Darabkh, Khalid A.: A–Z Survey of Internet of Things: Architectures, Protocols, Applications, Recent Advances, Future Directions and Recommendations. In: *Journal of Network and Computer Applications* 163 (2020), August, S. 102663. – ISSN 1084–8045

KFP07. Kress-Gazit, H.; Fainekos, G. E.; Pappas, G. J.: From Structured English to Robot Motion. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007. – ISSN 2153–0866, S. 2717–2722

KGR20. Koziolek, Heiko; Grüner, Sten; Rückert, Julius: A Comparison of MQTT Brokers for Distributed IoT Edge Computing. In: Jansen, Anton (Hrsg.); Malavolta, Ivano (Hrsg.); Muccini, Henry (Hrsg.); Ozkaya, Ipek (Hrsg.); Zimmermann, Olaf (Hrsg.): *Software Architecture*. Cham: Springer International Publishing, 2020 (Lecture Notes in Computer Science). – ISBN 978–3–030–58923–3, S. 352–368

KHE15. Khamis, Alaa; Hussein, Ahmed; Elmogy, Ahmed: Multi-Robot Task Allocation: A Review of the State-of-the-Art. In: Koubâa, Anis (Hrsg.); Martínez-de Dios, J.Ramiro (Hrsg.): *Cooperative Robots and Sensor Networks 2015*. Cham: Springer International Publishing, 2015 (Studies in Computational Intelligence). – ISBN 978–3–319–18299–5, S. 31–51

KHM18. Kolberg, Dennis; Herman, Jasko; Mohr, Florian: SmartFactory Systemarchitektur für Industrie 4.0-Produktionsanlagen / Technologie-Initiative SmartFactory KL e.V. Kaiserslautern, April 2018 (1.2). – Whitepaper. – 42 S.

KJ20. Kirks, Thomas; Jost, Jana: Mensch-Technik-Interaktion in Industrie-4.0-Umgebungen am Beispiel von EMILI. In: ten Hompel, Michael (Hrsg.); Bauernhansl, Thomas

(Hrsg.); Vogel-Heuser, Birgit (Hrsg.): *Handbuch Industrie 4.0: Band 3: Logistik*. Berlin, Heidelberg: Springer, 2020. – ISBN 978–3–662–58530–6, S. 529–539

KJU+18. Kirks, T.; Jost, J.; Uhlott, T.; Jakobs, M.: Towards Complex Adaptive Control Systems for Human-Robot-Interaction in Intralogistics*. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018. – ISSN 2153–0017, S. 2968–2973

KKH+13. Koskela, Timo; Kassinen, Otso; Harjula, Erkki; Ylianttila, Mika: P2P Group Management Systems: A Conceptual Analysis. In: *ACM Computing Surveys* 45 (2013), Februar, Nr. 2, S. 1–25. – ISSN 0360–0300, 1557–7341

KMG03. Kermarrec, A.-M.; Massoulie, L.; Ganesh, A.J.: Probabilistic Reliable Dissemination in Large-Scale Systems. In: *IEEE Transactions on Parallel and Distributed Systems* 14 (2003), März, Nr. 3, S. 248–258. – ISSN 1558–2183

Knu98. Knuth, Donald E.: *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. USA: Addison Wesley Longman Publishing Co., Inc., 1998. – ISBN 978–0–201–89685–5

Kor10. Koren, Yoram: *The Global Manufacturing Revolution: Product-Process-Business Integration and Reconfigurable Systems*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2010. – ISBN 978–0–470–61881–3 978–0–470–58377–7

Kow79. Kowalski, Robert: Algorithm = Logic + Control. In: *Communications of the ACM* 22 (1979), Juli, Nr. 7, S. 424–436. – ISSN 0001–0782

KPT+10. Kumar, Vishwa V.; Pandey, Mayank K.; Tiwari, M. K.; Ben-Arieh, David: Simultaneous Optimization of Parts and Operations Sequences in SSMS: A Chaos Embedded Taguchi Particle Swarm Optimization Approach. In: *Journal of Intelligent Manufacturing* 21 (2010), August, Nr. 4, S. 335–353. – ISSN 0956–5515, 1572–8145

Kra09. Kramer, Joshua: Advanced Message Queuing Protocol (AMQP). In: *Linux Journal* 2009 (2009), November, Nr. 187, S. 3:3. – ISSN 1075–3583

KSD13. Korsah, G. A.; Stentz, Anthony; Dias, M. B.: A Comprehensive Taxonomy for Multi-Robot Task Allocation. In: *The International Journal of Robotics Research* 32 (2013), Oktober, Nr. 12, S. 1495–1512. – ISSN 0278–3649, 1741–3176

KSP+10. Kraetzschmar, Gerhard; Shakhimardanov, Azamat; Paulus, Jan; Hochgeschwender, Nico; Reckhaus, Michael: Specifications of Architectures, Modules, Modularity, and Interfaces for the BROCRE Software Platform and Robot Control Architecture Workbench. Bonn, August 2010 (Deliverable D-2.2). – Forschungsbericht. – 70 S.

Lam98. Lamport, Leslie: The Part-Time Parliament. In: *ACM Transactions on Computer Systems* 16 (1998), Mai, Nr. 2, S. 133–169. – ISSN 0734–2071

Lam01. Lamport, Leslie: Paxos Made Simple. In: *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001), Dezember, S. 51–58

Lap17. Laplante, Philip A.: *What Every Engineer Should Know about Software Engineering*. 1st. USA: CRC Press, Inc., 2017. – ISBN 1–138–46853–3

Las21. Laskowski, Patrick: *Optimierung der Positionsfindung in null-balancierten Baumstrukturen für Peer-to-Peer Netzwerke*. Bochum, Ruhr-Universität Bochum, Bachelorarbeit, November 2021

LBK15. Lee, Jay; Bagheri, Behrad; Kao, Hung-An: A Cyber-Physical Systems Architecture for Industry 4.0-Based Manufacturing Systems. In: *Manufacturing Letters* 3 (2015), Januar, S. 18–23. – ISSN 2213–8463

LDE+22. Lünsch, Dennis; Detzner, Peter; Ebner, Andreas; Kerner, Soren: SWAP-IT: A Scalable and Lightweight Industry 4.0 Architecture for Cyber-Physical Production Systems. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. Mexico City, 2022, S. 7

LFK+14. Lasi, Heiner; Fettke, Peter; Kemper, Hans-Georg; Feld, Thomas; Hoffmann, Michael: Industry 4.0. In: *Business & Information Systems Engineering* 6 (2014), August, Nr. 4, S. 239–242. – ISSN 1867–0202

Lib11. Libert, Sergey: *Beitrag zur agentenbasierten Gestaltung von Materialflusssteuerungen*. Dortmund: TU Dortmund, 2011 (1). – ISBN 978–3–86975–049–1

LM90. Laporte, Gilbert; Martello, Silvano: The Selective Travelling Salesman Problem. In: *Discrete Applied Mathematics* 26 (1990), März, Nr. 2, S. 193–207. – ISSN 0166–218X

LMP+18. Lima, Keila; Marques, Eduardo R.; Pinto, Jose; Sousa, Joao B.: Dolphin: A Task Orchestration Language for Autonomous Vehicle Networks. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018. – ISSN 2153–0866, S. 603–610

LPB+15. Luzuriaga, Jorge E.; Perez, Miguel; Boronat, Pablo; Cano, Juan C.; Calafate, Carlos; Manzoni, Pietro: A Comparative Evaluation of AMQP and MQTT Protocols over Unstable and Mobile Networks. In: *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015. – ISSN 2331–9860, S. 931–936

LPR07. Leitao, Joao; Pereira, Jose; Rodrigues, Luis: Epidemic Broadcast Trees. In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, 2007. – ISSN 1060–9857, S. 301–310

LTL11. Leu, Jenq-Shiou; Tsai, Cheng-Wei; Lin, Wei-Hsiang: Resource Searching in an Unstructured P2P Network Based on Cloning Random Walker Assisted by Dominating Set. In: *Computer Networks* 55 (2011), Februar, Nr. 3, S. 722–733. – ISSN 1389–1286

Lüt17. Lüttge, Ulrich: Skalierungen und die Hierarchie der Module. In: Lüttge, Ulrich (Hrsg.): *Faszination Pflanzen*. Berlin, Heidelberg: Springer, 2017. – ISBN 978–3–662–52983–6, S. 137–144

LW05. Li, Xiuqi; Wu, Jie: Searching Techniques in Peer-to-Peer Networks. In: *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks* (2005), S. 31

LZL+05. Lo, V.; Zhou, Dayi; Liu, Yuhong; GauthierDickey, C.; Li, Jun: Scalable Supernode Selection in Peer-to-Peer Overlay Networks. In: *Second International Workshop on Hot Topics in Peer-to-Peer Systems*, 2005, S. 18–25

Mad19. Madden, Michael M.: Challenges Using the Linux Network Stack for Real-Time Communication. In: *AIAA Scitech 2019 Forum*. San Diego, California: American Institute of Aeronautics and Astronautics, Januar 2019. – ISBN 978–1–62410–578–4

Man. *Manufacturing USA*. https://www.manufacturingusa.com/, . – letzter Zugriff: 30.06.2021

Mar08. Martin, Robert C.: *Clean Code: A Handbook of Agile Software Craftsmanship*. First. USA: Prentice Hall PTR, 2008. – ISBN 0–13–235088–2

Mar14. Martin, Heinrich: *Transport- und Lagerlogistik: Planung, Struktur, Steuerung und Kosten von Systemen der Intralogistik*. 9. Springer Vieweg, 2014. – ISBN 978–3–658–03143–5

MAS+18. Mabkhot, Mohammed; Al-Ahmari, Abdulrahman; Salah, Bashir; Alkhalefah, Hisham: Requirements of the Smart Factory System: A Survey and Perspective. In: *Machines* 6 (2018), Juni, Nr. 2, S. 23. – ISSN 2075–1702

MCL20. Manterola-Lasa, Aratz; Casado-Mansilla, Diego; López-de-Ipiña, Diego: UnsServ: Unstructured Peer-to-Peer Library for Deploying Services in Smart Environments. In: *2020 5th International Conference on Smart and Sustainable Technologies (SpliTech)*, 2020, S. 1–6

Mey97. Meyer, Bertrand: *Object-Oriented Software Construction (2nd Ed.)*. USA: Prentice-Hall, Inc., 1997. – ISBN 0–13–629155–4

MFH01. Mentzer, John; Flint, Daniel; Hult, G. Tomas M.: Logistics Service Quality as a Segment-Customized Process. In: *Journal of Marketing* 65 (2001), Oktober, S. 82–104

MG20. Masinde, Newton; Graffi, Kalman: Peer-to-Peer-Based Social Networks: A Comprehensive Survey. In: *SN Computer Science* 1 (2020), September, Nr. 5, S. 299. – ISSN 2661–8907

MHC06. Min, Su-hong; Holliday, Joanne; Cho, Dong-sub: Optimal Super-peer Selection for Large-scale P2P System. In: *2006 International Conference on Hybrid Information Technology* Bd. 2, 2006, S. 588–593

MHS05. Mernik, Marjan; Heering, Jan; Sloane, Anthony M.: When and How to Develop Domain-Specific Languages. In: *ACM Computing Surveys* 37 (2005), Dezember, Nr. 4, S. 316–344. – ISSN 0360–0300

MIA17. Meissner, Hermann; Ilsen, Rebecca; Aurich, Jan C.: Analysis of Control Architectures in the Context of Industry 4.0. In: *Procedia CIRP* 62 (2017), Januar, S. 165–169. – ISSN 2212–8271

MKI+16. Maqsood, Tahir; Khalid, Osman; Irfan, Rizwana; Madani, Sajjad A.; Khan, Samee U.: Scalability Issues in Online Social Networks. In: *ACM Computing Surveys* 49 (2016), November, Nr. 2, S. 1–42. – ISSN 0360–0300, 1557–7341

MKL+. Milojicic, Dejan S.; Kalogeraki, Vana; Lukose, Rajan; Nagaraja, Kiran; Pruyne, Jim; Richard, Bruno; Rollins, Sami; Xu, Zhichen: Peer-to-Peer Computing.

MLD09. Mahnke, Wolfgang; Leitner, Stefan-Helmut; Damm, Matthias: *OPC Unified Architecture*. 1st. Springer Publishing Company, Incorporated, 2009. – ISBN 3–540–68898–6

MM02. Maymounkov, Petar; Mazières, David: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: *Peer-to-Peer Systems* Bd. 2429. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. – ISBN 978–3–540–45748–0, S. 53–65

MM04. Milanovic, N.; Malek, M.: Current Solutions for Web Service Composition. In: *IEEE Internet Computing* 8 (2004), November, Nr. 6, S. 51–59. – ISSN 1941–0131

Mon14. Monostori, László: Cyber-Physical Production Systems: Roots, Expectations and R&D Challenges. In: *Procedia CIRP* 17 (2014), Januar, S. 9–13. – ISSN 2212–8271

Mon17. In: Montresor, Alberto: *Gossip and Epidemic Protocols*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2017. – ISBN 978–0–471–34608–1, S. 1–15

MP17. McBride, Mike; Perkins, Charles: *Multicast Wifi Problem Statement*. https://www.ietf.org/archive/id/draft-mcbride-mboned-wifi-mcast-problem-statement-01.txt, Oktober 2017. – letzter Zugriff: 17.08.2021

MPP14. Moser, Michael; Pfeiffer, Michael; Pichler, Josef: A Novel Domain-Specific Language for the Robot Welding Automation Domain. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014. – ISSN 1946–0759, S. 1–6

MQT22. *MQTT - The Standard for IoT Messaging*. https://mqtt.org/, 2022. – letzter Zugriff: 16.05.2022

MRP+15. Marques, Eduardo R. B.; Ribeiro, Manuel; Pinto, Jose; Sousa, Joao B.; Martins, Francisco: NVL: A Coordination Language for Unmanned Vehicle Networks. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. Salamanca Spain: ACM, April 2015. – ISBN 978–1–4503–3196–8, S. 331–334

muh06. Engineering of Event-Based Systems. In: Mühl, Gero (Hrsg.); Fiege, Ludger (Hrsg.); Pietzuch, Peter (Hrsg.): *Distributed Event-Based Systems*. Berlin, Heidelberg: Springer, 2006. – ISBN 978–3–540–32653–3, S. 129–148

Mur89. Murata, T.: Petri Nets: Properties, Analysis and Applications. In: *Proceedings of the IEEE* 77 (1989), April, Nr. 4, S. 541–580. – ISSN 1558–2256

MVD+15. Monostori, László; Valckenaers, Paul; Dolgui, Alexandre; Panetto, Hervé; Brdys, Mietek; Csáji, Balázs C.: Cooperative Control in Production and Logistics. In: *Annual Reviews in Control* 39 (2015), Januar, S. 12–29. – ISSN 1367–5788

MZW07. Mors, A.; Zutt, J.; Witteveen, C.: Context-Aware Logistic Routing and Scheduling. In: *ICAPS*, 2007

NAI+03. Nau, D. S.; Au, T. C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; Yaman, F.: SHOP2: An HTN Planning System. In: *Journal of Artificial Intelligence Research* 20 (2003), Dezember, S. 379–404. – ISSN 1076–9757

Nai17. Naik, Nitin: Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP. In: *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, S. 1–7

NFH10. Nopper, Jan; Follert, Guido; ten Hompel, Michael: Gains in the Life-Cycle of Adaptable, Self-Organizing Material Handling Systems. In: *Progress in Material Handling Research: 2010* (2010), September, S. 11. – ISSN 9781882780167

NHW14. Nordmann, Arne; Hochgeschwender, Nico; Wrede, Sebastian: A Survey on Domain-Specific Languages in Robotics. In: Brugali, Davide (Hrsg.); Broenink, Jan F. (Hrsg.); Kroeger, Torsten (Hrsg.); MacDonald, Bruce A. (Hrsg.): *Simulation, Modeling, and Programming for Autonomous Robots*. Cham: Springer International Publishing, 2014 (Lecture Notes in Computer Science). – ISBN 978–3–319–11900–7, S. 195–206

NHW+16. Nordmann, Arne; Hochgeschwender, Nico; Wigand, Dennis L.; Wrede, Sebastian: A Survey on Domain-Specific Modeling and Languages in Robotics. In: *Journal of Software Engineering in Robotics (JOSER)* 7 (2016), Nr. 1, S. 75–99

NMG16. Nunes, Ernesto; McIntire, Mitchell; Gini, Maria: Decentralized Allocation of Tasks with Temporal and Precedence Constraints to a Team of Robots. In: *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2016, S. 197–202

OAS06. OASIS SOA Reference Model (SOA-RM) Technical Committee: Service Oriented Architecture Reference Model 1.0. 2006. – Forschungsbericht. – 31 S.

OAS12. OASIS: Advanced Message Queuing Protocol (AMQP) Version 1.0. 2012. – Forschungsbericht

OAS19. OASIS Message Queuing Telemetry Transport: *MQTT Version 5.0*. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, März 2019. – letzter Zugriff: 30.11.2021

Obj15. Object Management Group: *OMG Data Distribution Service (DDS) 1.4*. Oktober 2015

Obj21. Object Management Group: *The Real-time Publish-SubscribeProtocol DDS Interoperability Wire Protocol(DDSI-RTPSTM) Specification 2.5*. März 2021

OKF10. Oh, Sangyoon; Kim, Jai-Hoon; Fox, Geoffrey: Real-Time Performance Analysis for Publish/Subscribe Systems. In: *Future Generation Computer Systems* 26 (2010), März, Nr. 3, S. 318–323. – ISSN 0167–739X

OLK+21. Oughton, Edward J.; Lehr, William; Katsaros, Konstantinos; Selinis, Ioannis; Bubley, Dean; Kusuma, Julius: Revisiting Wireless Internet Connectivity: 5G vs Wi-Fi 6. In: *Telecommunications Policy* 45 (2021), Juni, Nr. 5, S. 102127. – ISSN 0308–5961

OO14. Ongaro, Diego; Ousterhout, John: In Search of an Understandable Consensus Algorithm. In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014. – ISBN 978–1–931971–10–2, S. 305–319

Ope12. Open Mobile Alliance: *NGSI Context Management*. Mai 2012

OST+20. Oyekanlu, Emmanuel A.; Smith, Alexander C.; Thomas, Windsor P.; Mulroy, Grethel; Hitesh, Dave; Ramsey, Matthew; Kuhn, David J.; Mcghinnis, Jason D.; Buonavita, Steven C.; Looper, Nickolus A.; Ng, Mason; Ng'oma, Anthony; Liu, Weimin; Mcbride, Patrick G.; Shultz, Michael G.; Cerasi, Craig; Sun, Dan: A Review of Recent Advances in Automated Guided Vehicle Technologies: Integration Challenges and Research Areas for 5G-Based Smart Manufacturing Applications. In: *IEEE Access* 8 (2020), S. 202312–202353. – ISSN 2169–3536

Par03a. Pardo-Castellote, G.: OMG Data-Distribution Service: Architectural Overview. In: *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, S. 200–206

Par03b. Parsons, Rebecca: Components and the World of Chaos. In: *IEEE Software* 20 (2003), Mai, Nr. 3, S. 83–85. – ISSN 0740–7459

Par13. Parr, Terence: *The Definitive ANTLR 4 Reference*. Second. Pragmatic Bookshelf, 2013. – ISBN 978–1–934356–99–9

PB16. Pinciroli, Carlo; Beltrame, Giovanni: Buzz: An Extensible Programming Language for Heterogeneous Swarm Robotics. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016. – ISSN 2153–0866, S. 3794–3800

PEK+07. Pietzuch, Peter; Eyers, David; Kounev, Samuel; Shand, Brian: Towards a Common API for Publish/Subscribe. In: *Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems - DEBS '07*. Toronto, Ontario, Canada: ACM Press, 2007. – ISBN 978–1–59593–665–3, S. 152

PGP16. Pfrommer, Julius; Grüner, Sten; Palm, Florian: Hybrid OPC UA and DDS: Combining Architectural Styles for the Industrial Internet. In: *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, 2016, S. 1–7

Pla16a. Platform Industrie 4.0: Aspects of the Research Roadmap in Application Scenarios / Federal Ministry for Economic Affairs and Energy (BMWi). Berlin, April 2016. – Forschungsbericht. – 40 S.

Pla16b. Plattform Industrie 4.0: Network-Based Communication for Industrie 4.0 – Proposal for an Administration Shell / Federal Ministry for Economic Affairs and Energy (BMWi). Berlin, November 2016. – Forschungsbericht. – 28 S.

Pla20. Plattform Industrie 4.0: Details of the Asset Administration Shell - Part 1 - The Exchange of Information between Partners in the Value Chain of Industrie 4.0 / Federal Ministry for Economic Affairs and Climate Action (BMWK). Berlin, November 2020 (Version 3.0RC01). – Forschungsbericht. – 523 S.

PMS+. Perkins, Charles E.; McBride, Mike; Stanley, Dorothy; Kumari, Warren; Zuniga, Juan-Carlos: Multicast Considerations over IEEE 802 Wireless Media / Internet Engineering Task Force (draft-ietf-mboned-ieee802-mcast-problems-13). – Internet Draft. – Work in Progress

Pnu77. Pnueli, Amir: The Temporal Logic of Programs. In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. USA: IEEE Computer Society, September 1977 (SFCS '77), S. 46–57

PTD+19. Profanter, Stefan; Tekat, Ayhun; Dorofeev, Kirill; Rickert, Markus; Knoll, Alois: OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols. In: *2019 IEEE International Conference on Industrial Technology (ICIT)*, 2019. – ISSN 2643–2978, S. 955–962

PZC+16. Pei, Changhua; Zhao, Youjian; Chen, Guo; Tang, Ruming; Meng, Yuan; Ma, Minghua; Ling, Ken; Pei, Dan: WiFi Can Be the Weakest Link of Round Trip Network Latency in the Wild. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, S. 1–9

QGC+. Quigley, Morgan; Gerkey, Brian; Conley, Ken; Faust, Josh; Foote, Tully; Leibs, Jeremy; Berger, Eric; Wheeler, Rob; Ng, Andrew: ROS: An Open-Source Robot Operating System.

QNF20. Quadrini, Walter; Negri, Elisa; Fumagalli, Luca: Open Interfaces for Connecting Automated Guided Vehicles to a Fleet Management System. In: *Procedia Manufacturing* 42 (2020), Januar, S. 406–413. – ISSN 2351–9789

QVL02. Qayyum, A.; Viennot, L.; Laouiti, A.: Multipoint Relaying for Flooding Broadcast Messages in Mobile Wireless Networks. In: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, 2002, S. 3866–3875

Ram13. Rampfl, Sebastian: Network Simulation and Its Limitations / TU München. München, 2013. – Seminar

RCJ09. Rogers, Alex; Corkill, Daniel D.; Jennings, Nicholas R.: Agent Technologies for Sensor Networks. In: *IEEE Intelligent Systems* 24 (2009), März, Nr. 2, S. 13–17. – ISSN 1941–1294

RD01. Rowstron, Antony; Druschel, Peter: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Guerraoui, Rachid (Hrsg.): *Middleware 2001*. Berlin, Heidelberg: Springer, 2001 (Lecture Notes in Computer Science). – ISBN 978–3–540–45518–9, S. 329–350

Reg08. Regele, Ralf: *Kooperative Multi-Roboter-Wegplanung durch heuristische Prioritätsanpassung*. Logos Verlag Berlin GmbH, 2008. – ISBN 978–3–8325–2028–1

RFH+. Ratnasamy, Sylvia; Francis, Paul; Handley, Mark; Shenker, Scott; Karp, Richard: A Scalable Content-Addressable Network.

RG06. Rahman, A.; Gburzynski, P.: Hidden Problems with the Hidden Node Problem. In: *23rd Biennial Symposium on Communications*. Kingston, ON, Canada: IEEE, Mai 2006, S. 270–273

RH10. Riley, George F.; Henderson, Thomas R.: The Ns-3 Network Simulator. In: Wehrle, Klaus (Hrsg.); Güneş, Mesut (Hrsg.); Gross, James (Hrsg.): *Modeling and Tools for*

Network Simulation. Berlin, Heidelberg: Springer, 2010. – ISBN 978–3–642–12331–3, S. 15–34

Rip01. Ripeanu, M.: Peer-to-Peer Architecture Case Study: Gnutella Network. In: *Proceedings First International Conference on Peer-to-Peer Computing*, 2001, S. 99–100

RKC+01. Rowstron, Antony; Kermarrec, Anne-Marie; Castro, Miguel; Druschel, Peter: Scribe: The Design of a Large-Scale Event Notification Infrastructure. In: Goos, Gerhard (Hrsg.); Hartmanis, Juris (Hrsg.); van Leeuwen, Jan (Hrsg.); Crowcroft, Jon (Hrsg.); Hofmann, Markus (Hrsg.): *Networked Group Communication* Bd. 2233. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. – ISBN 978–3–540–42824–4 978–3–540–45546–2, S. 30–43

RM06. Risson, John; Moors, Tim: Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. In: *Computer Networks* 50 (2006), Dezember, Nr. 17, S. 3485–3521. – ISSN 1389–1286

Roi22. Roidl, Moritz: *Zur innerbetrieblichen Logistik - Axiomatik und Betrachtung als kinodynamisches System*. Dortmund, TU Dortmund, Diss., 2022

RP12. Raja, P.; Pugazhenthi, S.: Optimal Path Planning of Mobile Robots: A Review. In: *International Journal of the Physical Sciences* 7 (2012), Februar, Nr. 9. – ISSN 19921950

SA98. Simmons, R.; Apfelbaum, D.: A Task Description Language for Robot Control. In: *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)* Bd. 3, 1998, S. 1931–1937 vol.3

SA12. Selvaraj, Chithra; Anand, Sheila: A Survey on Security Issues of Reputation Management Systems for Peer-to-Peer Networks. In: *Computer Science Review* 6 (2012), Juli, Nr. 4, S. 145–160. – ISSN 1574–0137

SAD+20. Schuh, Günther; Anderl, Reiner; Dumitrescu, Roman; Krüger, Antonio; ten Hompel, Michael: Industrie 4.0 Maturity Index. Die digitale Transformation von Unternehmen gestalten – UPDATE 2020. München, 2020. – Forschungsbericht. – 64 S.

Sch01. Schollmeier, R.: A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: *Proceedings First International Conference on Peer-to-Peer Computing*, 2001, S. 101–102

Sch18a. Schmidt, Michael: *Distribution Center Design Process - Ein Systemtechnikorientiertes Vorgehensmodell Zur Konzeptplanung von Logistikzentren*. 2018. – ISBN 978–3–86975–129–0

Sch18b. Scholz, Daniel: Multikriterielle Optimierung. In: *Optimierung Interaktiv: Grundlagen Verstehen, Modelle Erforschen Und Verfahren Anwenden*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018. – ISBN 978–3–662–57953–4, S. 169–187

SCI03. Shukla, Deepanshu; Chandran-Wadia, Leena; Iyer, Sridhar: Mitigating the Exposed Node Problem in IEEE 802.11 Ad Hoc Networks. In: *Proceedings. 12th International Conference on Computer Communications and Networks (IEEE Cat. No.03EX712)*, 2003. – ISSN 1095–2055, S. 157–162

SGCS+21. Seitz, Matthias; Gehlhoff, Felix; Cruz Salazar, Luis A.; Fay, Alexander; Vogel-Heuser, Birgit: Automation Platform Independent Multi-Agent System for Robust Networks of Production Resources in Industry 4.0. In: *Journal of Intelligent Manufacturing* 32 (2021), Oktober, Nr. 7, S. 2023–2041. – ISSN 0956–5515, 1572–8145

Sha48. Shannon, C. E.: A Mathematical Theory of Communication. In: *The Bell System Technical Journal* 27 (1948), Juli, Nr. 3, S. 379–423. – ISSN 0005–8580

SJG19. Surati, Shivangi; Jinwala, Devesh C.; Garg, Sanjay: BMMI-tree: A Peer-to-Peer m-Ary Tree Using 1-m Node Splitting for an Efficient Multidimensional Complex Query Search. In: *Journal of Parallel and Distributed Computing* 125 (2019), März, S. 1–17. – ISSN 07437315

SK16. Siciliano, Bruno; Khatib, Oussama: *Springer Handbook of Robotics*. Cham: Springer International Publishing, 2016. – ISBN 978–3–319–32551–4

SLS21. Stenzel, Jonas; Lünsch, Dennis; Schmitz, Lea: Automated Topology Creation for Global Path Planning of Large AGV Fleets. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021, S. 3373–3380

SM19. Sutherland, Craig J.; MacDonald, Bruce: RoboLang: A Simple Domain Specific Language to Script Robot Interactions. In: *2019 16th International Conference on Ubiquitous Robots (UR)*, 2019. – ISSN 2325–033X, S. 265–270

SMD+19. Shi-Wan Lin; Miller, Bradford; Durand, Jacques; Bleakley, Graham; Chigani, Amine; Martin, Robert; Murphy, Brett; Crawford, Mark: The Industrial Internet of Things Volume G1: Reference Architecture / Industrial Internet Consortium. 2019 (1.9). – Forschungsbericht. – 58 S.

Smi80. Smith: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. In: *IEEE Transactions on Computers* C-29 (1980), Dezember, Nr. 12, S. 1104–1113. – ISSN 1557–9956

SMK+01. Stoica, Ion; Morris, Robert; Karger, David; Kaashoek, M. F.; Balakrishnan, Hari: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: *Sigcomm'01*, 2001, S. 149–160

SMM+18. Solomon, Eldra; Martin, Charles; Martin, Diana W.; Berg, Linda R.: *Biology*. 11th edition. Australia ; Boston, MA: Cengage Learning, 2018. – ISBN 978–1–337–39293–8

SMS+17. Shafi, Mansoor; Molisch, Andreas F.; Smith, Peter J.; Haustein, Thomas; Zhu, Peiying; De Silva, Prasan; Tufvesson, Fredrik; Benjebbour, Anass; Wunder, Gerhard: 5G: A Tutorial Overview of Standards, Trials, Challenges, Deployment, and Practice. In: *IEEE Journal on Selected Areas in Communications* 35 (2017), Juni, Nr. 6, S. 1201–1221. – ISSN 1558–0008

SN15. Sidorov, Vasily; Ng, Wee K.: Transparent Data Encryption for Data-in-Use and Data-at-Rest in a Cloud-Based Database-as-a-Service Solution. In: *2015 IEEE World Congress on Services*, 2015. – ISSN 2378–3818, S. 221–228

SPP+19. Seder, Marija; Petrović, Luka; Peršić, Juraj; Popović, Goran; Petković, Tomislav; Šelek, Ana; Bićanić, Borna; Cvišić, Igor; Josić, Damir; Marković, Ivan; Petrović, Ivan; Muhammad, Ali: Open Platform Based Mobile Robot Control for Automation in Manufacturing Logistics. In: *IFAC-PapersOnLine* 52 (2019), Januar, Nr. 22, S. 95–100. – ISSN 2405–8963

ST09. Salehie, Mazeiar; Tahvildari, Ladan: Self-Adaptive Software: Landscape and Research Challenges. In: *ACM Transactions on Autonomous and Adaptive Systems* 4 (2009), Mai, Nr. 2, S. 14:1–14:42. – ISSN 1556–4665

ST17. van Steen, Maarten; Tanenbaum, Andrew S.: *Distributed Systems*. Third edition (Version 3.01 (2017)). London: Pearson Education, 2017. – ISBN 978–1–5430–5738–6 978–90–815406–2–9. – OCLC: 1006750554

Sta16. Statistisches Bundesamt: *Number of IoT Devices 2015-2025*. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2016. – letzter Zugriff: 25.03.2021

Sta18. Statistisches Bundesamt: *Marktprognosen zum Internet der Dinge*. Hamburg, 2018

Sta20. Statistisches Bundesamt: *IoT Connected Devices Worldwide 2019-2030*. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/, Dezember 2020. – letzter Zugriff: 01.09.2021

Sto22. Stolz, Oliver: *Einfluss der Netzwerktopologie auf die Kommunikation am Beispiel der dezentralen Pfadplanung für Fahrerlose Transportfahrzeuge*. Bochum, Ruhr-Universität Bochum, Bachelorarbeit, Juni 2022

SVL+17. St-Onge, David; Varadharajan, Vivek; Li, Guannan; Svogor, Ivan; Beltrame, Giovanni: ROS and Buzz: Consensus-Based Behaviors for Heterogeneous Teams. (2017), Oktober

TBF+. Terpstra, Wesley W.; Behnel, Stefan; Fiege, Ludger; Zeidler, Andreas; Buchmann, Alejandro P.: A Peer-to-Peer Approach to Content-Based Publish/Subscribe.

TE15. Trillas, Enric; Eciolaza, Luka: *Studies in Fuzziness and Soft Computing*. Bd. 320: *Fuzzy Logic*. Cham: Springer International Publishing, 2015. – ISBN 978–3–319–14202–9 978–3–319–14203–6

Tel21. Telefonica I+D: *Orion Context Broker*. Telefónica I+D, Mai 2021

tH20. ten Hompel, Michael; Henke, Michael: Logistik 4.0 in der Silicon Economy. In: ten Hompel, Michael (Hrsg.); Bauernhansl, Thomas (Hrsg.); Vogel-Heuser, Birgit (Hrsg.): *Handbuch Industrie 4.0: Band 3: Logistik*. Berlin, Heidelberg: Springer, 2020. – ISBN 978–3–662–58530–6, S. 3–9

TLM+19. Tarr, Dominic; Lavoie, Erick; Meyer, Aljoscha; Tschudin, Christian: Secure Scuttlebutt: An Identity-Centric Protocol for Subjective and Decentralized Applications. In: *Proceedings of the 6th ACM Conference on Information-Centric Networking*. Macao China: ACM, September 2019. – ISBN 978–1–4503–6970–1, S. 1–11

Tön22. Tönning, Lars: *Effizienter Broadcast in einem null-balancierten baumstrukturierten Peer-to-Peer Netzwerk*. Dortmund, TU Dortmund, Bachelorarbeit, November 2022

TPS+19. Trabesinger, Stefan; Pichler, Rudolf; Schall, Daniel; Gfrerer, Richard: Connectivity as a Prior Challenge in Establishing CPPS on Basis of Heterogeneous IT-software Environments. In: *Procedia Manufacturing* 31 (2019), Januar, S. 370–376. – ISSN 2351–9789

TR03. Tsoumakos, Dimitrios; Roussopoulos, Nick: A Comparison of Peer-to-Peer Search Methods. In: *International Workshop on the Web and Databases (WebDB)* 6th (2003), S. 6

TS15. Taori, Rakesh; Sridharan, Arun: Point-to-Multipoint in-Band Mmwave Backhaul for 5G Networks. In: *IEEE Communications Magazine* 53 (2015), Januar, Nr. 1, S. 195–201. – ISSN 1558–1896

tSB11a. ten Hompel, Michael; Sadowsky, Volker; Beck, Maria: Systemtypen in der Kommissionierung. In: ten Hompel, Michael (Hrsg.); Sadowsky, Volker (Hrsg.); Beck, Maria (Hrsg.): *Kommissionierung: Materialflusssysteme 2 - Planung und Berechnung der Kommissionierung in der Logistik*. Berlin, Heidelberg: Springer, 2011 (VDI-Buch). – ISBN 978–3–540–29940–0, S. 65–90

tSB11b. ten Hompel, Michael; Sadowsky, Volker; Beck, Maria: Technische Komponenten von Kommissioniersystemen. In: ten Hompel, Michael (Hrsg.); Sadowsky, Volker (Hrsg.); Beck, Maria (Hrsg.): *Kommissionierung: Materialflusssysteme 2 - Planung und Berechnung der Kommissionierung in der Logistik*. Berlin, Heidelberg: Springer, 2011 (VDI-Buch). – ISBN 978–3–540–29940–0, S. 43–64

TWB+10. Tompkins, J. A.; White, J. A.; Bozer, Y. A.; Tanchoco, J.M.A.: *Facilities Planning*. Wiley, 2010. – ISBN 978–0–470–44404–7

UGG16. Ungurean, Ioan; Gaitan, Nicoleta-Cristina; Gaitan, Vasile: A Middleware Based Architecture for the Industrial Internet of Things. 10 (2016), Juli, S. 2874–2891

Ull15. Ullrich, Günter: *Automated Guided Vehicle Systems: A Primer with Practical Applications*. Berlin Heidelberg: Springer-Verlag, 2015. – ISBN 978–3–662–44813–7

VB. Vidal, Thierry; Bidot, Julien: Dynamic Sequencing of Tasks in Simple Temporal Networks with Uncertainty.

VBM77. Vollmer, W.; Binder, W.; Minckwitz, J.: *Dr. Vollmer's Wörterbuch Der Mythologie Aller Völker*. Zentralantiquariat der Deutschen Demokratischen Republik, 1977

VC11. Van Cutsem, Tom: *Why Programming Languages?* http://soft.vub.ac.be/~tvcutsem/whypls.html, November 2011. – letzter Zugriff: 21.03.2021

VDI07. VDI-Fachbereich Technische Logistik: *VDI 3586-Flurförderzeuge - Begriffe, Kurzzeichen, Beispiele*. November 2007

Ver13. Verein Deutscher Ingenieure e.V: *Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation*. https://www.vdi.de/ueber-uns/presse/publikationen/details/cyber-physical-systems-chancen-und-nutzen-aus-sicht-der-automation, März 2013. – letzter Zugriff: 05.07.2021

Ver20. Verband der Automobilindustrie e.v. (VDA): *Interface for the Communication between Automated Guided Vehicles (AGV) and a Master Control*. Juni 2020

vKV00. van Deursen, Arie; Klint, Paul; Visser, Joost: Domain-Specific Languages: An Annotated Bibliography. In: *ACM SIGPLAN Notices* 35 (2000), Juni, Nr. 6, S. 26–36. – ISSN 0362–1340, 1558–1160

Voe13. Voelter, Markus: *DSL Engineering: Designing, Implementing and Using Domain-specific Languages*. CreateSpace Independent Publishing Platform, 2013. – ISBN 978–1–4812–1858–0

VRA15. Van Renesse, Robbert; Altinbuken, Deniz: Paxos Made Moderately Complex. In: *ACM Computing Surveys* 47 (2015), Februar, Nr. 3, S. 42:1–42:36. – ISSN 0360–0300

VRH04. Van Roy, Peter; Haridi, Seif: *Concepts, Techniques, and Models of Computer Programming*. 1st. The MIT Press, 2004. – ISBN 978–0–262–22069–9

VRM+16. Vivaldini, Kelen; Rocha, Luís F.; Martarelli, Nádia J.; Becker, Marcelo; Moreira, A. P.: Integrated Tasks Assignment and Routing for the Estimation of the Optimal Number of AGVS. In: *The International Journal of Advanced Manufacturing Technology* 82 (2016), Januar, Nr. 1, S. 719–736. – ISSN 1433–3015

WD06. Westkämper, Engelbert; Decker, Markus: *Einführung in die Organisation der Produktion*. Berlin Heidelberg: Springer, 2006 (Springer-Lehrbuch). – ISBN 978–3–540–26039–4

WE16. Weyrich, Michael; Ebert, Christof: Reference Architectures for the Internet of Things. In: *IEEE Software* 33 (2016), Januar, Nr. 1, S. 112–116. – ISSN 1937–4194

Weh20. Wehking, Karl-Heinz: *Technisches Handbuch Logistik 1: Fördertechnik, Materialfluss, Intralogistik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020. – ISBN 978–3–662–60866–1 978–3–662–60867–8

WG18. Wüst, Karl; Gervais, Arthur: Do You Need a Blockchain? In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, S. 45–54

WH07. Windt, Katja; Hülsmann, Michael: Changing Paradigms in Logistics — Understanding the Shift from Conventional Control to Autonomous Cooperation and Control. In: Hülsmann, Michael (Hrsg.); Windt, Katja (Hrsg.): *Understanding Autonomous Cooperation and Control in Logistics: The Impact of Autonomy on Management, Information, Communication and Material Flow*. Berlin, Heidelberg: Springer, 2007. – ISBN 978–3–540–47450–0, S. 1–16

WJ95. Wooldridge, Michael; Jennings, Nicholas R.: Intelligent Agents: Theory and Practice. In: *The Knowledge Engineering Review* 10 (1995), Juni, Nr. 2, S. 115–152. – ISSN 1469–8005, 0269–8889

WL99. Wu, Jie; Li, Hailan: On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks. In: *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications - DIALM '99*. Seattle, Washington, United States: ACM Press, 1999. – ISBN 978–1–58113–174–1, S. 7–14

WMO+16. Weyer, Stephan; Meyer, Torben; Ohmer, Moritz; Gorecky, Dominic; Zühlke, Detlef: Future Modeling and Simulation of CPS-based Factories: An Example from the Automotive Industry. In: *IFAC-PapersOnLine* 49 (2016), Januar, Nr. 31, S. 97–102. – ISSN 2405–8963

WPS+11. Watson, David S.; Piette, Mary A.; Sezgen, Osman; Motegi, Naoya: Machine to Machine (M2M) Technology in Demand Responsive Commercial Buildings. In: *Handbook of Web Based Energy Information and Control Systems*. River Publishers, 2011. – ISBN 978–1–00–315167–8

WSF+17. Walenta, Robert; Schellekens, Twan; Ferrein, Alexander; Schiffer, Stefan: A Decentralised System Approach for Controlling AGVs with ROS. In: *2017 IEEE AFRICON*, 2017. – ISSN 2153–0033, S. 1436–1441

XFD+07. Xhafa, Fatos; Fernandez, Raul; Daradoumis, Thanasis; Barolli, Leonard; Caballé, Santi: Improvement of JXTA Protocols for Supporting Reliable Distributed Applications in P2P Systems. In: Enokido, Tomoya (Hrsg.); Barolli, Leonard (Hrsg.); Takizawa, Makoto (Hrsg.): *Network-Based Information Systems*. Berlin, Heidelberg:

Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–74573–0, S. 345–354

XZL+20. Xiao, Yang; Zhang, Ning; Lou, Wenjing; Hou, Y. T.: A Survey of Distributed Consensus Protocols for Blockchain Networks. In: *IEEE Communications Surveys Tutorials* 22 (Secondquarter 2020), Nr. 2, S. 1432–1465. – ISSN 1553–877X

YAH+17. Yaqoob, Ibrar; Ahmed, Ejaz; Hashem, Ibrahim Abaker T.; Ahmed, Abdelmuttlib Ibrahim A.; Gani, Abdullah; Imran, Muhammad; Guizani, Mohsen: Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges. In: *IEEE Wireless Communications* 24 (2017), Juni, Nr. 3, S. 10–16. – ISSN 1558–0687

Yül18. Yülek, Murat A.: *How Nations Succeed: Manufacturing, Trade, Industrial Policy, and Economic Development*. 1st ed. 2018. Singapore: Palgrave Macmillan Singapore, 2018. – ISBN 9789811305689

YV. Yu, Haifeng; Vahdat, Amin: Design and Evaluation of a Continuous Consistency Model for Replicated Services.

Zim80. Zimmermann, H.: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. In: *IEEE Transactions on Communications* 28 (1980), April, Nr. 4, S. 425–432. – ISSN 1558–0857

ZIN+08. Ziparo, V. A.; Iocchi, L.; Nardi, D.; Palamara, P. F.; Costelha, H.: Petri Net Plans: A Formal Model for Representation and Execution of Multi-Robot Plans. In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, Mai 2008 (AAMAS '08). – ISBN 978–0–9817381–0–9, S. 79–86

ZKC+15. Zhong, Zhenzhe; Kulkarni, Parag; Cao, Fengming; Fan, Zhong; Armour, Simon: Issues and Challenges in Dense WiFi Networks. In: *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2015. – ISSN 2376–6506, S. 947–951

ZLZ15. Zhou, Keliang; Liu, Taigang; Zhou, Lifeng: Industry 4.0: Towards Future Industrial Opportunities and Challenges. In: *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2015, S. 2147–2152

ZPG+14. Zamfirescu, Constantin-Bala; Pirvu, Bogdan-Constantin; Gorecky, Dominic; Chakravarthy, Harish: Human-Centred Assembly: A Case Study for an Anthropocentric Cyber-physical System. In: *Procedia Technology* 15 (2014), Januar, S. 90–98. – ISSN 2212–0173

ZPS+13. Zamfirescu, Constantin-Bala; Pârvu, Bogdan-Constantin; Schlick, Jochen; Zühlke, Detlef: Preliminary Insides for an Anthropocentric Cyber-physical Reference Architecture of the Smart Factory. In: *Studies in Informatics and Control* 22 (2013), September, Nr. 3. – ISSN 12201766, 1841429X

Anhang

Anhang A
Studentische Arbeiten

In Tabelle A.1 sind jene studentischen Arbeiten aufgeführt, welche im Zusammenhang mit den Themenfeldern dieser Veröffentlichung stehen und welche zugleich von dem Autor betreut wurden.

Tabelle A.1: Übersicht der betreuten studentischen Arbeiten

Titel	**Typ**	**Autor:in**
Agenten-Ausfallfehlererkennung - Implementierung einer Anwendung zur Verfügbarkeitsüberwachung von Agenten innerhalb der FIWARE Plattform	BA	Pose
Simulation und Evaluation von Failure Detection Algorithmen innerhalb eines strukturierten P2P Overlay Networks	BA	Gödeke
Optimierung der Positionsfindung in null-balancierten Baumstrukturen für Peer-to-Peer-Netzwerke	BA	Laskowski
Digitalisierung im Kontext der Industrie 4.0: Eine quantitative Erhebung	MA	Busch
Entwurf und Entwicklung einer domänenspezifischen Sprache zur Beschreibung von Produktionsaufträgen in Cyber-Physischen Produktionssystemen	BA	Hörstrup
Einfluss der Netzwerktopologie auf die Kommunikation am Beispiel der dezentralen Pfadplanung für Fahrerlose Transportfahrzeuge	BA	Stolz
Scalable Decentralized Task Allocation for Cyber-Physical Production Systems	MA	Gödeke
Effizienter Broadcast in einem null-balancierten baumstrukturierten Peer-to-Peer Netzwerk	BA	Tönning

BA: Bachelorarbeit; MA: Masterarbeit

Anhang B
Detailergebnisse der Umfrage

Tabelle B.1: Anzahl der Entitäten bestehend aus Anzahl der Mitarbeiter, der Maschinen und der Fahrerlosen Transportfahrzeuge, basierend auf [Bus21]

	Unternehmensgröße		
	klein	**mittel**	**groß**
Datensätze	7	24	31
Statistischer Wert			
Median	33	150	375
mittelwert	37,14	171,50	1495,26
Schiefe	0,66	1,08	5,45
Min	4	64	56
Max	75	430	30020
0,25-Quantil	20	93	258
0,75-Quantil	55	222	633

Tabelle B.2: Daten der Online-Umfrage. Anzahl an Datensätzen für die Unternehmensgröße: klein = 7, mittel = 24, groß = 30. Basierend auf [Bus21].

Unternehmensgröße	Median	mittelwert	Schiefe	Min	Max	0,25-Quantil	0,75-Quantil
Mitarbeiter insgesamt							
klein	21,00	22,29	0,90	3,00	50,00	13,50	27,50
mittel	112,50	118,96	0,76	44,00	250,00	65,75	150,00
groß	350,00	1.089,84	5,41	50,00	20.000,00	215,00	575,00
... davon in der Produktion und Logistik							
klein	11,00	14,57	1,32	3,00	40,00	4,75	22,50
mittel	50,00	68,79	0,89	12,00	170,00	30,00	95,00
groß	200,00	803,61	5,49	8,00	17.000,00	90,00	335,00
Produktanzahl							
klein	20,00	89,00	1,30	3,00	300,00	10,00	140,00
mittel	125,00	411,33	1,15	1,00	1.500,00	18,75	625,00
groß	100,00	426,16	2,32	1,00	3.000,00	20,00	250,00
Variantenanzahl							
klein	2,00	7,14	2,63	1,00	40,00	1,00	2,50
mittel	5,00	391.325,09	4,80	2,00	9.000.000,00	2,00	32,50
groß	5,00	139,87	5,45	1,00	3.200,00	3,50	50,00
Produktionsanlagen							
klein	7,00	14,86	1,73	1,00	50,00	4,50	18,50
mittel	30,00	54,26	1,97	5,00	250,00	16,50	65,00
groß	50,00	417,00	5,42	4,00	10.000,00	20,75	100,00

Tabelle wird fortgesetzt auf der nächsten Seite

Tabelle B.2 – Forsetzung von der vorherigen Seite

Unternehmensgröße	Median	mittelwert	Schiefe	Min	Max	0,25-Quantil	0,75-Quantil
Fertigungsschritte							
klein	4,00	6,14	2,54	1,00	25,00	2,50	4,00
mittel	5,00	14,21	4,63	1,00	200,00	2,75	6,00
groß	7,50	30,38	4,86	1,00	500,00	3,25	13,75
Statische Lager							
klein	3,00	2,71	0,26	1,00	5,00	1,50	3,50
mittel	4,00	1.262,75	4,90	1,00	30.000,00	3,00	7,75
groß	4,00	174,60	3,91	1,00	3.000,00	2,00	9,25
Produktionslager							
klein	1,00	2,71	2,37	1,00	10,00	1,00	2,50
mittel	4,00	132,21	4,90	1,00	3.000,00	2,00	10,00
groß	7,50	67,70	4,20	1,00	1.000,00	2,00	30,00
Nicht-motorisierte Fahrzeuge							
klein	3,00	4,29	1,67	2,00	10,00	2,50	5,00
mittel	10,00	32,57	4,77	500,00	50,00	7,00	15,00
groß	15,00	69,52	5,36	4,00	1.500,00	10,00	30,00
Motorisierte personengebundene Fahrzeuge							
klein	2,50	2,66	0,89	1,00	5,00	2,00	3,00
mittel	5,00	5,83	1,69	2,00	20,00	2,00	6,25
groß	10,00	31,20	5,14	1,00	500,00	5,25	20,00

Tabelle wird fortgesetzt auf der nächsten Seite

Tabelle B.2 – Forsetzung von der vorherigen Seite

Unternehmensgröße	Median	mittelwert	Schiefe	Min	Max	0,25-Quantil	0,75-Quantil
Fahrerlose Transportfahrzeuge							
klein				- - -			
mittel	2,00	3,25	1,87	1,00	8,00	1,75	3,50
groß	5,00	7,25	1,43	1,00	20,00	4,25	10,00

Anhang C
ANTLR-Spezifikation der MFDL

Auflistung C.1: ANTLR-4-Spezifikation der Material Flow Description Language

```
// LEXER
lexer grammar MFDLLexer;

tokens { INDENT, DEDENT }

@lexer::header{
from antlr_denter.DenterHelper import DenterHelper
from MFDLParser import MFDLParser
}
@lexer::members {
class MFDLDenter(DenterHelper):
    def __init__(self, lexer, nl_token, indent_token, dedent_token,
     ignore_eof):
        super().__init__(nl_token, indent_token, dedent_token, ignore_eof)
        self.lexer: MFDLLexer = lexer

    def pull_token(self):
        return super(MFDLLexer, self.lexer).nextToken()

denter = None

def nextToken(self):
    if not self.denter:
        self.denter = self.MFDLDenter(self, self.NL, MFDLLexer.INDENT,
     MFDLLexer.DEDENT, ignore_eof=False)
    return self.denter.next_token()
}

STRUCT: 'Struct';
RULE: 'Rule';
MODULE: 'Module';
IMPORT: 'Import';
TASK: 'Task';
TRANSPORT: 'Transport';
MOVE: 'Move';
ACTION: 'Action';
FROM: 'From';
TO: 'To';
DO: 'Do';
ON_DONE: 'OnDone';
END: 'End';
REPEAT: 'Repeat';
TRANSPORT_ORDER_STEP: 'TransportOrderStep';
MOVE_ORDER_STEP: 'MoveOrderStep';
```

```
43 ACTION_ORDER_STEP: 'ActionOrderStep';
44 CONSTRAINTS: 'Constraints';
45 PARAMETERS: 'Parameters';
46 STARTED_BY: 'StartedBy';
47 FINISHED_BY: 'FinishedBy';
48 LOCATION: 'Location';
49 EVENT: 'Event';
50 TIME: 'Time';
51 CONSTRAINT: 'Constraint';
52 NUMBER_P: 'number';
53 STRING_P: 'string';
54 BOOLEAN_P: 'boolean';
55 TRUE: 'True';
56 FALSE: 'False';
57
58 COLON: ':';
59 DOT: '.';
60 COMMA: ',';
61 JSON_OPEN: '{' -> pushMode(JSON);
62
63 QUOTE: '"';
64
65 ARRAY_LEFT: '[';
66 ARRAY_RIGHT: ']';
67
68 COMMENT : '#'~[\n]+  -> skip;
69 WHITESPACE: [ \t]+ -> skip;
70 NL: ('\r'? '\n' ' '*);
71
72 LEFT_PARENTHESIS: '(';
73 RIGHT_PARENTHESIS: ')';
74 LESS_THAN: '<';
75 LESS_THAN_OR_EQUAL: '<=';
76 GREATER_THAN: '>';
77 GREATER_THAN_OR_EQUAL: '>=';
78 EQUAL: '=';
79 EQUAL_B: '==';
80 NOT_EQUAL_B: '!=';
81 PLUS: '+';
82 MINUS: '-';
83 SLASH: '/';
84 STAR: '*';
85
86 BOOLEAN_AND: 'And';
87 BOOLEAN_OR: 'Or';
88 BOOLEAN_NOT: '!';
89
90 INTEGER: [0-9]+;
91 FLOAT: INTEGER '.'INTEGER;
92
93 STRING: '"' ('\\"'|.)*? '"';
94 ID: [a-zA-Z_][a-zA-Z0-9_]*;
95
96 // JSON sub grammar
97 mode JSON;
98
99 JSON_STRING: '"' ('\\"'|.)*? '"';
100
101 JSON_COLON: ':';
102 JSON_QUOTE: '"';
103
104 JSON_COMMENT : '#' ~[\n]+  -> skip;
105
106 JSON_ARRAY_LEFT: '[';
107 JSON_ARRAY_RIGHT: ']';
108
109 JSON_COMMA: ',';
```

```
110
111 NUMBER
112    : '-'? INT ('.' [0-9] +)? EXP?
113    ;
114
115 fragment INT
116    : '0' | [1-9] [0-9]*
117    ;
118
119 // no leading zeros
120
121 fragment EXP
122    : [Ee] [+\-]? INT
123    ;
124
125 WS
126    : [ \t\n\r] + -> skip
127    ;
128
129 JSON_OPEN_2: '{' -> pushMode(JSON);
130 JSON_CLOSE: '}' -> popMode;
131
132 // GRAMMAR
133
134 parser grammar MFDLParser;
135
136 options {
137     tokenVocab = MFDLLexer;
138 }
139
140 program:
141     import_stmt* (struct | instance | rule_ | module | task | orderStep |
       NL)* EOF;
142
143 import_stmt:
144     IMPORT  module_path (COMMA module_path)* NL;
145
146 module_path:
147     ID (DOT ID)*;
148
149 struct:
150     STRUCT struct_id (COLON struct_id)? INDENT attribute_definition+ DEDENT
        END;
151
152 struct_id:
153     LOCATION | EVENT | TIME | CONSTRAINT | ID;
154
155 instance:
156     struct_id ID INDENT (attribute_assignment NL)+ DEDENT END;
157
158 rule_:
159     RULE rule_call INDENT (expression NL)+ DEDENT END;
160
161 rule_call:
162     ID LEFT_PARENTHESIS (rule_parameter (COMMA rule_parameter)*)?
       RIGHT_PARENTHESIS;
163
164 rule_parameter:
165     (ID | value) (EQUAL value)?;
166
167 module:
168     MODULE ID INDENT (struct | instance | rule_ | task | transportOrderStep
        | NL)* DEDENT END;
169
170 task:
171     TASK ID INDENT taskStatement+ DEDENT END;
172
```

```
173 taskStatement:
174     transportStatement | moveStatement | actionStatement | optStatement |
         repeatStatement | constraintStatement;
175
176 transportStatement:
177     TRANSPORT NL fromStatement toStatement;
178
179 fromStatement:
180     FROM ID (COMMA ID)* NL;
181
182 toStatement:
183     TO ID NL;
184
185 moveStatement:
186     MOVE NL TO ID NL;
187
188 actionStatement:
189     ACTION NL DO ID NL;
190
191 constraintStatement:
192     CONSTRAINTS COLON (expression | json_object) NL;
193
194 repeatStatement:
195     REPEAT COLON INTEGER NL;
196
197 orderStep:
198     transportOrderStep | actionOrderStep;
199
200 transportOrderStep:
201     TRANSPORT_ORDER_STEP ID INDENT tosStatement+ DEDENT END;
202
203 tosStatement:
204     locationStatement | parameterStatement | optStatement;
205
206 locationStatement:
207     LOCATION COLON ID NL;
208
209 parameterStatement:
210     PARAMETERS COLON (value | json_object) NL;
211
212 optStatement:
213     eventStatement | onDoneStatement;
214
215 eventStatement:
216     STARTED_BY COLON expression NL
217     | FINISHED_BY COLON expression NL;
218
219 onDoneStatement:
220     ON_DONE COLON ID (COMMA ID)* NL;
221
222 actionOrderStep:
223     ACTION_ORDER_STEP ID INDENT aosStatement+ DEDENT END;
224
225 aosStatement:
226     parameterStatement | optStatement;
227
228 attribute_definition:
229     ID COLON primitive NL;
230
231 attribute_assignment:
232     attribute_access COLON (value | json_object);
233
234 attribute_access:
235     ID (DOT ID)*;
236
237 primitive:
```

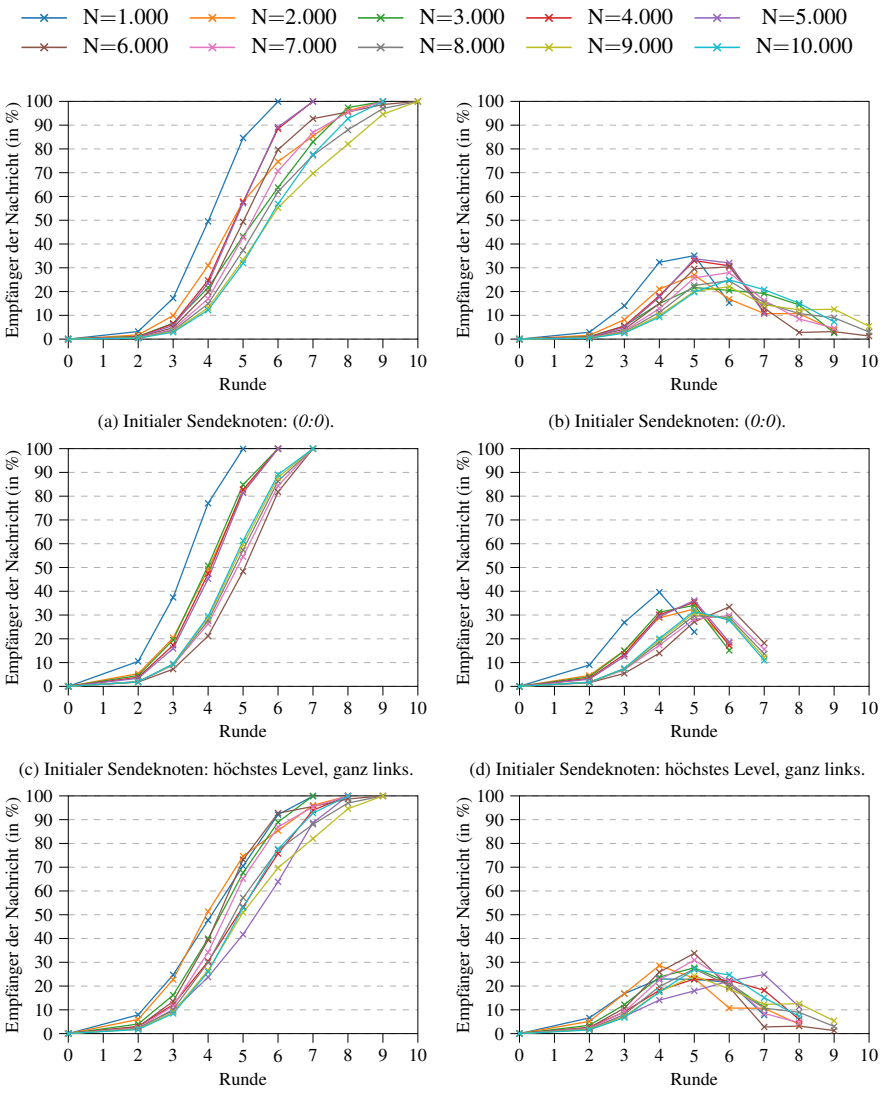
```
238     NUMBER_P | STRING_P | BOOLEAN_P | LOCATION | EVENT | TIME | CONSTRAINT
        | ID;
239
240 value:
241     TRUE | FALSE | INTEGER | FLOAT | STRING;
242
243 expression:
244     LEFT_PARENTHESIS expression RIGHT_PARENTHESIS
245     | unOperation expression
246     | expression binOperation expression
247     | expression BOOLEAN_AND expression
248     | expression BOOLEAN_OR expression
249     | attribute_access
250     | value
251     | rule_call;
252
253 binOperation:
254     LESS_THAN | LESS_THAN_OR_EQUAL | GREATER_THAN | GREATER_THAN_OR_EQUAL |
        EQUAL_B | NOT_EQUAL_B | STAR | SLASH | MINUS | PLUS;
255
256 unOperation:
257     BOOLEAN_NOT;
258
259 json_object:
260     json_open_bracket pair (JSON_COMMA pair)* JSON_CLOSE
261     | json_open_bracket JSON_CLOSE;
262
263 pair:
264     JSON_STRING JSON_COLON json_value;
265
266 json_open_bracket:
267     JSON_OPEN | JSON_OPEN_2;
268
269 json_value:
270     JSON_STRING | TRUE | FALSE | NUMBER | json_object | json_array;
271
272 json_array:
273     JSON_ARRAY_LEFT json_value (JSON_COMMA json_value)* JSON_ARRAY_RIGHT
274     | JSON_ARRAY_LEFT JSON_ARRAY_RIGHT;
```

Anhang D
Kumulierte prozentuale Ausbreitung einer Nachricht in MINHTON

Tabelle D.1: Level und Nummer für Knoten auf dem höchsten Level ganz links und auf dem zweithöchsten Level ganz rechts, in Abhängigkeit von der Netzwerkgröße N und dem Fanout m

	$m = 2$		$m = 4$		$m = 8$		$m = 16$	
N	HLGL	ZLGR	HLGL	ZLGR	HLGL	ZLGR	HLGL	ZLGR
1.000	(9:0)	(8:255)	(5:0)	(4:255)	(4:0)	(3:511)	(3:0)	(2:255)
2.000	(10:0)	(9:511)	(6:0)	(5:1023)	(4:0)	(3:511)	(3:0)	(2:255)
3.000	(11:0)	(10:1023)	(6:0)	(5:1023)	(4:0)	(3:511)	(3:0)	(2:255)
4.000	(11:0)	(10:1023)	(6:0)	(5:1023)	(4:0)	(3:511)	(3:0)	(2:255)
5.000	(12:0)	(11:2047)	(6:0)	(5:1023)	(5:0)	(4:4095)	(4:0)	(3:4095)
6.000	(12:0)	(11:2047)	(7:0)	(6:4095)	(5:0)	(4:4095)	(4:0)	(3:4095)
7.000	(12:0)	(11:2047)	(7:0)	(6:4095)	(5:0)	(4:4095)	(4:0)	(3:4095)
8.000	(12:0)	(11:2047)	(7:0)	(6:4095)	(5:0)	(4:4095)	(4:0)	(3:4095)
9.000	(13:0)	(12:4095)	(7:0)	(6:4095)	(5:0)	(4:4095)	(4:0)	(3:4095)
10.000	(13:0)	(12:4095)	(7:0)	(6:4095)	(5:0)	(4:4095)	(4:0)	(3:4095)

HLGL: höchstes Level, ganz links; ZLGR: zweithöchstes Level, ganz rechts



(a) Initialer Sendeknoten: (*0:0*).

(b) Initialer Sendeknoten: (*0:0*).

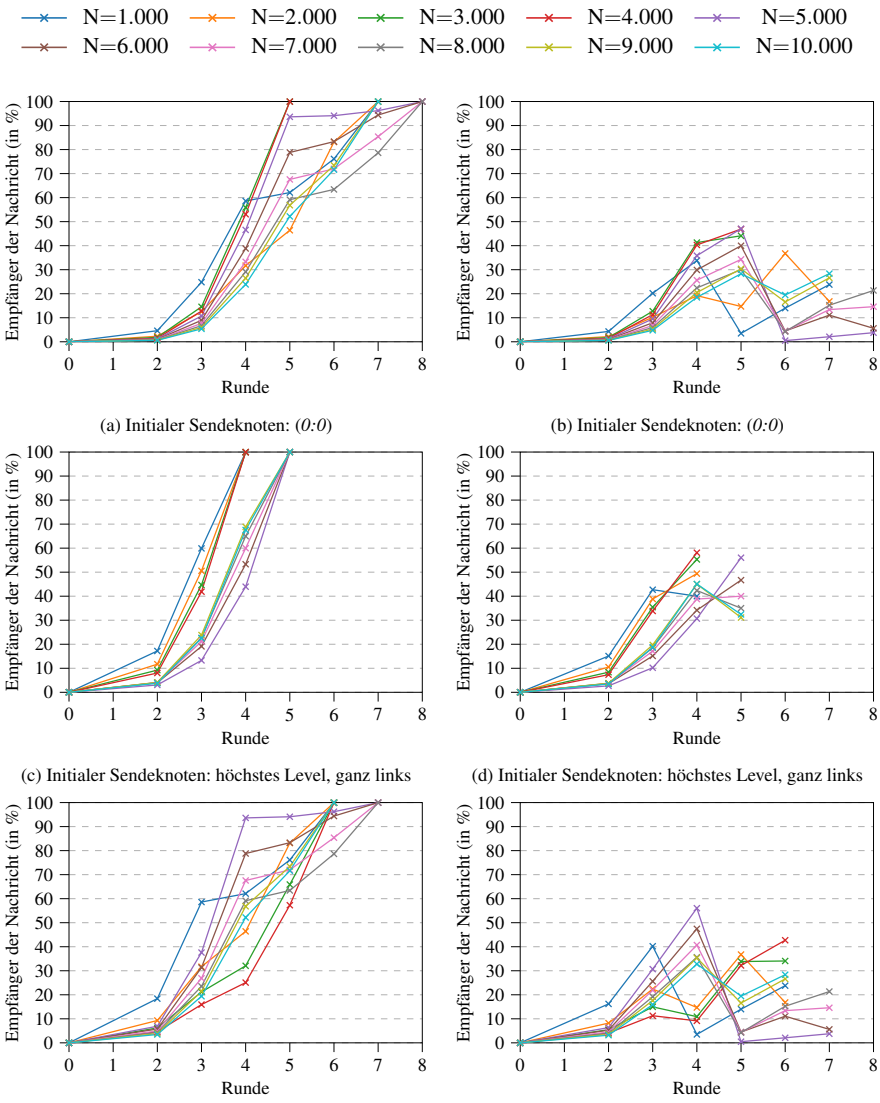
(c) Initialer Sendeknoten: höchstes Level, ganz links.

(d) Initialer Sendeknoten: höchstes Level, ganz links.

(e) Initialer Sendeknoten: zweithöchstes Level, ganz rechts.

(f) Initialer Sendeknoten: zweithöchstes Level, ganz rechts.

Abb. D.1: Kumulierte prozentuale Ausbreitung einer Nachricht bis zur Runde (a, c und e) bzw. prozentuale Ausbreitung einer Nachricht je Runde (b, d und f) in Abhängigkeit von der Netzwerkgröße N, dem Fanout $m = 4$ und der Baumhöhe h für unterschiedliche initiale Sendeknoten.



(a) Initialer Sendeknoten: (*0:0*)

(b) Initialer Sendeknoten: (*0:0*)

(c) Initialer Sendeknoten: höchstes Level, ganz links

(d) Initialer Sendeknoten: höchstes Level, ganz links

(e) Initialer Sendeknoten: zweithöchstes Level, ganz rechts

(f) Initialer Sendeknoten: zweithöchstes Level, ganz rechts

Abb. D.2: Kumulierte prozentuale Ausbreitung einer Nachricht bis zur Runde (a, c und e) bzw. prozentuale Ausbreitung einer Nachricht je Runde (b, d und f) in Abhängigkeit von der Netzwerkgröße N, dem Fanout $m = 8$ und der Baumhöhe h für unterschiedliche initiale Sendeknoten

Anhang E
Zustände der Interaktion

Auflistung E.1: Statusänderungen in der Interaktion

```
enum MessageTypes : uint8_t {
  task_update = 6,            // (1:1)
  call_for_proposal = 7,      // (m:n) Publish-Subscribe
  submission = 8,             // (m:n) Publish-Subscribe
  iteration_notification = 9, // (m:n) Publish-Subscribe
  winner_notification = 10,   // (1:1)
  winner_response = 11,       // (1:1)
  k_unknown_negotiation_msg_type = 100,
};
```

Die Folgenden Zustände (vgl. Auflistung E.2) werden in einer (1:1)-Kommunikation zwischen dem logischen FTF-Agenten und dem logischen MF-Agenten ausgetauscht.

Auflistung E.2: Zustände eines Transportauftrags

```
enum OrderStates : uint8_t{
  Created = 0,
  Queued = 1,
  Started = 2,
  GoToPickUpLocation = 3,
  ReachedPickUpLocation = 4,
  Load = 5,
  Loaded = 6,
  GoToDeliveryLocation = 7,
  ReachedDeliveryLocation = 8,
  Unload = 9,
  Unloaded = 10,
  Finished = 11,
  Error = 12,
};
```