Band 409

Verlagsschriftenreihe des Heinz Nixdorf Instituts
Prof. Dr.-Ing. Iris Gräßler (Hrsg.)
Produktentstehung

Alexander Pöhler

Automatisierte dezentrale Produktionssteuerung für cyber-physische Produktionssysteme mit digitaler Repräsentation der Beschäftigten

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Alexander Pöhler

Automatisierte dezentrale Produktionssteuerung für cyber-physische Produktionssysteme mit digitaler Repräsentation der Beschäftigten

Automated decentralized production control for cyber-physical production systems with digital representation of the employees

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Band 409 der Verlagsschriftenreihe des Heinz Nixdorf Instituts

ISSN (Online): 2365-4422
ISBN: 978-3-947647-28-6

Satz und Gestaltung: Alexander Pöhler

Automatisierte dezentrale Produktionssteuerung für cyber-physische Produktionssysteme mit digitaler Repräsentation der Beschäftigten

zur Erlangung des akademischen Grades eines
DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)
der Fakultät Maschinenbau
der Universität Paderborn

genehmigte
DISSERTATION

von
M.Sc. Alexander Pöhler
Paderborn

Tag des Kolloquiums:	08.03.2022
Referent:	Prof. Dr.-Ing. Iris Gräßler
Korreferent:	Prof. Dr. Gregor Engels

Zusammenfassung

In dieser Arbeit wird die Entwicklung einer dezentralen Produktionssteuerung mit digitaler Repräsentation zur Anwendung in cyber-physischen Produktionssystemen (CPPS) beschrieben. Die dezentrale Produktionssteuerung geschieht hierbei selbstorganisiert, da sie die Einlastung und Umplanung von Aufträgen sowie die Koordination zwischen den einzelnen Produktionselementen selbstständig übernimmt. Dies erlaubt die automatisierte Steuerung von CPPS. Dabei spielen die Beschäftigten in dieser Betrachtung eine wesentliche Rolle. Ihre Bedürfnisse und Wünsche werden von dem Steuerungssystem berücksichtigt. Dafür wurde das Konzept der digitalen Repräsentation von Beschäftigten entwickelt. Diese Repräsentation erlaubt den individuellen Eingriff von Beschäftigten bei Entscheidungen von dem Steuerungssystem.

Das automatisierte dezentrale Steuerungssystem wurde in einer Laborumgebung, die zu einem CPPS umgebaut wurde, implementiert. Darauf basierend wurde eine zweistufige Validierung durchgeführt. Ein Teil der Validierung findet durch eine reale Umsetzung in einem Labor statt, bei der ein Vergleich zu dem vorher im Labor eingesetzten zentralen Produktionssteuerungssystem gezogen wird. Die zweite Validierung erfolgt mit Hilfe einer Simulationsumgebung. Dabei konnte die Implementierung der Selbstorganisation und eine Steigerung der Liefertreue der dezentralen Produktionssteuerung gegenüber heutzutage üblicherweise eingesetzten Verfahren nachgewiesen werden.

Abstract

This thesis describes the development of a decentralized production control system with digital representation for the automation of cyber-physical production systems (CPPS). The decentralized production control handles the dispatching and rescheduling of orders as well as the self-organized coordination among the production elements. This enables self-organized control of CPPS. In this process, the employees play an essential role in the consideration. Their needs and wishes are taken into account by the control system. For this purpose, the concept of digital representation of employees was developed. This realizes the possibility for the employees to intervene in decisions of the control system.

The automated decentralized production control system was implemented in a laboratory environment, which was converted to a CPPS. Based on this implementation, a two-stage validation was performed. One part of the validation takes place through a real implementation in the laboratory, where a comparison is made with the previously used centralized production control system. The second validation takes place with the help of a simulation environment. Thereby the implementation of the self-organization and an increase of the delivery reliability could be proven compared to procedures usually used today.

Liste der Vorveröffentlichungen

2019

- Gräßler, Iris; Pöhler, Alexander: Human-centric design of cyber-physical production systems. In: Putnik, Goran D. (Hrsg.) Procedia CIRP - Proceedings of the 29th CIRP Design Conference, Nr. 84, S. 251-256, ISSN: 2212-8271, PROCIR-D-19-00344R1, DOI: https://doi.org/10.1016/j.procir.2019.04.199, Póvoa de Varzim, 8. - 10. Mai 2019 CIRP, Elsevier B.V.

- Gräßler, Iris; Pöhler, Alexander: Simulation von Cyber-Physischen Produktionssystemen in einer Forschungsinfrastruktur. In: Bertram, Torsten; Corves, Burkhard; Gräßler, Iris; Janschek, Klaus (Hrsg.) Fachtagung Mechatronik 2019 Paderborn, S. 149-154, 27. - 28. Mrz. 2019

2018

- Gräßler, Iris; Hentze, Julian; Pöhler, Alexander: Self-organizing production systems: Implications for product design. In: 12th CIRP Conference on Intelligent Computation in Manufacturin Engineering, Nr. 79, S. 546-550, Gulf of Naples, 18. - 20. Jul. 2018, Procedia CIRP

- Gräßler, Iris; Pöhler, Alexander: Intelligent control of an assembly station by integration of a digital twin for employees into the decentralized control system. In: Procedia Manufacturing - Proceedings of 4th International Conference on System-integrated Intelligence Intelligent, flexible and connected systems in products and production (SysInt), 19. - 20. Jun. 2018, Elsevier B.V.

- Gräßler, Iris; Pöhler, Alexander: Produktentstehung im Zeitalter von Industrie 4.0. In: Maier, Günter; Engels, Gregor; Steffen, Eckhard (Hrsg.) Handbuch Gestaltung digitaler und vernetzter Arbeitswelten, Kapitel: Produktentstehung im Zeitalter von Industrie 4.0, S. 1-21. Springer-Verlag Berlin Heidelberg, https://doi.org/10.1007/978-3-662-52903-4_23-1, Jan. 2018

- Fischer, Christoph; Pöhler, Alexander: Supporting the Change to Digitalized Production Environments Through Learning Organization Development. In: Harteis, Christian (Hrsg.) The Impact of Digitalization in the Workplace - An Educational View, Kapitel: Supporting the Change to Digitalized Production Environments Through Learning Organization Development, S. 141-160. Springer International Publishing

2017

- Gräßler, Iris; Pöhler, Alexander: Integration of a digital twin as human representation in a scheduling procedure of a cyber-physical production system. In: Proceedings of 2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM2017), Singapore, 10. - 13. Dez. 2017

- Gräßler, Iris; Pöhler, Alexander: Intelligent devices in a decentralized production system concept. In: Proceedings of 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering, Ischia, 19. - 21. Jul. 2017, Procedia CIRP

- Gräßler, Iris; Pöhler, Alexander; Hentze, Julian: Decoupling of product and production development in flexible production environments. In: Complex Systems Engineering and Development Proceedings of the 27th CIRP Design Conference Cranfield University, UK, S. 548-553, 10. - 12. Mai 2017, Procedia CIRP

- Gräßler, Iris; Pöhler, Alexander: Implementation of an adapted holonic production architecture. In: Proceedings of the 50th CIRP conference on manufacturing systems (CIRP-CMS), 2. - 5. Mai 2017, Procedia CIRP

2016

- Gräßler, Iris; Pöhler, Alexander; Pottebaum, Jens: Creation of a Learning Factory for Cyber Physical Production Systems. Proceedings of the 6th conference on learning factories, 29. - 30. Jun. 2016, Procedia CIRP

2015

- Gräßler, Iris; Pöhler, Alexander; Scholle, Philipp: CPPS - Based Market Access Opportunities for Production Capacity Providers. In: Padoano, Elio; Villmer, Franz-Josef (Hrsg.) 5th International Conference Production Engineering and Management, S. 67-77, ISBN 978-3-941645-11-0, Okt. 2015

Inhaltsverzeichnis

Seite

1 Einleitung

Die heutige Produktion sieht sich durch die steigende Nachfrage nach individuellen Produkten [AR11, Grä04], sich verkürzenden Produkt- und Technologielebenszyklen [ALW06] und einem steigenden Wettbewerb [SWB+12] großen Veränderungen ausgesetzt. Die Kombination aus der Produktion individueller Produkte unter wettbewerbsfähigen Kosten in kürzeren Produkt- und Technologielebenszyklen erfordert eine hochdynamische, flexible Produktion [RZ03]. Flexibilität steht deswegen im Vordergrund zahlreicher Optimierungsbestrebungen in der Produktion [Pil06]. Die Umsetzung von Flexibilität und individueller Produkte resultiert in einer deutlichen Steigerung der Komplexität der Produktionsplanung und -steuerung. Diese Komplexitätssteigerung ergibt sich zudem durch die folgenden drei Faktoren: eine steigende Anzahl an zu produzierenden Varianten, die Bestrebungen nach kürzeren Durchlaufzeiten und höheren Auslastungen und gestiegenen Gestaltungsmöglichkeiten durch die Einführung von flexibel einsetzbaren Produktionsanlagen. Um die zunehmende Variantenvielfalt bei gleichbleibender oder steigender Produktivität zu gewährleisten, findet eine zunehmende Durchdringung von Software in die Produktionssysteme statt. Diese Software dient insbesondere der Automatisierung komplexer Planungs- und Steuerungsvorgänge [KWH13]. Cyber-physische Systeme (CPS) stellen hierbei die nächste Entwicklungsstufe der Integration von Softwaresystemen in technische Systeme hin zu vollständig softwaretechnisch integrierten und miteinander vernetzten Systemen dar [VDI20]. CPS bezeichnen dabei Systeme, die reale (physische) Objekte und Prozesse mit informationsverarbeitenden (virtuellen) Objekten und Prozessen über offene, teilweise globale und jederzeit miteinander verbundene Informationsnetze verknüpfen [VDI20]. Der Einsatz von CPS in der Produktion wird als cyber-physisches Produktionssystem (CPPS) bezeichnet. CPPS werden künftig zu einem grundlegenden Wandel der Planung und Steuerung in der Produktion führen, dessen Auswirkungen unter anderem durch den Begriff „Industrie 4.0“ ausgedrückt werden [Wah11].

Industrie 4.0 steht für die vierte industrielle Revolution. Aufbauend auf der Einführung von CPPS und der damit einhergehenden Vernetzung aller Produktionsmittel, wird im Rahmen von Industrie 4.0 eine disruptive technologische Entwicklung der Produktion in Gang gesetzt [Rei17]. Der Begriff „Industrie 4.0“ wurde im Rahmen der Hannover Messe 2011 durch Kagermann und Wahlster geprägt und soll die Tragweite der aktuellen Entwicklungen in Produkt- und Produktionstechnologien veranschaulichen [KLW11]. Er fasst bisherige technologische Entwicklungen in der Produktion seit dem 18. Jahrhundert in drei Entwicklungsstufen (siehe Abbildung 1) zusammen, die durch disruptive Technologien ausgelöst wurden. Die erste Stufe der industriellen Revolution wurde durch die Einführung der Dampfmaschine und damit die Mechanisierung der menschlichen und tierischen Arbeitskraft bzw. Luft- und Wasserkraft initiiert. Die zweite Revolutionsstufe bildet die Einführung der Massenfertigung durch die Unterteilung von Arbeitsoperationen in kleinste Schritte ab Ende des 20. Jahrhunderts. Zwischen 1950 und

1960 fand durch die Einführung von Speicherprogrammierbaren Steuerungen (SPS) zur Steuerung von Produktionsmaschinen die dritte Revolution und damit eine fortschreitende Automatisierung auch für flexible Produktionsprozesse statt. Es wird nun davon ausgegangen, dass die Einführung von CPS in der Produktion zu einem ähnlichen Technologieschub führen wird und ein neues Zeitalter der Produktion, der Industrie 4.0, einleiten wird.

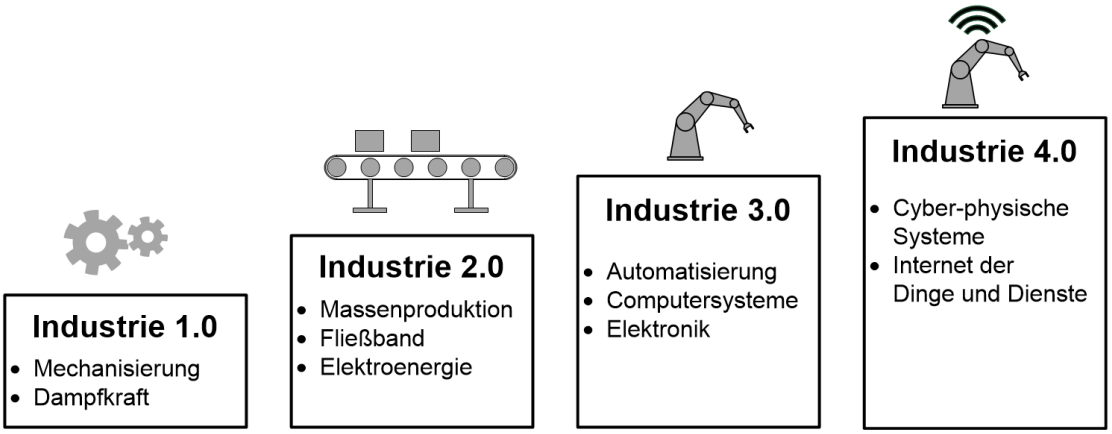

Abbildung 1: Die industriellen Revolutionen bis zur Industrie 4.0 (in Anlehnung an [Aca12])

Die im Rahmen von Industrie 4.0 erfolgende disruptive technologische Entwicklung wird weitreichende Auswirkungen auf zahlreiche Aspekte industrieller Produktionsunternehmen haben [Aca12]. Durch Industrie 4.0 werden über die Produktion hinausgehend sich Innovationen in Geschäftsmodellen [BGP+16], in der Sicherheit von Anlagen und in Produkten ergeben. In Bezug auf die Produktion fasst Acatech [Aca12] die Auswirkungen auf die Begriffe der vertikalen und horizontalen Integration zusammen. Durch die Vernetzung von Maschinen, Betriebsmitteln, Werkstücken sowie Lager- und Transportsystemen über das Internet der Dinge und Dienste kann sich die Produktion dezentral koordinieren und bei Bedarf flexibel rekonfigurieren. Diese unternehmensweite Vernetzung von Produktionsmitteln über mehrere Hierarchieebenen hinweg wird als vertikale Integration beschrieben. Gleichzeitig ermöglicht die unternehmensübergreifende Vernetzung von Geschäfts- und Produktionsprozessen eine durchgängige Produktentstehung [Grä15, GHB18, GH20] und die Bildung von Wertschöpfungsnetzwerken. Diese unternehmensübergreifende Vernetzung wird als horizontale Integration bezeichnet. [SG16]

Die Rolle der Beschäftigten erfährt durch die Einführung von CPPS ebenso einen grundlegenden Wandel [JRJ+13]. Die durchgängig vernetzten und dynamischen Abläufe verändern Arbeitsprozesse grundlegend und ermöglichen eine Flexibilisierung der Beschäftigung. Das Produktionsplanungs- und Steuerungssystem (PPS) stellt dabei einen Schlüsselfaktor zur Erreichung flexibler Arbeitsprozesse dar. Bei den Forschungsarbeiten im Kontext von Industrie 4.0 werden bisher selten Auswirkungen auf Beschäftigte

untersucht. Die Personalplanung betrachtet zudem den Menschen bisher als reine Ressource. Die Einführung von modernen Informations- und Kommunikationstechnologien (IKT) in der Produktion eröffnet die Veränderung hin zu menschenzentrierten Systemen, die die Anforderungen und Bedürfnisse von Beschäftigten berücksichtigen. Auch wenn es seit der Einführung der Fließbandfertigung zahlreiche Entwicklungen im Bereich der Gestaltung von Arbeitsplätzen und -prozessen gegeben hat, wurde die Arbeit in der Produktion nur zu einem geringen Grad erleichtert [Gri12]. Die Freiheitsgrade der Beschäftigten hinsichtlich Arbeitsweise, -geschwindigkeit und -pausen sind deutlich eingeschränkt [SLB09, Egb13]. Die Arbeit in der Produktion ist weiterhin von hoher Monotonie und einseitigen Belastungen ohne Tätigkeits- und Handlungsspielraum der eingesetzten Beschäftigten geprägt [Kra00, GBS13]. Die Neugestaltung von Produktionssystemen im Rahmen von Industrie 4.0 eröffnet die Möglichkeit, die darin ausgeführten Arbeits-, Planungs- und Steuerungsprozesse ebenso neu zu gestalten und damit den Arbeitsplatz von Produktionsbeschäftigten grundlegend aufzuwerten. [Bau13]

1.1 Problemstellung

Mit der Einführung von CPPS sind zwei wesentliche Potenziale gegenüber bisherigen Produktionssystemen verbunden [Aca12]:

- Eine selbstständige Konfiguration der Produktionssysteme zur eigenständigen Ausrichtung auf die Auftrags- und Umgebungssituation.
- Eine Steuerung zur Anpassung auf veränderte Umgebungsbedingungen durch eine kontinuierliche Überwachung der Umgebung.

Um beide Potenziale nutzen zu können, sind entsprechende Lösungen zur Umsetzung im Planungs- und Steuerungssystem erforderlich. Beide Potenziale – sowohl die selbstständige Konfiguration als auch die Anpassung an veränderte Umgebungsbedingungen – zielen auf eine Erhöhung der Flexibilität durch eine verbesserte Vorausschau und schnellere Reaktionsgeschwindigkeiten des Produktionssystems ab. Diese befinden sich im Aufgabenbereich der Produktionssteuerung. Die Produktionssteuerung ist dabei derzeitig systemseitig in die Planung in PPS-Systemen integriert. Die Aufgabe der Produktionssteuerung ist dabei im Wesentlichen die von der Produktionsplanung erstellten Produktionsreihenfolgepläne auszuführen, den Status zurückzumelden und gegebenenfalls eine Neuplanung anzustoßen [PMD+14]. Heutzutage eingesetzte PPS-Systeme werden dabei als starr und unflexibel wahrgenommen [CHM15]. Diese Inflexibilität stellt einen Gegensatz zu den Herausforderungen dar, die sich aus der Produktion individueller Produkte ergeben [SW06]. Es wird davon ausgegangen, dass sich die Produktionsplanung durch CPPS auf die Planung von Produktionsmengen und Fertigstellungsterminen konzentrieren wird [GSS+15, KMN15]. Dahingegen wird sich die Aufgabe der Produktionssteuerung in Richtung autonomer Entscheidungsfindung auf Basis grober Planvorgaben aus der

Produktionsplanung entwickeln. Dies betrifft die bisherigen klassischen Aufgabenbereiche der Produktionsplanung, wie z. B. die Reihenfolgebildung von Aufträgen. Eine automatisierte Produktionssteuerung, die individuelle Anforderungen komplexer Produktionsprozesse berücksichtigt und somit die Potenziale von CPPS ausschöpfen würde, existiert derzeit nicht [Gru17]. Die Einführung von CPPS bietet die Möglichkeit einer Umgestaltung der Produktionssteuerung hin zu automatisierten Steuerungssystemen [Löd13]. Gerade diese automatisierte Steuerung (Selbststeuerung) des Produktionssystems stellt für Unternehmen jedoch die technologisch größte Herausforderung zur Ausschöpfung der Potenziale von CPPS dar [REG+13].

Eine auf Selbststeuerung basierendes Produktionssteuerungsverfahren, die die Anforderungen der Produktion von individuellen Produkten berücksichtigt, ist derzeit im Stand der Forschung nicht gegeben. Insbesondere eine Übernahme von Aufgaben der Produktionsplanung, im Wesentlichen die automatisierte Umplanung des Produktionssystems, wird derzeit nicht betrachtet. Dies ist vor allem daher bemerkenswert, da erst die Berücksichtigung der Wechselwirkungen zwischen diesen Aufgaben eine sehr gute Produktionssteuerung ermöglicht und dazu führen kann diese zu automatisieren. Das dies bisher nicht betrachtet wurde, liegt laut [Gru17] an drei Faktoren. Erstens ist es Forschern möglich, wesentlich komplexere Verfahren zu entwickeln, falls einzelne Aufgaben isoliert betrachtet werden [Löd13]. Zweitens ist der Aufwand, insbesondere zur Modellierung und Simulation bei größerem Betrachtungsumfang, ebenfalls deutlich höher. Drittens ist der Aufgabenumfang der Fertigungssteuerung in den letzten Jahren gestiegen [Löd13], sodass der Bedarf einer integrierten Betrachtung der Aufgaben neu ist.

Die Entwicklung eines automatisierten Produktionssteuerungssystems bringt ebenso eine Veränderung der Rolle der Beschäftigten mit sich. Sobald die Produktionssteuerung dazu befähigt wird, kurzfristige Umplanungen zu realisieren, greift sie in die Planung ein und muss zum Beispiel den Personalplan oder die Materialplanung in die Umplanung mit einbeziehen. Dies bedeutet, dass die Produktionssteuerung automatisch Steuerungsentscheidungen treffen muss, die die Mitarbeiter direkt beeinflussen. Eine Betrachtung der Auswirkungen auf die Mitarbeiter von solchen automatisierten Steuerungssystemen wurde bisher nicht vorgenommen, da bisher immer menschliche Planer diesen Prozess übernommen haben [BLH16].

Die Neuentwicklung eines Produktionssteuerungssystems muss somit nicht nur die technischen Herausforderungen einer integrierten, automatisierten Lösung bewältigen, sondern ebenso eine formale Einbindung der Beschäftigten in die Steuerungsabläufe ermöglichen. Eine solche integrierte Lösung dient dazu, den Gedanken automatisierter Systeme auf die industrielle Produktion zu übertragen und gleichzeitig den Menschen in die Veränderung mit einzubeziehen.

1.2 Zielsetzung der Arbeit

Im Rahmen dieses Dissertationsvorhabens soll ein neues Verfahren der selbststeuernden Produktionssteuerung entwickelt und evaluiert werden. Diese soll die veränderte Rolle und das erweiterte Aufgabenspektrum der Produktionssteuerung in einem CPPS Produktionssystem berücksichtigen und realen Anwendungsfällen evaluiert werden.

Die grundlegenden Anforderungen an die zu entwickelnde Verfahren sind:

- Integrierte Betrachtung der Aufgaben der Produktionssteuerung: Auftragsfreigabe, Auftragsverteilugn, Reihenfolgebildung und Kapazitätssteuerung

- Berücksichtigung der Anforderungen der Produktion individueller Produkte in CPPS

- Betrachtung von Störungen während der Produktion

- Berücksichtigung der steigenden Bedeutung der Termintreue

- Orientierung an Planvorgaben der Produktionsplanung

Bezogen auf die Beschäftigten sollen für diese durch die Automatisierung der Produktionssteuerung keine Nachteile entstehen. Dies bedeutet, dass Ihnen durch die Abbildung in einem technischen System nicht die Möglichkeit entzogen werden soll Einfluss auf die Produktionssteuerung zu nehmen, also diese eigenständig zu manipulieren und Produktionsentscheidungen zu ändern. Zudem müssen die Wünsche und Bedürfnisse zum Beispiel bezogen auf die Auftrags- und Arbeitszuweisung von der Produktionssteuerung wie bisher berücksichtigt werden können.

Das Ziel dieser Arbeit liegt somit in der Entwicklung eines neuen Verfahrens zur automatisierten und dezentralen Produktionssteuerung für CPPS, das die Bedürfnisse und Anforderungen von Beschäftigten durch eine digitale Repräsentation während der Planung und Steuerung von Aufträgen berücksichtigt. Das dabei entstehende automatisierte Produktionssteuerungssystem soll ein erster Ansatz für eine vollumfänglich integrierte Lösung zur dezentralen bzw. flexiblen Steuerung von CPPS sein. Entsprechend der Charakteristika von CPPS soll ein Informationsaustausch über PPS-relevante Daten zwischen allen Produktionselementen des CPPS stattfinden und die selbstorganisierte Ausführung [GPH17] der Produktionspläne durch die einzelnen Elemente des Produktionssystems geschehen.

Es wird beabsichtigt, die Beschäftigten als aktive Elemente in das Steuerungssystem des CPPS einzubinden. Hierzu sollen notwendige Informationen über die Beschäftigten, sowie deren Fähigkeiten, Einsatzzeiten und Bedürfnisse wie Weiterbildung, Pausenzeiten formalisiert und im Rahmen der Entscheidungsfindung für den Personaleinsatz berücksichtigt werden. Umgekehrt sollen die Beschäftigten die Möglichkeit erhalten, sie betreffende maschinelle Entscheidungsvorgänge zu beeinflussen und bei Entscheidungen zu intervenieren. Diese Einbindung der Beschäftigten soll dabei weitestgehend

automatisiert geschehen, wozu ein gewisser Kenntnisstand (Datenbasis) des CPPS über die Beschäftigten erforderlich ist.

1.3 Spezifizierung des Betrachtungsbereichs

Diese Arbeit beschränkt sich auf die Untersuchung von Produktionssteuerungen für Produktionssysteme, die auf die Produktion von individuellen Produkten ausgelegt sind. Produktionssteuerungen – bzw. weiter betrachtet PPS – sind dem Produktions-management bzw. der Produktionslogistik untergeordnet [SVR10, Sch06, Neb07, DS10].

Im Rahmen der Produktionsforschung existieren zahlreiche Konzepte, die technologische Entwicklungen im Rahmen der Produktionssysteme und deren Planung und Steuerung beschreiben. CPPS stellen hierbei eine allgemeine Weiterentwicklung dieser Konzepte dar, die deren Ausprägungen beinhalten oder auf diese Konzepte angewendet werden können. Die bedeutendsten Konzepte, die aktuelle Entwicklungen im Rahmen der Produktionsforschung ausdrücken, sind nachfolgend beschrieben:

- Das Bionic-Manufacturing-Konzept betrachtet Produktionssysteme, abgeleitet aus der Biologie, als Organismen, die auf Einflüsse aus der Umgebung reagieren. Das Produkt selbst wird hierbei als Organismus angesehen. Dieser Organismus steht während der Produktion im Wettbewerb mit anderen Produkten. Produkt und Produktionssystem sind dabei selbstorganisiert und beinhalten genetische Informationen, die eine Evolution zur selbstständigen Verbesserung zulassen [UHF+00, Gro13].
- Flexible Fertigungssysteme nutzen Mechanisierung und einfache Automatisierungslösungen, um die Flexibilität von Produktionssystemen zu verbessern und die darauf produzierbare Produktvielfalt zu erhöhen [JKB03]. Die Einführung wurde durch die damit verbundene hohe Softwarekomplexität sowie die Investitions- und Wartungskosten bei geringer Rekonvaleszenz jedoch stark begrenzt [MUK00]. Durch fortschreitende Systementwicklungen im Rahmen von Industrie 4.0 erfährt dieses Thema jedoch ein gesteigertes Interesse [Hee17].
- Rekonfigurierbare Produktionssysteme sollen ähnlich wie flexible Fertigungssysteme die Flexibilität von Produktionssystemen erhöhen. Sie nutzen wiederverwendbare und modulare Elemente, um die Produktionsanlaufphase deutlich zu verkürzen und gleichzeitig die Zuverlässigkeit beizubehalten [KHJ99]. Die Realisierung der dabei entstehenden „Produktionsstraßen“ über Module, die sich über standardisierte Schnittstellen austauschen lassen, soll dabei die Zuverlässigkeit gegenüber flexiblen Fertigungssystemen erhöhen. Der Produktionsprozess eines Moduls selbst wird hierbei in der Regel nur geringfügig verändert und die Flexibilität des Produktionssystems ergibt sich durch die Zusammenstellung der Module. [Hee17]

- Cloud-based Manufacturing ist ein service- und kundenorientiertes Fertigungsmodell, bei dem die Kunden auf gemeinsame online-verfügbare Ressourcen zugreifen können. Es handelt sich somit um ein System zur Vernetzung von Produktionssystemen über Fabrik- und Unternehmensgrenzen hinweg. Durch die Nutzung verteilter Ressourcen sollen die Effizienz und insbesondere die Ressourcenauslastung der Produktionssysteme verbessert werden [DGR+13, VBC+20].
- Das Genetic-Manufacturing-Konzept zieht wie das Bionic-Manufacturing-Konzept Analogien zwischen Produktionssystemen und Organismen. Beim Genetic Manufacturing wird hierzu das Konzept der DNS eingeführt, wobei die DNS eines lebenden Organismus Produktinformationen repräsentiert. Die im Rahmen des Genetic Manufacturing verwendeten Algorithmen basieren auf diesem Grundgedanken und die Systeme sollen ebenso autonom agieren [Ued93, QML19].
- Ein Holonic-Manufacturing-System besteht aus autonom und kooperativ agierenden Holonen. Holone können hierbei sowohl reale Systemelemente (z. B. Arbeitsstationen, Produkte) als auch virtuelle Elemente (z. B. Kundenaufträge) darstellen. Die einzelnen Holone können untereinander interagieren. Zudem besitzt jedes Element einen definierten Grad an Autonomie, für den ein gewisses Maß an technischer Intelligenz (siehe Abschnitt 2.2) benötigt wird [Chr94, VV15].
- Ein Multi-Agenten-System (MAS) besteht aus einem System untereinander interagierender Agenten. Jeder Agent ist hierbei in der Lage, autonom Entscheidungen zu treffen und diese gegebenenfalls auch auszuführen. Agenten stellen dabei Softwarekonstrukte dar, die mit realen Systemen gekoppelt sein können [MVK06, VV15].
- Das Random-Manufacturing-Konzept basiert auf dem Gedanken, Planungs- und Steuerungsentscheidungen bezogen auf die Produktion erst während des Produktionsprozesses festzulegen und nicht, wie sonst üblich, bereits davor. Diese Herangehensweise soll die Flexibilität und die Resilienz von Produktionssystemen erhöhen [IO94, QML19].

CPPS stellen im Gegensatz zu den genannten Konzepten ein übergeordnetes Konzept dar, das Ansätze wie Multi-Agentensysteme beinhaltet und auf Organisationsformen wie etwa holonische Produktionssysteme angewendet werden kann [KWH13]. Ein wesentliches Merkmal von CPPS bildet die dezentrale Steuerung der einzelnen Elemente des Produktionssystems. Zur Veranschaulichung und Vereinfachung werden im Folgenden insbesondere holonische Produktionssysteme betrachtet, bei denen die einzelnen Produktionselemente (Holone) autonom agieren können. Dadurch können sie losgelöst von den anderen Elementen des Produktionssystems betrachtet werden. Diese Betrachtungsweise hilft der Veranschaulichung der dezentralen Steuerung, die im Rahmen dieser Arbeit entwickelt wird. Die Autonomie der einzelnen Produktionselemente bedeutet auf Grund nicht zu beachtender Abhängigkeiten eine

deutliche Vereinfachung für die Produktionssteuerung. Die in dieser Arbeit entwickelte Produktionssteuerung soll auf beliebige CPPS übertragbar sein, womit keine Festlegung auf eine Organisationsform oder ein Forschungskonzept vorgenommen wird.

1.4 Wissenschaftliches Vorgehen

In diesem Abschnitt wird das wissenschaftliche Vorgehen zur Entwicklung eines Verfahrens für die automatisierte dezentrale Produktionssteuerung mit digitaler Repräsentation, die die Berücksichtigung der Bedürfnisse und Anforderungen von Beschäftigten ermöglicht, beschrieben. Dabei kann wissenschaftlicher Erkenntnisgewinn grundsätzlich auf unterschiedliche Art und Weise erfolgen. Zudem existiert eine Vielzahl unterschiedlicher Verständnisse von Wissenschaftlichkeit [Sie10]. Das Verständnis weicht hierbei nicht nur zwischen, sondern auch innerhalb der Disziplinen voneinander ab. Ulrich [Ulr71, UH76] unterscheidet beispielsweise zwischen Formal- und Realwissenschaften, wobei sich letztere theoretische oder – wie in dieser Arbeit – praktische Ziele setzen. Die Dissertation ist somit den Realwissenschaften zuzuordnen. Es werden in der Dissertation Modelle der Realität, also Modelle von Produktionsprozessen von Unternehmen, für den Erkenntnisgewinn verwendet.

Im Wesentlichen stellt diese Arbeit die Entwicklung eines Informationssystems der Produktionssteuerung dar. Frank verknüpft mit dem Erkenntnisgewinn von Informationssystemen das Erkenntnisziel, die Wirtschaftlichkeit durch den Einsatz des Informationssystems oder bei dessen Entwicklung, Einführung, Betrieb und Wartung zu steigern [Fra07]. Die Wirtschaftlichkeit bezeichnet dabei eine Verbesserung im Hinblick auf einen der drei Aspekte Kosten, Qualität und Zeit.

Informationssysteme bezeichnen dabei stets sozio-technische Systeme, die durch das Wirken personeller und maschineller Aufgabenträger gekennzeichnet sind [ÖBF+10]. Der Erkenntnisgewinn bei Informationssystemen kann grundlegend auf zwei Arten erfolgen [ÖBF+10]. So basiert der verhaltensorientierte Ansatz darauf, auf Grundlage von Beobachtungen Erkenntnisse zu gewinnen. In der Regel werden auf Basis eines Tests entwickelte Theorien überprüft, um so eine Erkenntnis abzuleiten. Die Theorie beinhaltet dabei eine Vorhersage eines Verhaltens, das durch das Design, die Umsetzung und das Management von Informationssystemen nachzuweisen versucht wird. In der Regel wird versucht, diese Theorie durch empirische Studien nachzuweisen. Der zweite Ansatz ist der konstruktivistische Ansatz, bei dem eine problemlösungsorientierte Auseinandersetzung mit einem Thema erreicht wird. Grundlage hierfür bildet ein konkretes Problem, für das eine Lösung erzielt werden soll. Diese Lösung wird anschließend im Rahmen von Untersuchungen validiert.

Das Ziel dieser Arbeit liegt in der Entwicklung einer dezentralen Produktionssteuerung zur selbstorganisierten Steuerung von CPPS. Die wesentlichen Ergebnisartefakte bilden dabei ein Verfahren der Produktionssteuerung und eine digitale Repräsentation des Menschen, wobei letztere eine neue Herangehensweise zur Integration von Menschen in

Produktionssysteme darstellt. Diese Forschungsarbeit basiert auf einem konstruktivistischen Ansatz. Das wissenschaftliche Vorgehen für diese Arbeit ist dabei an die konstruktivistische Herangehensweise der Design Science Research Methodology for Information Systems Research (DSRM) nach Peffers angelehnt [PTR+07]. Peffers definiert hierin fünf Schritte für den wissenschaftlichen Erkenntnisgewinn:

1. Problemanalyse für die Identifikation des Forschungsproblems und Beschreibung des Ziels
2. Anforderungsanalyse zur Transformation des identifizierten Problems in Ziele
3. Konzeption und Verfahrensentwurf
4. Prototypische Implementierung
5. Validierung

Diese wissenschaftliche Vorgehensweise spiegelt sich im Aufbau der Arbeit wider, auf den im nächsten Abschnitt eingegangen wird.

1.5 Aufbau der Arbeit

Aufbauend auf der DSRM wird die Arbeit in acht Kapitel unterteilt (siehe Abbildung 2). Im Anschluss an die Einleitung, in der die Motivation, Zielsetzung, wissenschaftliche Vorgehensweise und der Aufbau der Arbeit erläutert werden, wird in Kapitel 2 die Ausgangssituation erläutert. Hierzu werden wesentliche Begriffe für diese Arbeit definiert und die für das Verständnis der Arbeit relevanten Grundlagen beschrieben, wobei insbesondere auf die Produktionsplanung und -steuerung eingegangen wird.

Kapitel 3 stellt danach den relevanten Stand der Forschung und Technik vor. Aufbauend auf den Grundlagen werden aktuelle Entwicklungen von Produktionssteuerungen und deren Übertragbarkeit auf die hier behandelte Problematik analysiert. Aus dem Stand der Forschung und Technik werden dann der Handlungsbedarf und die sich ergebenden Anforderungen an eine Neuentwicklung im Feld der Produktionssteuerungen abgeleitet, wodurch eine neue Art der Integration von Beschäftigten in Produktionssysteme ermöglicht wird.

In Kapitel 4 wird die Entwicklung der dezentralen Produktionssteuerung vorgestellt. Basierend auf den im Kapitel 3 abgeleiteten Entwicklungsanforderungen wird die entsprechende Lösung entwickelt.

In Kapitel 5 wird die Entwicklung und Implementierung der digitalen Repräsentation der Beschäftigten beschrieben. Diese dient als Instrument, um im Rahmen der selbstorganisierten Produktionssteuerung Änderungen durch den Menschen zuzulassen und eine menschenzentrierte Planung und Steuerung zu ermöglichen.

Darauf aufbauend wird in Kapitel 6 die Implementierung der dezentralen Produktionssteuerung im Smart Automation Labor des Lehrstuhls für

Produktentwicklung am Heinz Nixdorf Institut beschrieben. Hierzu wird im ersten Schritt die Erweiterung des Labors vorgestellt. Anschließend wird die Implementierung der Hard- und Software der dezentralen Produktionssteuerung vorgestellt.

Anschließend wird die Validierung der Produktionssteuerung anhand eines Anwendungsszenarios in Kapitel 7 vorgestellt. Hierbei wird neben Untersuchungen am realen System eine simulationsbasierte Validierung durchgeführt, um das Verhalten der dezentralen Produktionssteuerung mit bestehenden Produktionssteuerungsverfahren zu vergleichen. Dieser Vergleich zeigt die Anwendbarkeit des neu entwickelten Steuerungssystems für die Produktion individueller Produkte und die Verbesserung gegenüber bestehenden Ansätzen.

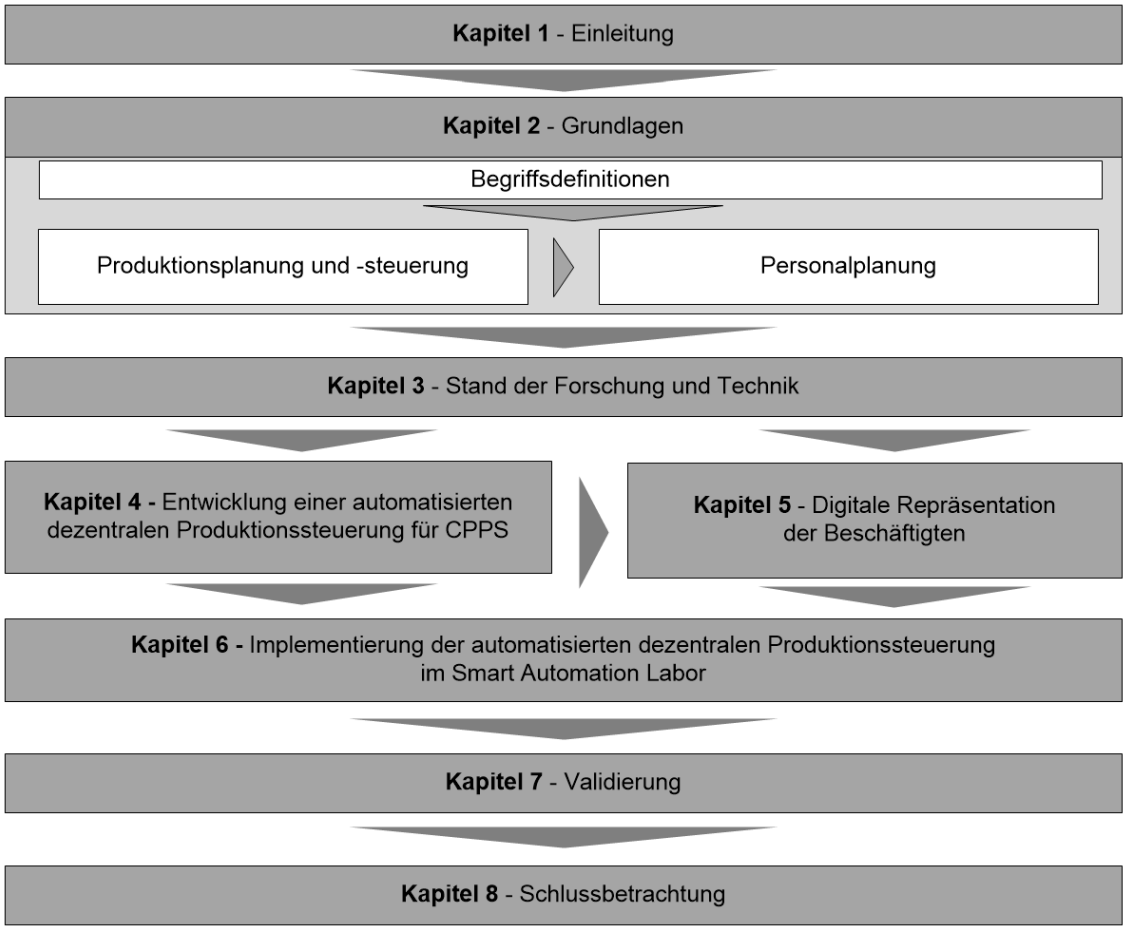

Abbildung 2: Aufbau der Arbeit

2 Grundlagen

Diese Arbeit untersucht die beiden folgenden zusammenhängenden und durch Industrie 4.0 motivierten Herausforderungen im Rahmen der PPS:

- die Entwicklung eines flexiblen, automatisierten Produktionssteuerungssystems für CPPS
- die Integration von Beschäftigten in das entwickelte Produktionssteuerungssystem

In diesem Kapitel werden vorab die in dieser Arbeit verwendeten, grundlegenden Begriffe erläutert. Die in der Literatur uneinheitlich verwendeten Begriffe werden definiert (Abschnitt 2.1). Anschließend werden die Grundlagen des Hauptthemenfelds dieser Arbeit kurz präsentiert und daraus die oben genannten beiden Herausforderungen abgeleitet. Zuerst werden die heute angewandten PPS-Ansätze in Abschnitt 2.2 vorgestellt. Diese PPS-Ansätze sind nicht auf den Wandel hin zu individualisierten Produkten und einer möglichst flexiblen Produktion zugeschnitten. CPPS wurden eingeführt, um diesen Herausforderungen systemtechnisch zu begegnen. Das Grundkonzept der CPPS wird zum Ende des Abschnitts vorgestellt. Die Planung von menschlichen Tätigkeiten wird in Abschnitt 2.3 beschrieben. Daraus werden die Herausforderungen zur Verbesserung der Planung von menschlichen Tätigkeiten und die Anforderung an die Gestaltung einer verbesserten Einflussnahme abgeleitet. Hierfür wird das Konzept der digitalen Repräsentation des Menschen entwickelt. Die Kernidee der digitalen Repräsentation ist dem digitalen Zwilling von technischen Systemen entnommen, welcher ebenso in Abschnitt 2.3 vorgestellt wird.

2.1 Begriffsdefinitionen

Produktionsplanung und -steuerung (PPS)

Die PPS beschäftigt sich mit der Planung und Steuerung von Fertigungs- und Montageprozessen eines industriellen Produktionsbetriebes. Planung und Steuerung erfolgen dabei unter Berücksichtigung von Mengen, Terminen und Kapazitäten [Sch12]. Die PPS-Systeme dienen der Beherrschung der Produktionsabläufe und der Umsetzung der Ergebnisse der Arbeitsplanung. Gerade bei komplexen Produktionsprozessen werden über die Effizienz der PPS die wesentlichen Einflussfaktoren einer wirtschaftlichen Produktion festgelegt [Wie14]. Hackstein hat die Aufgaben von der PPS wie folgt zusammengefasst und konkretisiert [Hac89]. Die Produktionsprogramm-, Mengen- sowie Termin- und Kapazitätsplanung werden den Aufgaben der Produktionsplanung zugeordnet. Die Auftragsveranlassung und -überwachung werden der Produktions-steuerung zugeordnet (siehe Abbildung 3). Der VDI [VDI92] definiert die Aufgaben der PPS entsprechend. Die Produktionsplanung bezeichnet dabei nach [VDI92] das Suchen und Festlegen von Zielen für die Produktion, das Vorbereiten von Produktionsaufgaben und das Festlegen des Ablaufes zum Erreichen dieser Ziele. Die Produktionsplanung

selbst wird auf Grundlage der Planungshorizonte noch weiter in eine lang-, mittel- und kurzfristige Planung unterteilt [Hee17]. Die Produktionssteuerung dient nach [VDI92] dem Veranlassen, Überwachen und Sichern der Durchführung von Produktionsaufgaben hinsichtlich des Bedarfs (Menge und Termin) sowie der Qualität, Kosten und Arbeitsbedingungen. Darauf aufbauend werden die Aufgaben der PPS in [Löd13] in die Produktionsprogrammplanung, die Produktionsbedarfsplanung, die Eigenfertigungsplanung und -steuerung (mit der Ablaufplanung als Unteraufgabe) sowie die Fremdbezugsplanung und -steuerung unterteilt. Im Rahmen der Eigenfertigung ist die Steuerung für die Reihenfolgeplanung der Aufträge verantwortlich. Anhand der festgelegten Auftragsreihenfolgen wird die Abarbeitung dieser in der Produktion veranlasst. Zudem finden ebenso ein Überwachen und Sichern der Durchführung der Auftragsreihenfolgepläne hinsichtlich Bedarf und Qualität statt [Sch12].

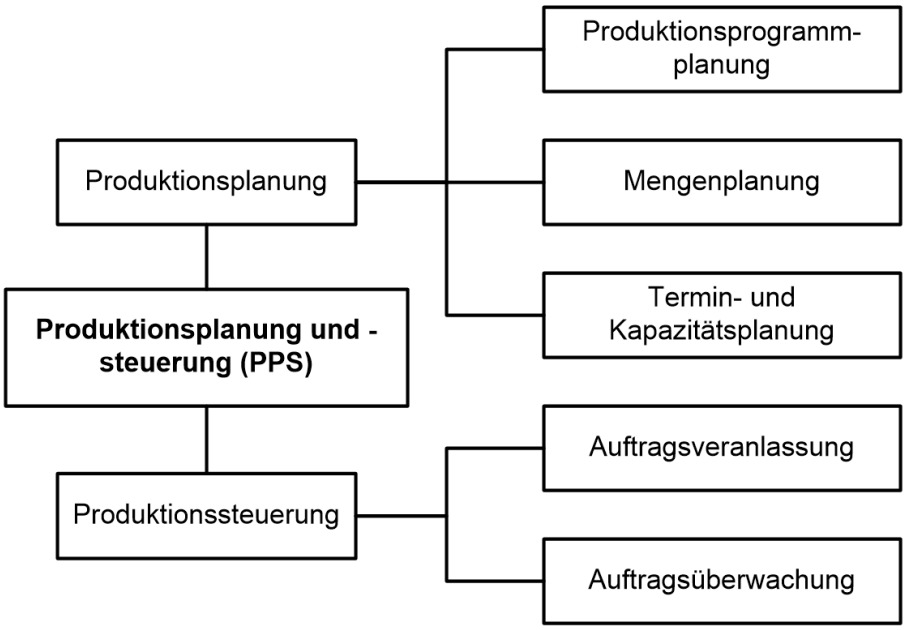

Abbildung 3: Aufgaben von Produktionsplanungs- und -steuerungssystemen [Hac89]

Cyber-physische Systeme (CPS)

Der Begriff CPS wird in der Literatur uneinheitlich definiert. Lee [Lee08] definiert CPS wie folgt: “Cyber-Physical Systems (CPS) are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa.” Es besagt, dass ein CPS aus untereinander vernetzten cyber-physischen Geräten (Cyber-Physical Devices) besteht, die Informationen austauschen und verarbeiten [Rei17]. In der VDI 2206 [VDI20] wird ein CPS als System, das reale (physische) Objekte und Prozesse mit informationsverarbeitenden (virtuellen) Objekten und Prozessen über offene, teilweise globale und jederzeit miteinander verbundene Informationsnetze verknüpft, bezeichnet. Ein CPS wird dabei maßgeblich durch seine Vernetzung mit dem Internet der Dinge und Dienste während des Betriebs

gekennzeichnet. Somit können sich Eigenschaftsänderungen während des Betriebs ergeben. Den Kern bilden dabei ein oder mehrere mechatronische Systeme mit ihren Sensoren, der Informationsverarbeitung und den Aktoren. Dadurch ist das System in der Lage, über seine Systemgrenze hinaus mit umgebenden Systemen zu kommunizieren und zu interagieren. In dieser Arbeit wird der Begriff CPS analog zur Definition der VDI 2206 verwendet, da hierin sowohl reale Objekte als auch deren Prozesse betrachtet werden.

Cyber-physische Produktionssysteme (CPPS)

CPPS stellen die Anwendung von CPS in der Produktion als Produktionssystem dar. Grundlage eines CPPS bilden vernetzte und digitale Produkte, die mit der Produktion verbunden sind [REG+13]. CPPS ermöglichen durch die Vernetzung von cyber-physischen Produktionsmitteln eine dezentrale, reaktionsfähige Produktions- und Logistiksteuerung [SRW+15]. Daraus ergeben sich die wesentlichen Eigenschaften von CPPS im Vergleich zu konventionellen Produktionssystemen [Bro10]:

- Ad-hoc-Vernetzbarkeit
- Selbstkonfiguration
- dezentrale, intelligente Datenverarbeitung

Dezentrale Produktionssteuerung

Die dezentrale Produktionssteuerung beschäftigt sich mit der Übertragung der Aufgaben der PPS von einem zentralen an viele dezentrale Elemente. Diese Dezentralität ermöglicht die dezentrale Entscheidungsfindung durch die einzelnen Elemente [WH07]. Die Vernetzung innerhalb von CPPS bietet die Möglichkeit, die notwendigen Informationen auf die dezentralen Elemente eines Produktionssystems für die Entscheidungsfindung zu aggregieren [Gro14]. Die Dezentralisierung macht dabei die Komplexität der Steuerungsaufgabe handhabbar [Rei09].

Autonomie und Selbstorganisation

Die Begriffe „Autonomie“, „Selbstorganisation“ werden häufig synonym verwendet. In dieser Arbeit wird durchgehend der Begriff Selbstorganisation verwendet. Selbstorganisation bildet eine Systemeigenschaft, die durch die grundlegenden Fähigkeiten zur eigenständigen und lokalen Informationsverarbeitung, Entscheidungs-findung und Ausführung bestimmt wird [Wyc09]. Selbstorganisation wird dabei als eine Erweiterung der Automatisierung angesehen und bezeichnet, dass ein System in der Lage ist, sich eigenständig, auch bei sich ändernden Randbedingungen, zu steuern [PS06].

Personalplanung

Personalplanung ist ein Teil der Unternehmensplanung und des Personalwesens [BB07]. Dabei müssen sowohl die Qualität und Quantität der Beschäftigten als auch die Kostenpläne, individuelle Erwartungen und betriebliche Erfordernisse gleichzeitig betrachtet werden [Oec06, Dru08]. In dieser Arbeit wird die Personalplanung bzw. die

Personaleinsatzplanung im Rahmen von PPS-Systemen adressiert, da die Planung der Einsatzzeiten und -orte von Beschäftigten ein wesentlicher Teil der Kapazitätsplanung von PPS-Systemen ist [Thi11]. In aktuellen PPS-Systemen wird der Mensch als Ressource angesehen und entsprechend über Durchschnittswerte von Arbeitsleistung und Anwesenheit eingeplant. Der aktuelle Stand der Forschung hinsichtlich Personaleinsatzplanung entspricht eben dieser Betrachtung des Menschen in der Produktion als Ressource [GGN17, BBB+13].

Digitaler Zwilling

Ein Digitaler Zwilling ist eine digitale Repräsentation einer Produktinstanz (reales Gerät, Objekt, Maschine, Dienst oder immaterielles Gut) oder einer Instanz eines Produkt-Service-Systems (ein aus Produkt und zugehöriger Dienstleistung bestehendem System). Diese digitale Repräsentation beinhaltet ausgewählte Merkmale, Zustände und Verhalten der Produktinstanz oder des Systems [SD19].

Ein Digitaler Zwilling wird aus dem Digitalen Master abgeleitet oder aus einer realen Produktinstanz erzeugt (siehe Abbildung 4) [SFL19]. Zudem kann ein Digitaler Schatten Daten einer realen Produktinstanz enthalten sowie Informationen über seine Herstellung abbilden. Ein Digitaler Schatten wird somit als Teilmenge eines Digitalen Zwillings verstanden. Der Digitale Zwilling enthält folglich Verknüpfungen aus Digitalem Master und Digitalem Schatten. Der Digitale Master beinhaltet dabei die Produktgeometrie sowie verhaltensbeschreibende Modelle eines Produkts bzw. Systems. Der Digitale Schatten wird als Abbild der Betriebs-, Zustands- oder Prozessdaten der realen Produktinstanz, auch hinsichtlich seiner Herstellung, verstanden. Ein Digitaler Zwilling ist in der Lage, entsprechende Daten der zugehörigen Produktinstanz zu erfassen, zu speichern, zu verarbeiten und bereitzustellen oder sogar bilateral mit der physischen Produktinstanz zu kommunizieren [WiGeP20].

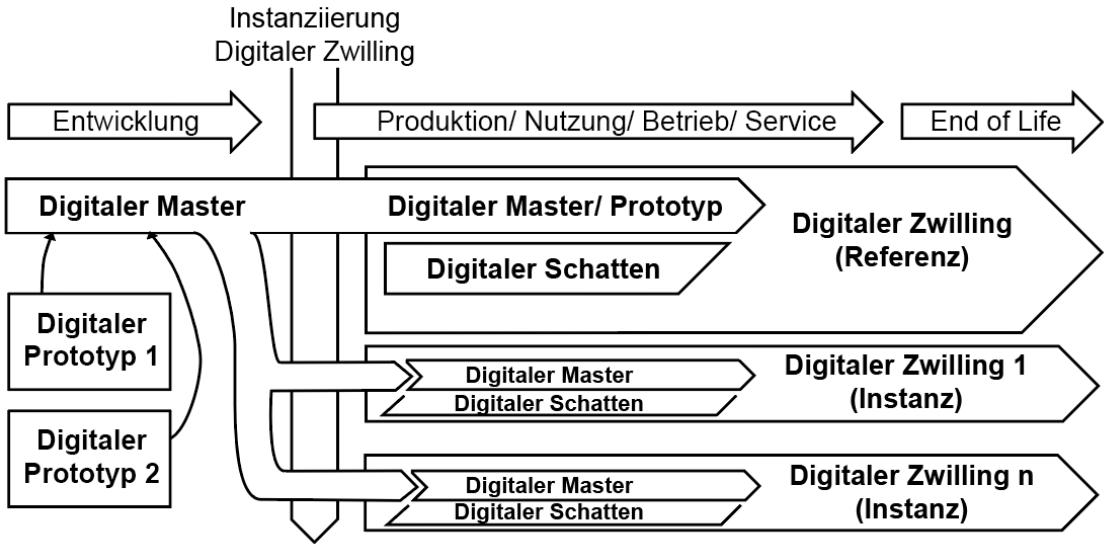

Abbildung 4: Definition des Digitalen Zwillings eines technischen Produkts [WiGeP20]

Die Anwendung des digitalen Zwillings ist dabei ausschließlich auf technische Systeme, bzw. technischen Systemen, die mit Menschen interagieren vorgesehen. Erst im Rahmen von Arbeit 4.0 wurde das Konzept des digitalen Zwillings auf Beschäftigte übertragen, um ihre Situation am Arbeitsplatz zu beobachten und zu verbessern. [Eng20]

Digitale Repräsentation des Menschen

Ein Ziel dieser Arbeit ist es, Beschäftigten in dem Produktionssystem Eingriffe in die Planung zu ermöglichen, um Steuerungsentscheidungen beeinflussen zu können. Aus diesem Grund wurde die digitale Repräsentation des Menschen entwickelt. Die digitale Repräsentation des Menschen ist dabei analog zu dem digitalen Zwilling von Produktinstanzen zu sehen. Die digitale Repräsentation des Menschen bildet das Verhalten eines individuellen Menschen nach und übernimmt bedeutsame Entscheidungen im Verhandlungsprozess. Hierzu werden alle relevanten Parameter der Tätigkeiten der Beschäftigten quantifiziert und entsprechend abgebildet. Der Begriff „digitale Repräsentation“ wurde gewählt, da das Verhalten der Beschäftigten analysiert wird, um die Verhandlung mit den anderen Elementen in der Produktion zu übernehmen. [GP17]

2.2 Produktionsplanung und -steuerung (PPS)

In diesem Abschnitt werden zuerst die wesentlichen Aufgaben und das Grundkonzept von PPS vorgestellt. Daraus werden zum Ende des Abschnitts die Herausforderungen bezogen auf eine flexible, automatisierte Produktion abgeleitet.

Die Aufgaben der **Produktionsplanung** sind in Abbildung 3 dargestellt. Die Aufgaben der PPS können in die Produktionsprogrammplanung, die Produktionsbedarfsplanung, die Eigenfertigungsplanung und -steuerung (mit der Ablaufplanung als Unteraufgabe) sowie die Fremdbezugsplanung und -steuerung unterteilt werden. Die Produktionsprogrammplanung erstellt auf Grundlage von Markt- und Vertriebsprognosen ein Produktionsprogramm, das die Primärbedarfe je Produkt und Planungsperiode angibt. Die Planungsperioden sind dabei stark von den Produkten, den zu produzierenden Aufträgen und dem Unternehmen abhängig. Mit der Festlegung des Produktionsprogramms unter Hinzunahme der konkreten Absatzplanung kann die Produktionsbedarfsplanung durchgeführt werden. Hierbei werden die eingespielten Aufträge eingeplant und deren Bruttoprimärbedarfe mit dem vorhandenen Lagerbestand verrechnet. Die sich daraus ergebenden Nettoprimärbedarfe bilden wiederum die Planungsgrundlage für eine grobe Planung der Ressourcen und damit des Produktionsprogramms. Aus den resultierenden Primärbedarfen des Produktions-programms werden mit Hilfe der Stücklisten die Bruttosekundär- und Bruttotertiärbedarfe von Baugruppen, Bauteilen und Rohmaterialien ermittelt. Abgeglichen mit den Lagerbeständen können die daraus resultierenden Nettobedarfe eingeplant werden. Je nach Beschaffungsart wird anschließend eine Kapazitäts- oder Beschaffungsplanung durchgeführt. Treten im Rahmen der Kapazitätsplanung

Unterdeckungen auf, wird versucht, diese über zusätzliche Kapazitäten oder die Fremdvergabe der Eigenfertigungsteile zu beseitigen. Die Fremdvergabe von Eigenfertigungsteilen ist beispielsweise in Hochkonjunkturzeiten von besonderer Bedeutung, wenn die Nachfrage nicht mit den Kapazitäten der Eigenfertigung bedient werden kann. Im Fall der Eigenfertigung legt die Eigenfertigungsplanung Losgrößen fest und führt eine detaillierte Ablaufplanung der Produktionsaufträge sowie eine Verfügbarkeitsprüfung benötigter Ressourcen durch. Im Rahmen der Fremdbezugsplanung werden die ermittelten Termine und Mengen entsprechend mit den Lieferanten verhandelt. Anschließend findet im Rahmen einer Eigenfertigungsplanung eine Terminierung der Aufträge statt. Die Querschnittsaufgaben umfassen das Auftragsmanagement, beispielsweise die Koordination von Produktionsaufträgen über verschiedene Bereiche der Produktion hinweg, das Lagermanagement und das Controlling der PPS. [Löd13, LE99]

Die **Produktionssteuerung** findet im Rahmen der Fremdbezugs- und Eigenfertigungssteuerung statt. Im Rahmen der Beschaffung müssen die eingeplanten Aufträge veranlasst werden. Die vom Lieferanten ausgeführten Aufträge müssen dabei bezüglich Qualität, Kosten und Einhaltung der ausgehandelten Lieferbedingungen überwacht werden. Die Steuerung der Eigenfertigung geschieht über die Reihenfolgeplanung der Aufträge, wobei anhand der festgelegten Auftragsreihenfolgen die Abarbeitung dieser in der Produktion veranlasst wird. Im Rahmen der Eigenfertigungssteuerung finden ebenso eine Überwachung und Sicherung der Durchführung der Auftragsreihenfolgepläne hinsichtlich Bedarf (Menge und Termin) und Qualität statt [Sch12].

Das oberste Ziel der PPS liegt in der Optimierung der Wirtschaftlichkeit, ausgedrückt durch das Zieldreieck von Qualität, Kosten und Zeit [Kur05]. Konkret kann die Produktionsplanung hierbei nach verschiedenen Zielgrößen optimiert werden. Die konkreten klassischen Zielgrößen der PPS umfassen dabei eine möglichst hohe Termintreue, eine möglichst hohe und gleichbleibende Auslastung, minimale Durchlaufzeiten und minimale Bestände [Hac89, Sch12]. Zusätzlich zu den vier klassischen Zielgrößen wurde eine möglichst hohe Flexibilität in die Betrachtung mit aufgenommen [Sch12]. Die Wechselwirkungen zwischen diesen Zielen erfordern eine entsprechende Abwägung, um ein ausreichendes Maß an Wirtschaftlichkeit zu erreichen. Beispielsweise führen höhere Bestände im Allgemeinen sowohl zu höheren Durchlaufzeiten als auch zu einer höheren Auslastung. Dieser immanente Zielkonflikt ist auch als das „Dilemma der Fertigungssteuerung“ [Wie14] bekannt. Wiendahl [Wie14] stellt dieses Dilemma in Abbildung 5 dar.

Die Zielerreichung in der PPS bezeichnet somit einen Zielkonflikt aus den konkurrierenden Zielgrößen. Die Zielgrößen können im Rahmen von PPS wenigen Regelgrößen zugeordnet werden, wobei die Termintreue wesentlich von den zur Verfügung stehenden Kapazitäten und Reihenfolgeabweichungen abhängig ist [Mei09]. Der Erfüllungsgrad der Reihenfolgeplanung wird durch die Reihenfolgegüte ausgedrückt

[Kuy13]. Bezüglich der anderen drei Zielgrößen ist – vorausgesetzt, es wird nicht mit Pufferzeiten geplant, – im Wesentlichen der zu Grunde liegende Bestand die zu beachtende Zielgröße. Nyhuis und Wiendahl [NH08] beziehen diese Zielgrößen hauptsächlich auf die Produktionsplanung, da die Produktionssteuerung nur für die Veranlassung der Produktionspläne zuständig und nur für kurzfristige Planänderungen verantwortlich ist, um eventuelle Planabweichungen zu korrigieren. Lödding [Löd13] fasst den Aufgabenbereich der Produktionssteuerung weiter, indem er diesem die Anpassung der Stellgrößen zuschreibt, die über Regelgrößen die gewünschten Zielgrößen erreichen sollen.

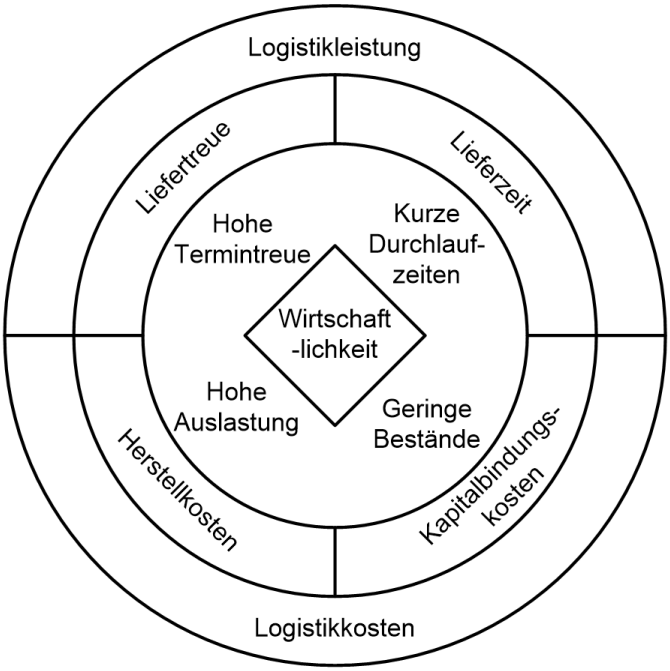

Abbildung 5: logistische Zielgrößen der PPS – Dilemma der Fertigungssteuerung [Wie14]

Produktionssteuerungsabläufe bestehen aus den drei Verfahren Auftragsfreigabe, Kapazitätssteuerung und Reihenfolgebildung [Löd08]. Da neue Aufträge für ein Produktionssystem entweder direkt freigegeben oder erst vorgehalten und dann nach bestimmten Kriterien freigegeben werden, können die Kriterien hierbei als vorgegebener Freigabezeitpunkt oder eine Unterdeckung des Auftragsbestands definiert werden. Dieses Vorhalten der Aufträge kann zur Steuerung der Produktion und zur Realisierung eines kontinuierlichen Produktionsflusses eingesetzt werden. Sobald ein Auftrag freigegeben ist, muss die Reihenfolge festgelegt werden, wann dieser zu produzieren ist. Die Reihenfolgebildung dient zur Erstellung eines Reihenfolgeplans, der ausdrückt, welcher Auftrag an einer Station als Nächster gestartet werden soll. Die Reihenfolge wird dabei in der Regel auf Basis der Liefertermine ermittelt. Die Kapazitätssteuerung prüft daraufhin im Betrieb, ob Maßnahmen zur Kapazitätserhöhung oder -absenkung erforderlich sind. Eine Maßnahme zur Kapazitätserhöhung kann zum Beispiel das

Veranlassen einer Überstunde sein. Durch einen kontinuierlichen Abgleich zwischen den zu produzierenden Aufträgen und den vorhandenen Kapazitäten wird dies geprüft. Die Verfahren zur Produktionssteuerung werden detailliert in Abschnitt 3.1 beschrieben.

Automatisierungspyramide zur Umsetzung von PPS

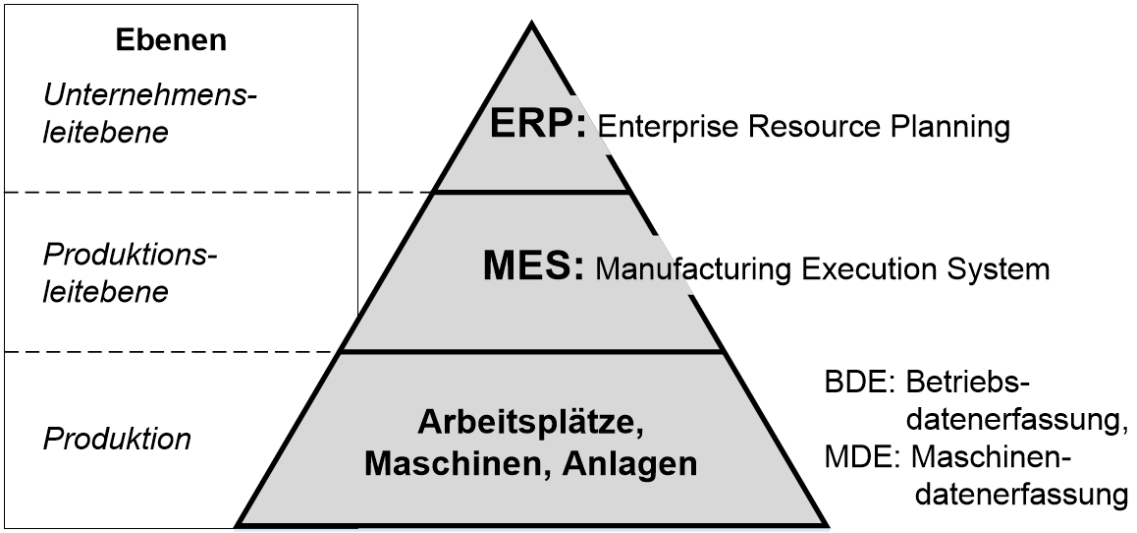

Abbildung 6: Automatisierungspyramide mit drei Ebenen nach [Kle06, VDI07]

Die informationstechnische Umsetzung von PPS-Systemen kann auf unterschiedliche Weise erfolgen. Bis zum Auftreten von Manufacturing-Execution-Systemen (MES) wurden PPS-Systeme in der Regel als Modul von Enterprise-Resource-Planning (ERP)-Systemen oder als eigenständige Software mit Schnittstellen zum ERP-System umgesetzt [Wie14]. Mit der Einführung von MES findet in der Regel eine Aufteilung der PPS-Funktionen statt. Die Beziehung zwischen dem ERP-System und MES kann mit Hilfe der Automatisierungspyramide (siehe Abbildung 6) dargestellt werden. Die Automatisierungspyramide drückt dabei die Zusammenhänge zwischen den für die Planung und Steuerung der Produktion eingesetzten Systemen aus. Es existieren unterschiedliche Arten der Darstellung der Automatisierungspyramide [MPM17], die sich in der Anzahl der Ebenen, der Terminologie und der Zuweisung der unterschiedlichen Aufgaben zu den Ebenen unterscheiden [Kle06]. Der Grundgedanke der Automatisierungspyramide zur Darstellung der Zusammenhänge zwischen den IT-Systemen in der Produktion ist jedoch in jeder Darstellung gleich [MPM17]. Für industrielle Produktionsbetriebe wird die Automatisierungspyramide häufig in die Unternehmensleitebene, die Fertigungsleitebene und die Fertigungsebene unterteilt. Diese Unterteilung ist zum Beispiel in der VDI 5600 dargestellt [VDI07]. Die Unternehmensleitebene verwendet als Softwarelösung in der Regel ERP-Systeme und ist im Rahmen der PPS für die Kapazitäts-, Material- und Arbeitsplanung zuständig. Das MES nimmt die Feinplanung, also die genaue Terminierung und Zuordnung zu Anlagen bzw. Kapazitätseinheiten, vor. Dies geschieht beim MES in der Regel im Rahmen einer

Leitstandsplanung. Auf der Produktionsebene finden die Steuerung und Überwachung der Produktion statt. Mit Hilfe von Maschinen- und Betriebsdatenerfassungsterminals erfolgt die Übermittlung der Produktionsreihenfolgen zu den Steuerungen sowie die Überwachung der Anlagen und Arbeitsplätzen.

Die Automatisierungspyramide soll die Zusammenhänge zwischen diesen Systemen verdeutlichen. Die Pyramidendarstellung veranschaulicht dabei mehrere Eigenheiten der Systeme. So nehmen die Informationsdichte und die Menge der aufgenommenen Daten nach unten stark zu, während die Gültigkeit von Planungszyklen stark abnimmt. Zudem nimmt die Anzahl der betrachteten Systeme nach unten hin zu. Während im ERP-System die Planung auf das gesamte Unternehmen und zusammengefasste Kapazitäts- und Beschäftigtengruppen angewendet wird, findet dies im MES für konkrete Anlagen statt, die im Rahmen des Produktionsprozesses auf einzelne speicherprogrammierbare Steuerungen und Arbeitsstationen heruntergebrochen werden [Sch13b].

Die Planung und die Steuerung der Produktion werden beispielhaft in der nachfolgenden Abbildung 7 verdeutlicht. Hierbei wird auf der rechten Seite der Planungs- und Steuerungszyklus der Produktion dargestellt und auf die einzelnen Ebenen der Automatisierungspyramide übertragen. Übernommen wurde der Planungs- und Steuerungszyklus von Pinedo [Pin16]. Da die Zuordnung unternehmens- und systemspezifisch ist, wird sie an dieser Stelle nur beispielhaft dargestellt und kann nicht verallgemeinert werden.

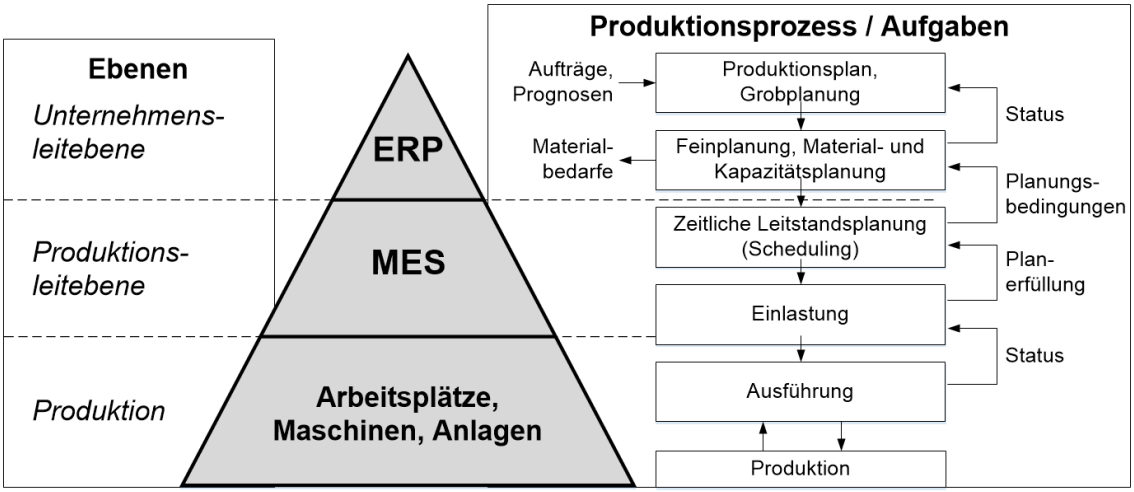

Abbildung 7: Vergleich der Automatisierungspyramide zum Planungs- und Steuerungsprozess [Pin16]

Ausgehend von den im ERP-System hinterlegten Aufträgen, Bedarfen und Vorhersagen wird die betriebswirtschaftliche Planung des nächsten Planungszyklus durchgeführt. Auf Basis der daraus resultierenden Ergebnisse wird im Rahmen der Produktionsprogrammplanung ein grober Produktionsplan festgelegt. Die Fremdbezugsentscheidungen können bereits mit der betriebswirtschaftlichen Planung oder aber in späteren Phasen der Planung getroffen werden. Darauf aufbauend wird auf

Grundlage der im ERP-System hinterlegten Bestände und der verfügbaren Kapazitäten die Material- und Kapazitätsplanung angestoßen. Die zeitliche Einplanung (Scheduling) der Aufträge und die Übertragung der Reihenfolgepläne geschehen mit Hilfe des MES. Das Ergebnis sind Reihenfolgepläne, die anschließend über Schnittstellen oder Visualisierungsinstrumente den Anlagen oder Arbeitsplätzen mitgeteilt werden und somit an den jeweiligen Stationen über direkte Schnittstellen zum MES ausgeführt werden können. Die in der Produktion eingesetzten Arbeitsplätze verfügen hierbei über Terminals zur Betriebsdatenerfassung (BDE) und die Speicherprogrammierbaren Steuerungen (SPS) der Produktionsanlagen werden in der Regel direkt über Feldbussysteme oder über Terminals zur Maschinendatenerfassung (MDE) mit dem MES verbunden. Auf dem Hallenboden findet anschließend die Ausführung dieser Planung statt.

Das Scheduling wird heutzutage meist durch Menschen manuell durchgeführt, die die Produktionsaufträge den entsprechenden Anlagen zuordnen. Für die automatisierte Planung existieren zahlreiche Methoden, insbesondere aus der wissenschaftlichen Disziplin des Operations Research. Die Zielsetzung des Operations Research liegt in der möglichst exakten Lösung von Optimierungsproblemen unter Berücksichtigung von Nebenbedingungen [NM02, KS04]. Hierfür wird ausgehend von einem Realproblem dieses Problem formalisiert und in ein mathematisches Problem umformuliert, das im Anschluss durch die Anwendung eines Algorithmus oder einer Heuristik gelöst wird [Mül92]. Die Teilgebiete des Operations Research umfassen die lineare Programmierung, die Graphentheorie und Netzplantechnik, die ganzzahlige und kombinatorische Optimierung, die dynamische Optimierung, die nichtlineare Optimierung, die Warteschlangentheorie sowie die Simulation [DD07]. Viele der im Rahmen der Produktionsplanung vorhandenen Problemstellungen sind sehr komplex, oder die zur Planung notwendigen Informationen sind nur unzureichend vorhanden, weswegen Probleme der Produktionsplanung mit analytischen Vorgehensweisen häufig nicht effizient in einer vertretbaren Rechenzeit gelöst werden können. [CD06]

Für Zuordnungs-, Reihenfolge-, Gruppierungs- und Auswahlprobleme eignet sich insbesondere die ganzzahlige und kombinatorische Optimierung. Die Maschinenbelegungs- oder Reihenfolgeplanung stellt dabei ein klassisches Problem des Operations Research dar [NM02]. Zur Modellierung von Maschinenbelegungsproblemen werden gemischt-ganzzahlige lineare Programmierungsansätze angewendet. Die Optimierungsmodelle dieser Programmierungsansätze beinhalten im Allgemeinen eine oder mehrere lineare Zielfunktionen sowie lineare Nebenbedingungen, wobei letztere dazu dienen, die spezifischen Rahmenbedingungen des Optimierungsproblems einzubringen. [DD07]

Die Lösung mathematisch formulierter Optimierungsmodelle kann über exakte und heuristische Verfahren geschehen. Exakte Lösungsverfahren garantieren dabei, dass die optimale Lösung für ein lösbares Entscheidungsproblem identifiziert wird. Die Verfahren unterteilen sich in Schnittebenenverfahren, in Entscheidungsbaumverfahren

(z. B. vollständige Enumeration, Branch-and-Bound-Verfahren), und in Kombinationen aus den beiden Verfahrensarten [DD07]. Bei komplexen Problemen wie z. B. der Maschinenbelegungsplanung ist eine Lösung des Problems mit exakten Verfahren aufgrund der großen Rechenzeit nicht immer möglich. Heuristiken bieten daher die Möglichkeit, gute Lösungen innerhalb kurzer Rechenzeiten zu identifizieren [NM02]. Diese Verfahren lassen sich in Eröffnungs- und Iterationsverfahren kategorisieren [ZB05]. Während ein Eröffnungsverfahren für eine Ausgangslösung geeignet ist, wird ein Iterationsverfahren zur sukzessiven Annäherung der Lösung an ein Optimum herangezogen, wobei in vielen Fällen ausgehend von der vorhergehenden Lösung ein weiteres Optimum gesucht wird.

Heuristische Lösungen garantieren zwar keine optimale Lösung, erzielen dafür aber häufig ein akzeptables Ergebnis in einer definierten Rechenzeit. Die meisten dieser Heuristiken werden gezielt für Problemstellungen entwickelt und nutzen problemspezifisches Wissen, um den möglichen Lösungsraum einzugrenzen. Im Gegensatz zu diesen problemspezifischen Heuristiken arbeiten Metaheuristiken (z. B. Tabu-Suche, Genetische Algorithmen) ohne problemspezifisches Wissen, sondern nutzen existierende Heuristiken oder Prioritätsregeln zur Errechnung einer Initiallösung in Kombination mit entsprechenden Verbesserungsprozeduren [Jun09].

Generell muss bei der Auswahl eines grundlegenden Lösungsansatzes zwischen den Vorteilen einer besseren Lösung der Produktionsplanung und den Nachteilen eines höheren Rechenaufwandes abgewogen werden. Reale Produktionsplanungsprobleme sind in der Regel NP-schwere mathematische Probleme, sodass optimale Lösungsansätze nicht anwendbar sind und mehrheitlich Heuristiken eingesetzt werden. In der Praxis ist das Finden einer gültigen Lösung – also einer Lösung, die sämtliche Nebenbedingungen und Restriktionen erfüllt, – im Rahmen der Produktionsplanung von größerer Relevanz als das Finden einer optimalen Lösung [Van09]. Planabweichungen erfordern neben der Kompensation durch die Fertigungssteuerung auch Änderungen am Produktionsplan. Dies ist der Fall, da Produktionssysteme in der Realität dynamischen und stochastischen Einflüssen und Störungen unterliegen. Eine solche Veränderungen kann beispielsweise durch eine sich ändernde Kundennachfrage oder Krankmeldungen von Beschäftigten hervorgerufen werden.

Heutzutage eingesetzte PPS-Systeme weisen eine eindeutige hierarchische Struktur zur Planung und Steuerung der Produktion auf: von der Produktionsbedarfs-, Kapazitäts- und Beschaffungsplanung über die Terminplanung bis zu der Produktionssteuerung. Die dabei angewandten starren Prozesse sind jedoch ein Gegensatz zu der geforderten schnellen Reaktion von flexiblen Produktionssystemen [GY16]. Falls dabei noch Änderungsbedarfe (wie zum Beispiel die Einbringung neuer Aufträge) auftreten, muss der gesamte Prozess umgeplant werden. Ein wesentliches Ziel der Einführung von **CPPS** ist diesen Planungs- und Steuerungsprozess optimieren zu können und flexibler zu gestalten [BLG17]. CPPS erweitern die Betriebsmittel des Produktionssystems und sollen systemseitig eine flexible Produktionssteuerung ermöglichen [REG+13]. Die wirksame

Umsetzung von Flexibilität in der Produktion erfordert ebenso eine organisationstechnische Flexibilisierung, die durch dezentrale Strukturen erreicht werden kann [SKN17, BFH+16, VV15]. Die Dezentralisierung ermöglicht die Vereinfachung der Komplexität der Steuerungsaufgabe [Rei09]. Demgegenüber stehen Nachteile von **dezentralen Produktionssteuerungen** insbesondere hinsichtlich der Sicherheit und Beobachtbarkeit gegenüber. Monostori [MVD+15] fasst die Vor- und Nachteile dezentraler Steuerungsstrukturen in der Produktion wie folgt zusammen:

Tabelle 1: Vor- und Nachteile dezentraler Steuerungsstrukturen in der Produktion [MVD+15]

Vorteile	Nachteile
• Offenheit • Zuverlässigkeit • Leistung • Skalierbarkeit • Flexibilität • Verteilung	• Dezentrale Informationen • Sicherheit/Vertraulichkeit • Kommunikations-Overhead • Entscheidung ‚Myopie' • Chaotisches Verhalten • Komplex zu analysieren

Die Produktion individueller Produkte erfordert insbesondere eine erhöhte Flexibilität. Dezentrale Produktionsorganisationsformen bieten hierfür enorme Vorteile, weswegen in der Forschung eine Verlagerung von zentral gesteuerten hierarchischen Systemen hin zu einer dezentralen, intelligenten Steuerung der Systemelemente stattfindet [SGP09]. Dabei existieren unterschiedliche Ausprägungsformen von Dezentralität. Ma und weitere [MNS19] vergleichen z. B. autonome, dezentrale Steuerungsmechanismen mit zentralen Strukturen. Die zentralisierte Planung erreicht bei statischen Produktionssystemen mit wenig Produktionsstufen unabhängig von der Systemkomplexität bessere Ergebnisse. Das dezentrale System erzielt jedoch bessere Ergebnisse für mehrstufige und dynamische Produktionsszenarien. Dabei zielen zentralisierte Methoden darauf ab, alle Einschränkungen gleichzeitig zu erfüllen. In dynamischen Umgebungen erwies sich diese Herangehensweise als zu restriktiv: dezentrale Methoden der Planung und Steuerung würden das Problem aufteilen und diese Teilprobleme sukzessive lösen. Die Organisationsformen werden in hierarchisch, hybrid und dezentral unterschieden. Die vollständige Umsetzung von Dezentralität wird hierbei als anarchisch (siehe Abbildung 8) bezeichnet.

Dezentrale Steuerungsstrukturen basieren nicht auf eindeutig festgelegten Hierarchien. Aufgaben an die Gesamtstruktur müssen durch Kommunikation zwischen einzelnen Elementen des Gesamtsystems erfolgen.

Auf Grund dieser Vorteile und dem Fokus auf CPPS wird in dieser Arbeit ein dezentrales Produktionssteuerungssystem entwickelt, um eine flexible, automatisierte Produktion von individuellen Aufträgen in einem Produktionssystem zu ermöglichen.

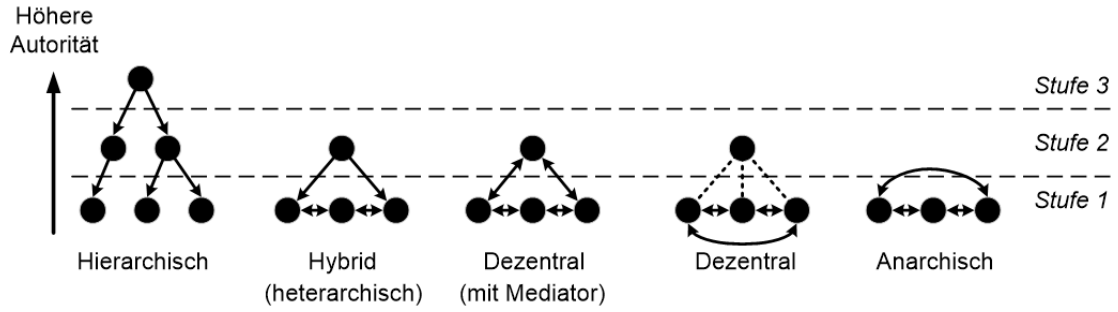

Abbildung 8: Unterteilung von Organisationsformen nach Zentralität (nach [MNS19])

2.3 Personaleinsatzplanung

Im Gegensatz zu der berechenbaren Planung und Steuerung von regelbasierten technischen Prozessen, gibt es bei von Beschäftigten ausgeführten Prozessen große Schwankungen. Diese Schwankungen zeigen sich insbesondere in den Bearbeitungszeiten der Produktionsprozesse, aber auch in Verfügbarkeiten und Arbeitsqualität. Dies erschwert die Planung und Steuerung erheblich und führt zu ungenauen bzw. falschen Planungsvorhersagen und damit zu Fehlplanungen. Um diese Fehlplanungen zu beseitigen, wird nachfolgend das dafür grundlegende Thema „Personaleinsatzplanung“ vorgestellt. Zuerst wird auf die derzeit angewandten Verfahren der Personaleinsatzplanung eingegangen. Darauf aufbauend werden die Probleme vorgestellt, die sich bei der Planung von Beschäftigten ausgeführten Produktionsprozessen ergeben. Die dabei zu Grunde liegenden Herausforderungen bilden die Motivation für die Entwicklung der digitalen Repräsentation von Beschäftigten. Die Entwicklung der digitalen Repräsentation von Beschäftigten wird anschließend in Kapitel 5 detailliert vorgestellt.

Die Personaleinsatzplanung ist ein wesentlicher Teil der Kapazitätsplanung von PPS. Die von den Beschäftigten ausgeführten Prozesse werden dabei wesentlich vereinfacht, um die Komplexität der Personal- bzw. Kapazitätsplanung auf ein handhabbares Niveau zu senken. Zum Beispiel werden Personalressourcen nur über Durchschnittswerte von Arbeitsleistung und Anwesenheit eingeplant [Thi11]. Die Zuordnung von Personal zu Arbeitsstationen oder Arbeitsstations-, bzw. Kapazitätsgruppen wird dabei als fest betrachtet [Sch12]. Dies bedeutet, dass es einen festen Pool an Bedienpersonal je Arbeitsstation oder Kapazitätsgruppe gibt, das ausschließlich auf den zugewiesenen Maschinen arbeitet. Für die PPS umfassen relevante Personaldaten die Qualifikation des Personals, Lohnkostensätze und den Zeitraum der Zuordnung zu einer Maschine oder Maschinengruppe. Die Ressourcenzuweisung des Personals findet dabei im Rahmen der Kapazitätsplanung statt. Nach der Zuweisung zu Arbeitsstationen werden bei der Zuweisung von Aufträgen und der zeitlichen Terminierung in der Regel nur die Arbeitsstation bei der Planung betrachtet. Eine entsprechende Planung von Beschäftigten findet wegen der hohen Komplexität in der Regel nur bei besonderen Fertigungs- und

Montageformen wie dem Handwerk, dem Schiffsbau oder allgemein der Baustellenfertigung statt.

Beschäftigte sind sehr flexibel einsetzbar. Diese Flexibilität muss jedoch bei der Planung und Steuerung berücksichtigt werden, bzw. das System muss eine solche Flexibilität individuell abbilden können [BTK+12]. Für die Planung ist eine große Menge an Randbedingungen zu beachten und für die Steuerung sind nur Steuerungsentscheidungen innerhalb dieser Randbedingungen möglich. Auf Grund dieser Komplexität können Beschäftigte als Ressource bezogen auf die Auswirkungen auf den Produktionsprozess nur schwer individualisiert abgebildet werden. Zudem unterscheiden sich Beschäftigte in ihrer Qualifikation, Erfahrung, Arbeitsgeschwindigkeit und zeitlichen Flexibilität. Die Auswirkungen dieser Ausprägungen auf die Ausführung des Produktionsprozesses können dabei ebenso nicht eindeutig abgebildet werden.

Darüber hinaus hat die Bedeutung der Befriedigung der Mitarbeiterbedürfnisse bei Personal- und Dispositionsentscheidungen stetig zugenommen. Unternehmen bieten Teilzeitverträge oder flexible Arbeitszeiten an und berücksichtigen bei der Erstellung von Arbeitszeitplänen die Präferenzen der Beschäftigten (z. B. Zusammenarbeit mit jemandem, Präferenz für einen bestimmten Schichttyp, bestimmte freie Tage und vieles mehr). Daraus ergeben sich weitere Personaleinsatzplanungsprobleme, welche sich unterschiedlich klassifizieren lassen. Eine der ersten Klassifizierungsmethoden für Personaleinsatzplanungsprobleme wurde von Baker vorgeschlagen [Bak76]. Dieser unterscheidet drei Hauptgruppen: die Schichtplanung, die Planung mit Fehltagen und die Tourenplanung, die die beiden ersten Typen kombiniert.

Eine andere häufig referenzierte Klassifizierungsmethode der Personaleinsatzplanungsprobleme basiert auf der Grundlage der angewandten Lösungsmethode [BBS91]. Alfares [Alf04] fasst in einer Umfrage Lösungsmethoden für die Zuordnungsproblematik im Rahmen der Personaleinsatzplanung zusammen: manuelle Lösung, ganzzahlige Programmierung, implizite Modellierung, Dekomposition, Zielprogrammierung, Arbeitspaketgenerierung und Metaheuristiken. Ernst et al. [EJK+04a] stellen eine Überprüfung der Personaleinsatzplanung vor. Ihre Klassifizierung stellt den Prozess der Dienstplanung (oder Personaleinsatzplanung) als eine Reihe von Modulen dar: Bedarfsmodellierung, Fehlzeitenplanung, Schichtplanung, Aufgabenzuordnung und Besetzung. Zudem überprüfen Ernst et al. [EJK+04a] die Dienstplanmethoden und -techniken, wobei sie die verschiedenen Ansätze in fünf Gruppen untergliedern: Nachfragemodellierung, Künstliche-Intelligenz-Ansätze (Fuzzy-Set-Theorie, Such- und Expertensysteme), Constraint-Programmierung, Metaheuristiken und weitere mathematische Programmieransätze.

Die zahlreichen Klassifizierungsmethoden versuchen mit verschiedenen Betrachtungsperspektiven die Personaleinsatzplanungsprobleme zuzuordnen und entsprechende Lösungen herauszufinden. Das spiegelt die hohe Komplexität der Personaleinsatzplanung wider. Allerdings müssen in der Regel verschiedene

Klassifizierungsmethoden in einem Unternehmen betrachtet werden, um den verschiedenen Personaleinsatzplanungsproblemen zu begegnen. Im Folgenden werden die Personaleinsatzplanungsprobleme im Rahmen der Produktion zusammengefasst.

Der Mensch wird bisher im Rahmen der Produktionsplanung und -steuerung als Ressource, ähnlich zu industriellen Anlagen, betrachtet. Die Einplanung der Aufträge geschieht in der Regel über die Maschinen bzw. Arbeitsplätze. Diese Zuweisung im Kapazitätsplan, der je nach Unternehmen täglich, wöchentlich, monatlich oder quartalsweise iterierend sein kann, wird anschließend normalerweise nicht mehr verändert. Eher werden kurzfristige Maßnahmen eingesetzt, um Lastschwankungen auszugleichen. Dies kann zum Beispiel die Einführung einer Überstunde, das Tauschen einer Schicht oder der Einsatz von zusätzlichem Personal sein. Auf Grund zahlreicher Faktoren treten solche Lastschwankungen sehr häufig im Betrieb auf. Neben Ausfällen z. B. durch Krankheit, unvorhergesehene Ereignisse oder Verzögerungen z. B. durch Probleme im Produktionsprozess sind diese Schwankungen jedoch häufig auf ungenaue und fehlerhafte Planungen zurückzuführen. Ein Grund hierfür stellt die Verwendung von ungenauen Zeiten zur Planung der Produktionsprozesse dar.

Die Aufträge, die anschließend den von Menschen bedienten Anlagen zugewiesen werden, werden über Durchschnittszeiten eingeplant. Eine individuelle Betrachtung der Auftragszeiten findet in der Regel nicht statt. Die Durchschnittszeiten ergeben sich über Vorgabezeiten, also Zeiten, mit denen festgelegt wurde, in welcher Zeitdauer ein Prozess zu erfolgen hat. Die Vorgabezeiten werden durch experimentelle Verfahren und analytische Methoden ermittelt:

Bei der Zeitaufnahme werden die Arbeitsschritte in einem möglichst optimalen Zustand des Arbeitssystems beobachtet. Gegebenenfalls werden sie in weitere Schritte zerlegt, die dann mittels Stoppuhr gemessen werden. Ein allgemeines Problem von Vorgabezeiten besteht darin, dass die Arbeitswelt Besonderheiten und Störungen unterworfen ist und diese nicht durch eine statische Zahl abgebildet werden können. Insbesondere dann, wenn im Rahmen der Tätigkeiten mit Schwankungen und Ausfällen zu rechnen ist, bilden diese Zeiten selten die Realität ab. Die Prozessausführung ist in der Praxis stark variabel und weicht von den Vorgabezeiten bzw. den Takten, die eigentlich als Grundlage für die Planung verwendet werden, ab. Auf Grund unterschiedlicher Leistungen, Handhabungen, ungeplanter Verzögerungen, Pausen, Ablenkungen etc. wird der Produktionsprozess auf unterschiedliche Weise von den eingesetzten Beschäftigten ausgeführt. Diese Diskrepanz ist in der nachfolgenden Abbildung 9 verdeutlicht. Hierin wird eine Vorgabezeit mit unterschiedlichen Operationen dargestellt. Der eigentliche Takt, also die äußere Vorgabe, wird dabei nur ab und zu eingehalten. Um den Prozess dennoch sicher ausführen zu können, ist es erforderlich, die Zeiten weitestgehend unter dem Takt zu halten. Dies führt zu dem generellen Problem, dass in der Auslegung und Planung die Potenziale der Arbeitsleistung nicht ausgeschöpft werden können (siehe Abbildung 9). Treten z. B. in einem Fließprozess Bearbeitungszeiten über dem Takt auf, entstehen daraus Verzögerungen und Unterbrechungen bei nachfolgenden Stationen.

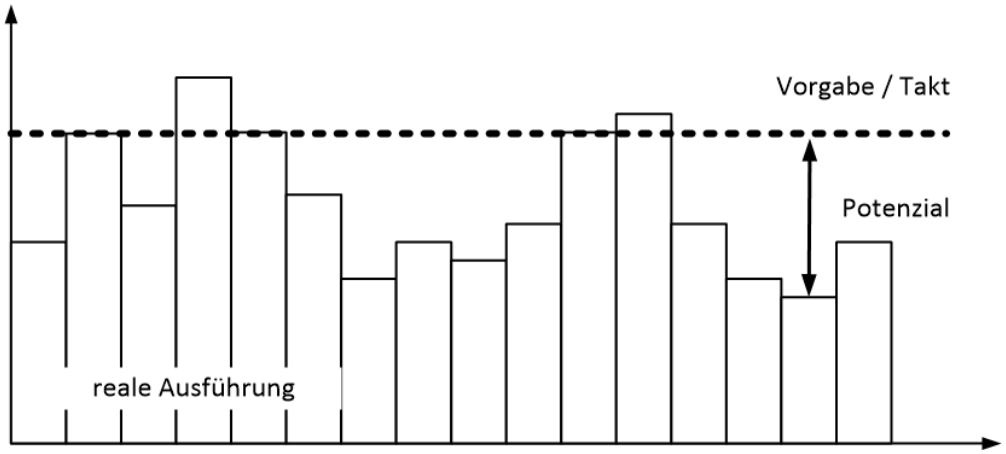

Abbildung 9: Diskrepanz zwischen realer Ausführung und Vorgabe

Es existieren somit große Verbesserungspotenziale hinsichtlich der Erkennung der tatsächlichen Ausführungszeiten und deren Nutzung in der Planung und Steuerung der Prozesse. Insbesondere, wenn Menschen einen Großteil des Arbeitsprozesses ausführen, verursacht dies größere Schwankungen, weshalb Potenziale durch die Verwendung von statischen Durchschnittswerten ungenutzt bleiben. Zur Optimierung gibt es bereits zahlreiche Bestrebungen [BQB10]. Im Folgenden werden zwei bekannte Verfahren zur Optimierung der Personaleinsatzplanung und ihre Ergebnisse vorgestellt.

Ein Verfahren, um individuell Planungen vornehmen zu können, ist die Verrechnung der Vorgabezeiten mit Leistungsgraden. Häufig findet diese Verrechnung nicht nur für einzelne Beschäftigte, sondern auch für Kapazitätsgruppen, z.B. in Form von Teams, Anwendung. Hierbei wird in der Regel die individuelle Arbeitsleistung von Beschäftigten in der Planung statisch berücksichtigt. Die Berechnung geschieht dabei wie folgt:

Die Zeitdauer t soll in diesem Fall die Bearbeitungszeit ausdrücken, die gleichwohl nur zum Teil von der menschlichen Leistung abhängig ist. Die Aufschlüsselung der Bearbeitungszeit ist in der nachfolgenden Abbildung 10 dargestellt. Im ersten Schritt können die Bearbeitungszeiten danach unterschieden werden, ob sie durch die Beschäftigten oder Betriebsmittel verursacht werden. Die Auftragszeit, also die Zeit, die an dem Auftrag gearbeitet wird, wird in Rüst- und Ausführungszeit unterteilt [REFA93]. Die Verteilzeit lässt sich nicht dem Prozess zuordnen und beinhaltet zum Beispiel Pausenzeiten oder auch Optimierungspotenziale.

Dabei unterliegen menschliche Tätigkeiten einer großen Variabilität, die unter anderem von Tempelmeier [Tem13] untersucht wurde. Dieser stellte fest, dass in der manuellen Montage Schwankungen von 80 bis 150 % in Unternehmen vorhanden sind. Zwischen einzelnen Mitarbeiterinnen und Mitarbeitern sind bei gleichen Tätigkeiten zudem Schwankungen von durchschnittlich 10 % mehr oder weniger der durchschnittlichen Bearbeitungszeit anzunehmen.

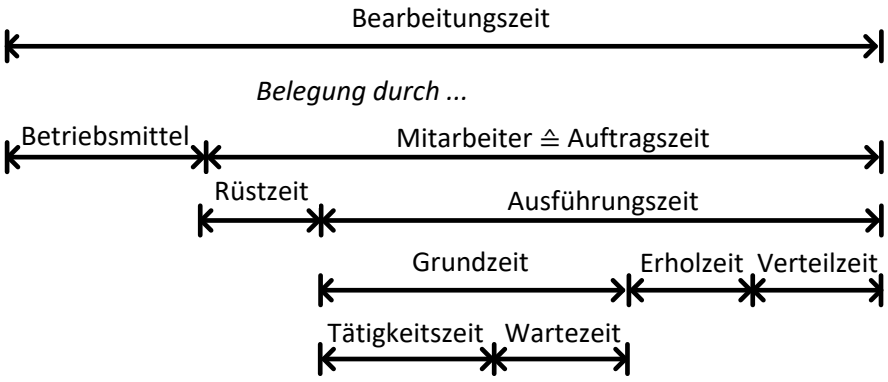

Abbildung 10: Aufschlüsselung der Bearbeitungszeit

Die Schwankungen können mit vorhandenen individuellen Stammdaten nur innerhalb eines Spektrums abgeschätzt werden. Die Prognose der Bearbeitungszeiten ist somit ebenfalls Schwankungen unterlegen. Dieser Umstand ist in der nachfolgenden Abbildung visualisiert. Glonegger hat in diesem Zusammenhang bereits eine Arbeit zur individuellen Betrachtung von Arbeitszeiten vorgelegt, deren Grundkonzept in Abbildung 11 ersichtlich ist [Glo14]. Hierbei wird versucht, die Schwankungen zu ermitteln und dadurch der realen Bearbeitungszeit anzunähern. Durch die Simulation der Schwankungen kann die Planung und Verteilung von Aufträgen auf unterschiedliche Situationen angepasst werden.

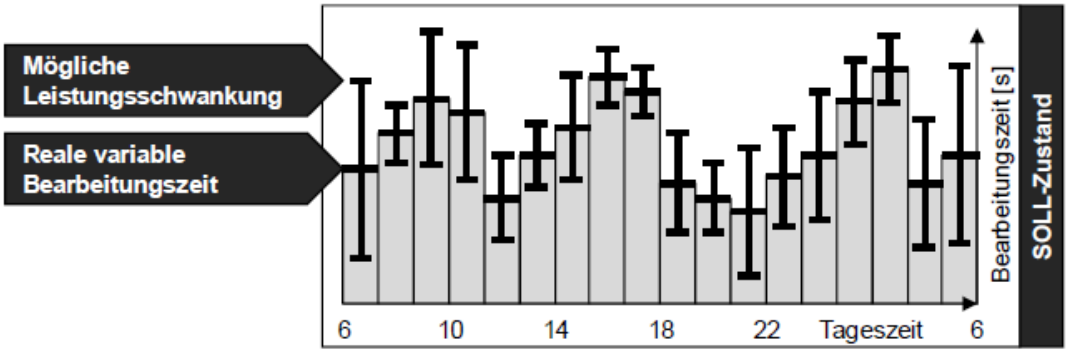

Abbildung 11: Schwankungen der Prozessausführung im Vergleich zur Vorgabe [Glo14]

Aus den oben dargestellten Optimierungsergebnissen wird die Ungenauigkeit der bisher genutzten statischen Berechnung der Arbeitsleistung von Beschäftigten und der Abschätzung der Ausführungszeit verdeutlicht. Das veranschaulicht das wesentliche Problem in der Planung menschlicher Tätigkeiten: die Diskrepanzen zwischen den realen Ausführungszeiten und den Planungswerten. Die digitale Repräsentation von Produktionsbeschäftigten bietet eine Datenaufnahme zur Berechnung der individuellen Leistungsdaten in Echtzeit. Die digitale Repräsentation soll dazu dienen, die Bearbeitungszeiten individuell zu prognostizieren, um diese anschließend für die Planung der Produktion verwenden zu können. Auf die genaue Messung der Zeiten sowie die

Abschätzung der Schwankungen und sich ergebenden Zeiten wird in Abschnitt 5 näher eingegangen.

Neben der Verbesserung der Grundlage der Bearbeitungszeiten soll die digitale Repräsentation der Beschäftigten dazu dienen, die Verteilung von Beschäftigten zu Anlagen und die Verteilung von Aufträgen zu Beschäftigten zu optimieren. Insbesondere die Flexibilität menschlicher Arbeitskraft soll durch den Einsatz der digitalen Repräsentation genutzt werden können. Mitarbeiterinnen und Mitarbeiter sind dazu befähigt, unterschiedliche Prozesse auszuführen, und sie können flexibel dort eingesetzt werden, wo sie für den Produktionsprozess am meisten gebraucht werden. Da die Verteilung von Beschäftigten jedoch sehr kompliziert ist, wird dies heutzutage nicht umgesetzt. Stattdessen wird versucht, die Arbeit auf einfache, gut planbare Prozesse herunterzubrechen. Die digitale Repräsentation ermöglicht die Realisierung dieser flexiblen Verteilung von Personal zu auszuführenden Aufgaben. Gleichwohl findet diese Verteilung nicht fremdbestimmt statt, sondern die Mitarbeiterin bzw. der Mitarbeiter kann Einfluss darauf nehmen.

3 Stand der Forschung und Technik

Auf Basis der Ausgangssituation sowie der dargelegten Grundlagen in Kapitel 2 wird nachfolgend der aktuelle Stand der Forschung und Technik beschrieben. Das Alleinstellungsmerkmal dieser Arbeit bildet die Entwicklung einer automatisierte dezentralen Produktionssteuerung für CPPS mit digitaler Repräsentation der Beschäftigten, um damit eine Selbstorganisation von Produktionssystemen zu erreichen. Entsprechend werden in diesem Kapitel Produktionssteuerungen (vgl. Abschnitt 3.1) und deren Einsatz in CPPS und die Einbindung der Beschäftigten in CPPS (vgl. Abschnitt 3.2) vorgestellt. Anschließend werden Anforderungen an die Entwicklung der dezentralen Produktionssteuerung und die Einbindung der Beschäftigten darin abgeleitet. Das Kapitel schließt mit den Forschungszielen und der Abgrenzung zum Stand der Forschung und Technik in Abschnitt 3.3 ab.

3.1 Produktionssteuerung in cyber-physischen Produktionssystemen (CPPS)

In den Abschnitten 1.3 und 2.2 werden die wesentlichen Eigenschaften von CPPS sowie die Grundlagen von PPS vorgestellt. Daraus ergeben sich die Schlüsselpotenziale von CPPS: die Selbstkonfiguration und die Anpassung auf sich verändernde Randbedingungen. Die Ausschöpfung der Potenziale hängt im Wesentlichen von der Produktionssteuerung ab. Selbstorganisierte CPPS werden durch den Einsatz von automatisierten Produktionssteuerungen ermöglicht.

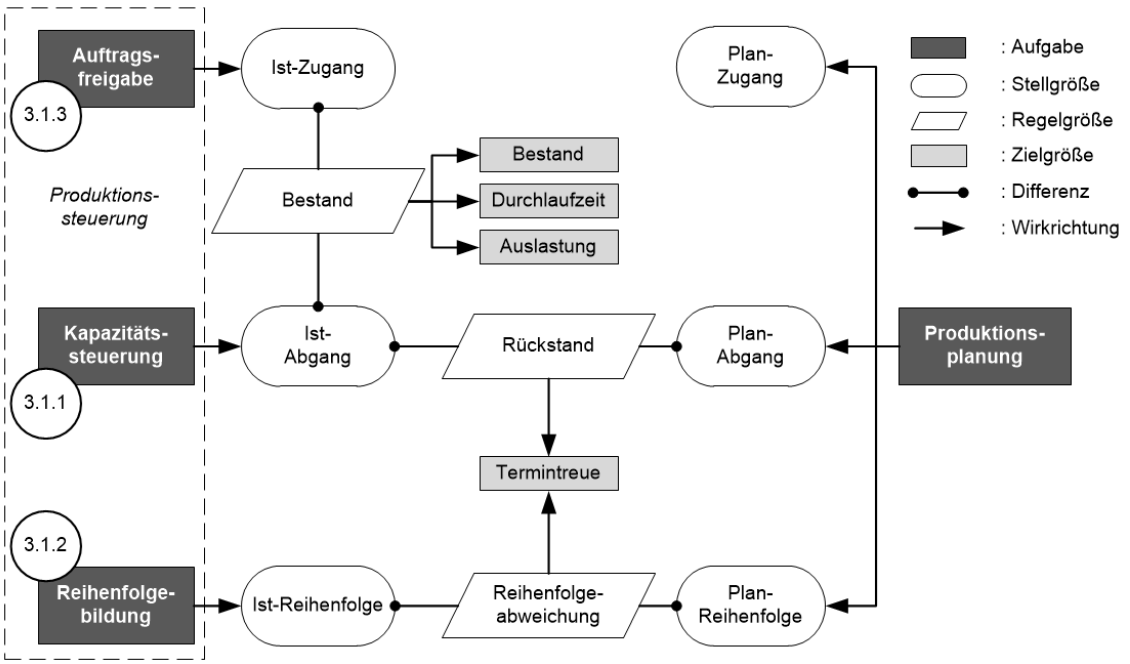

Abbildung 12: Ablauf der PPS mit Verweisen auf Abschnitte der Arbeit

Die Produktionssteuerung lässt sich in die drei Aufgaben Kapazitätssteuerung, Reihenfolgebildung und Auftragsfreigabe unterteilen [Löd08]. Ein Schaubild zur Darstellung der Aufgaben der PPS zeigt Abbildung 12. Auf der linken Seite sind dabei die Aufgaben der Produktionssteuerung mit den Verweisen auf die Abschnitte dargestellt. In der Mitte werden die logistischen Zielgrößen (siehe Abschnitt 2.2), die sich aus den Regelgrößen Bestand, Rückstand und Reihenfolgeabweichungen ergeben, dargestellt. Im Folgenden werden die Ansätze zur Durchführung dieser drei Aufgaben der Produktionssteuerung vorgestellt.

3.1.1 Kapazitätssteuerung

Bei der Entwicklung von Produktionssystemen wird eine Systemkapazität ausgelegt. Eine darauf aufbauende Kapazitätserhöhung ist z. B. durch einen längeren Mitarbeitereinsatz oder eine Anpassung des Prozessablaufs möglich. Diese Kapazität bildet die Grundlage für die Kapazitätssteuerung. Für jede Kapazitätsgruppe führt die Kapazitätssteuerung einen Abgleich zwischen der benötigten Kapazität und der vorhandenen Kapazität durch. Wenn es dabei Diskrepanzen gibt, greift die Kapazitätssteuerung mit Maßnahmen der Kapazitätsflexibilität ein. Maßnahmen können zum Beispiel das Ablehnen oder Vorziehen von Aufträgen und die Einrichtung von Zusatzschichten bzw. der Abbau von Überstunden sein. Durch die Ergreifung der Maßnahmen kann bei einem Unterangebot trotzdem die Termintreue gewährleisten werden. Bei einem Überangebot können durch die Umverteilung von Ressourcen Kosten eingespart werden. Im Gegensatz zu der langfristigen, vorausschauenden Kapazitätsplanung beschäftigt sich die Kapazitätssteuerung mit der kurzfristigen Veränderung des Kapazitätsangebots. Dabei ist zu beachten, dass eine kurzfristige Kapazitätserhöhung durch die Veränderung der Anzahl an Betriebsmittel oder die Fremdvergabe von Aufträgen bzw. Arbeitsvorgängen nur schwer umzusetzen ist [Löd08]. Im Folgenden werden zwei heute in der Praxis üblich eingesetzte Verfahren zur Kapazitätssteuerung vorgestellt.

Rückstandsregelung

Die Rückstandsregelung kann in Produktionssystemen verwendet werden, bei denen die Kapazität im Betrieb kurzfristig geändert werden kann. Die ist zum Beispiel die Montage, wo durch das Hinzufügen eines weiteren Beschäftigten die Kapazität einer Montagestation gesteigert werden kann. Das Verfahren basiert auf der Messung des Rückstands. Über die Differenz zwischen Soll-Produktion und Ist-Produktion wird der Rückstand berechnet. Sobald Schwellwerte des Rückstands überschritten werden, soll eine Anpassung der Kapazität oder eine Umplanung der Produktion stattfinden, um die geplante Produktionsausführung weiterhin gewährleisten zu können. Der Einsatz der Rückstandsregelung bietet sich in flexiblen Produktionsumgebungen an und ist im Vergleich zu anderen Verfahren einfach zu realisieren. Die notwendige Betriebsdatenerfassung und die Auswertung gestalten sich ebenso auf Grund der ausschließlichen Betrachtung der Soll- und Ist-Produktion sehr einfach. Erst bei der

Betrachtung von Wertschöpfungsketten und deren Rückschlüsse auf die einzelnen Stationen eines Produktionssystems müssen entsprechende Informationssysteme eingesetzt werden [Löd08].

Terminorientierte Kapazitätssteuerung

Die terminorientierte Kapazitätssteuerung strebt eine möglichst hohe Termintreue an. Die Berechnung der Termintreue erfolgt über die Betrachtung der gesamten Wertschöpfungskette. Die Grundlage der Berechnung basiert auf Verzögerungen an Anlagen. Es wird ermittelt, ob diese Verzögerungen zu einer Verspätung des Auftrages führen. Ist dies nicht der Fall, müssen keine Maßnahmen ergriffen werden. Gibt es jedoch Auswirkungen auf die Liefertreue, werden entsprechende Maßnahmen ergriffen. Werden Verspätungen prognostiziert, werden Maßnahmen zur Kapazitätserhöhung eingesetzt, um die Verspätungen zu vermindern bzw. zu vermeiden. Da die gesamte Wertschöpfungskette betrachtet wird, sind Berechnungen der terminorientierten Kapazitätssteuerung mit einem hohen Aufwand verbunden. Die Ergebnisse weisen dabei aber eine sehr hohe Güte auf, da direkt eine Folgenabschätzung mitaufgenommen wird und nur solche Maßnahmen ergriffen werden, die Auswirkungen auf das Gesamtsystem mit sich bringen [Beg05].

3.1.2 Reihenfolgebildung

Die Reihenfolgebildung dient der Erstellung des Reihenfolgeplans der zu produzierenden Aufträgen. Der Reihenfolgeplan drückt aus, welcher Auftrag an einer Station als Nächstes gestartet werden soll. Die Reihenfolge hängt dabei im Wesentlichen von den Lieferterminen und bestehenden Zusammenhängen in der Wertschöpfungskette ab. Derzeit agieren die meisten Unternehmen so, dass sie jedem Auftrag einen Liefertermin zuweisen und dann nur anhand des Liefertermins die Reihenfolgebildung vornehmen [Sch12]. Für eine optimale Reihenfolgebildung müssen jedoch alle vier Optimierungszielgrößen der Produktion – also Bestände, Termintreue, Durchlaufzeit und Auslastung – berücksichtigt werden. Nachfolgend werden die bekannten Verfahren zur Reihenfolgebildung erläutert.

First In – First Out (FIFO)

Das einfachste Verfahren ist das FIFO. Hierbei werden alle Aufträge nach der Reihenfolge bearbeitet, in der sie an einer Arbeitsstation auftreten. Die auftretenden Aufträge werden dabei nacheinander bearbeitet und neu eintreffende erst einmal zurückgestellt. Das Verfahren ist einfach umzusetzen und die Durchlaufzeiten bleiben auftragsbezogen relativ konstant. Die dadurch erhöhte Planungssicherheit verbessert die Liefertreue. Allerdings wird das Gesamtproduktionssystem beeinflusst, sobald Fehler oder Abweichungen von den Produktionsplänen auftreten [Löd08].

Frühester Starttermin

Bei diesem Verfahren werden Aufträge nach einem vorgeplanten Startzeitpunkt an den Arbeitsstationen sortiert. Dem Auftrag mit dem frühesten Starttermin wird dabei die höchste Priorität beigemessen und dieser wird nach der Fertigstellung des gerade bearbeiteten Auftrags als Nächstes bearbeitet. Voraussetzungen für dieses Verfahren sind die Festlegung eines Soll-Starttermins für jede Arbeitsstation und die Kommunikation des Starttermins an die jeweiligen Arbeitsstationen. Bei Umplanungen ist ebenfalls eine entsprechende Änderung dieser Starttermine erforderlich. Der sich daraus ergebende Aufwand verhindert eine weite Verbreitung dieses Verfahrens in der Praxis [Löd08].

Frühester Endtermin

Ähnlich wie beim Verfahren des frühesten Starttermins werden bei diesem Verfahren die Produktionsreihenfolgen nach dem frühesten geplanten Endtermin der Aufträge gebildet. Im Vergleich zu dem Verfahren des frühsten Starttermins ist die Termintreue in diesem Fall erhöht, da der Endtermin stets vor dem Liefertermin des Auftrags definiert wird. Das Verfahren ist einfach durchzuführen, falls nur die Endtermine des Gesamtauftrags weitergegeben werden. Diese sind in jedem Fall bekannt und können daher direkt mit den Arbeitsaufträgen verknüpft werden [Löd08].

Schlupfzeitregelung

Die Schlupfzeit bezeichnet die Zeitdifferenz zwischen dem Soll-Fertigstellungstermin und dem erreichbaren Fertigstellungstermin. Die Priorisierung der Aufträge wird hierbei nach der Schlupfzeit geregelt. Die Aufträge mit der niedrigsten Schlupfzeit erhalten hierbei die höchste Priorität. Die Reihenfolge wird anschließend nach dem Gefährdungsgrad bezüglich einer Nichterfüllung des Liefertermins gebildet. Dadurch wird eine hohe Termintreue garantiert: die Aufträge mit der größten Planabweichung werden zuerst bearbeitet und die Reihenfolge anderer Aufträge wird neu geregelt. Dieses Verfahren hilft insbesondere bei der Beseitigung von Störungen im Betrieb. Anstelle diese Störung wie beim FIFO-Prinzip an die nächsten Stationen weiterzureichen, werden die Aufträge mit den größten Abweichungen bevorzugt behandelt. Allerdings können viele Planabweichungen von diesem Verfahren resultieren, die bei stark verketteten Prozessen wiederum zu neuen Störungen führen können. Wenn ein Auftrag von mehreren Arbeitsstationen bearbeitet wird, hat ein Auftrag anfänglich genügend Schlupfzeit wegen der Addition der einzelnen Schlupfzeiten aller Arbeitsstationen. Nach dem Durchlauf einiger Arbeitsstationen wird die addierte Schlupfzeit immer geringer. Dies führt zu einer steigenden Priorität des Auftrags und kann bei unterschiedlichen Verkettungen Störungen der gesamten Produktion verursachen. Um dies zu umgehen, müssen zusätzlich Verfahren zur Verteilung der Schlupfzeit auf die Arbeitsvorgänge berücksichtigt werden [Stü12, Lop05].

Rüstoptimierte Reihenfolgebildung

Die rüstoptimierte Reihenfolgebildung ist vor allem für rüstaufwändige Produktionen einzusetzen. Dies ist der Fall, wenn Rüstzeitwechsel auftragsspezifisch sind und einen großen Anteil an der Bearbeitungszeit einnehmen. Dabei werden die Aufträge nach den geringsten Rüstzeiten sortiert. Dafür müssen alle Rüstzeiten der zu produzierenden Produkte und deren Auswirkungen untereinander bekannt sein. Das heißt, wenn zuerst Produkt A produziert wird, müssen die Rüstzeiten der anschließenden Produktion von Produkt B und C bekannt sein. Dadurch werden Engpässe möglichst maximal ausgelastet und die Gesamtauslastung entsprechend erhöht [Löd08].

3.1.3 Auftragsfreigabe

Verfahren zur Auftragsfreigabe können allgemein nach dem Detaillierungsgrad, der Auslösungslogik und danach klassifiziert werden, ob es sich um eine direkte oder verzögerte Auftragsfreigabe handelt. Aufträge können direkt nach dem Erstellen freigegeben werden oder zu einem bestimmten Zeitpunkt. Es können zudem Regeln bestehen, die die Freigabe beeinflussen. Der Detaillierungsgrad entscheidet darüber, ob der gesamte Auftrag freigegeben wird, oder ob zuerst dessen Arbeitsvorgänge separat freigegeben werden. Die Auslösungslogik sorgt dafür, ob die Aufträge intervallmäßig zu bestimmten Zeitpunkten oder ereignisorientiert freigegeben werden. Aus dieser Klassifizierung ergeben sich unterschiedliche Freigabeverfahren. Lödding hat hierzu die Verfahren der Auftragsfreigabe in die in Abbildung 13 dargestellten Klassen zusammengefasst [Löd13].

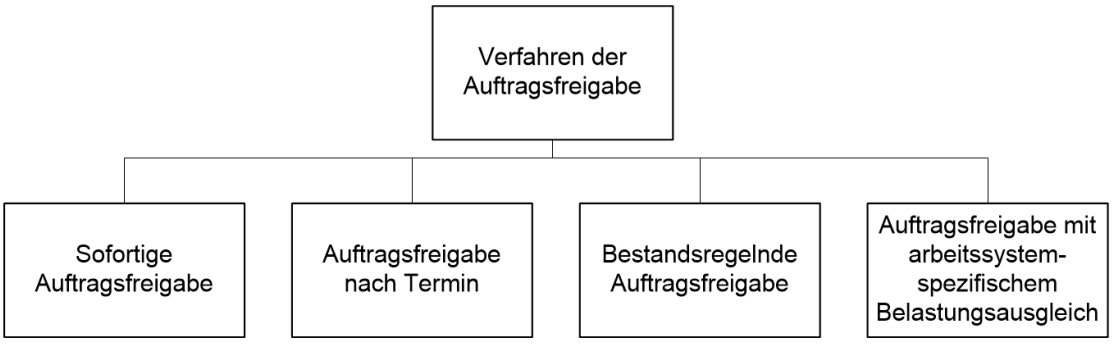

Abbildung 13: Verfahren der Auftragsfreigabe nach [Löd13]

Bei der **sofortigen Auftragsfreigabe** werden die Aufträge unmittelbar nach ihrer Erzeugung für die Produktion freigegeben. Bestandsprüfungen oder die Prüfung sonstiger Voraussetzungen werden hier nicht vorgenommen [Löd13].

Die **Auftragsfreigabe nach Termin** schreibt Termine vor, nach denen die Aufträge für die Produktion freigegeben werden. Dies wurde im Rahmen des MRP-II-Konzeptes (Material Resource Planning) etabliert und ist fester Bestandteil der Planung bei zahlreichen Unternehmen. Eine Bestandsregelung findet hierbei ebenso nicht statt. Diese

terminlich gesteuerte Auftragsfreigabe kann zu einer Glättung des Flusses auch periodisch erfolgen. Dies dient der zeitlichen Verteilung der Aufträge. Die Auftragsfreigabe kann zudem mit Hilfe von Regeln hinsichtlich weiterer Aspekte wie zum Beispiel Rüstzeiten optimiert werden.

Bestandsregelnde Auftragsfreigabeverfahren versuchen, die Kapazität der Produktion gleichmäßig auszulasten. Die Grundidee ist, dass der Output aus der Produktion durch neuen Input ersetzt wird. Dadurch soll ein konstant hohes Auftragsvolumen in der Produktion bearbeitet werden. Nachfolgend werden drei Verfahren im Rahmen der bestandsregelnden Auftragsfreigabe vorgestellt, auf welche in dieser Arbeit Bezug genommen wird. [Löd13]

Die Grundidee der **Engpasssteuerung** ist, dass eine Kette nur so stark wie ihr schwächstes Glied ist. Die Kette kann nur verbessert werden, indem das schwächste Glied verbessert wird. Auslastungsverluste im Engpass-Produktionsprozess führen zu Leistungsverlusten der gesamten Produktionslinie. Daher beruht die Engpasssteuerung auf dem Prinzip, dass ein neuer Auftrag für die Produktion erst freigegeben wird, wenn der Engpass-Produktionsprozess einen Auftrag fertiggestellt hat. Dafür teilt die Engpasssteuerung die Produktionslinie in einen bestandsgeregelten Bereich – bis einschließlich des Engpass-Produktionsprozesses – und einen nicht bestandsgeregelten Bereich nach dem Engpass-Produktionsprozess auf (siehe Abbildung 14). Sinnvoll eingesetzt werden kann dieses Verfahren, wenn ein eindeutig bestimmbarer Engpass-Produktionsprozess vorliegt. Bei der Engpasssteuerung setzt sich die Auftragsdurchlaufzeit aus zwei Bestandteilen zusammen: der Durchlaufzeit im bestandsgeregelten und der Durchlaufzeit im nicht bestandsgeregelten Teil des Produktionsprozesses. Die Ermittlung der Plan-Durchlaufzeit muss somit unterschiedlich erfolgen. [Löd13]

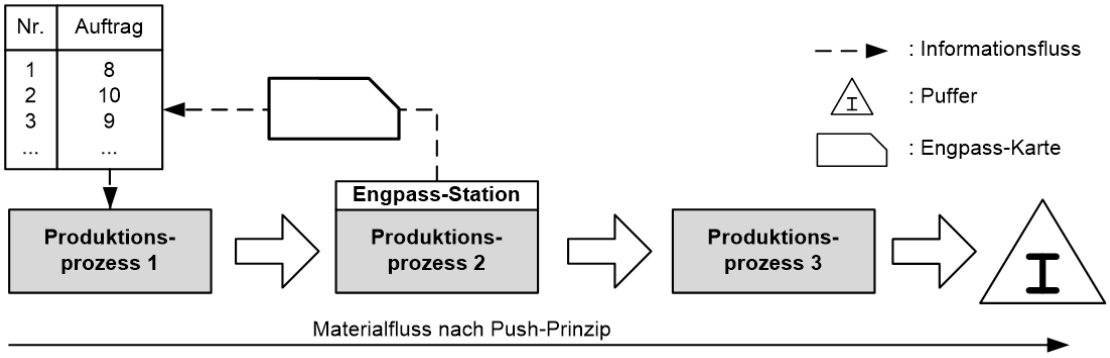

Abbildung 14: Funktionsweise der Engpasssteuerung nach [Löd13]

Das zweite Verfahren stellt das sogenannte **Constant-Work-in-Progress (CONWIP)** dar, bei dem Aufträge freigegeben werden, sobald eine bestimmte Untergrenze erreicht ist. Dieser Ablauf wird in der Abbildung 15 veranschaulicht: wenn der Produktions-prozess 3 die Untergrenze des Bestands erreicht hat, wird ein Signal für die

Auftragsfreigabe erzeugt. Die Reihenfolge der Auftragsfreigabe ergibt sich dabei nach vorher definierten Prioritätsregeln [SHW89]. CONWIP erzeugt gut prognostizierbare Durchlaufzeiten und ist für die Fließfertigung einfach umzusetzen, weswegen sich zahlreiche Anwendungen in der Industrie finden [Löd08]. Die Verfahrensregeln der CONWIP-Steuerung sind mit denen der Engpass-Steuerung vergleichbar. Der Unterschied dazu besteht darin, dass die Engpasssteuerung den Bestand nur bis zum Engpass-Arbeitssystem regelt, sobald der Bestand in der Produktionslinie bis einschließlich des Engpass-Arbeitssystems einen definierten Planwert erreicht.

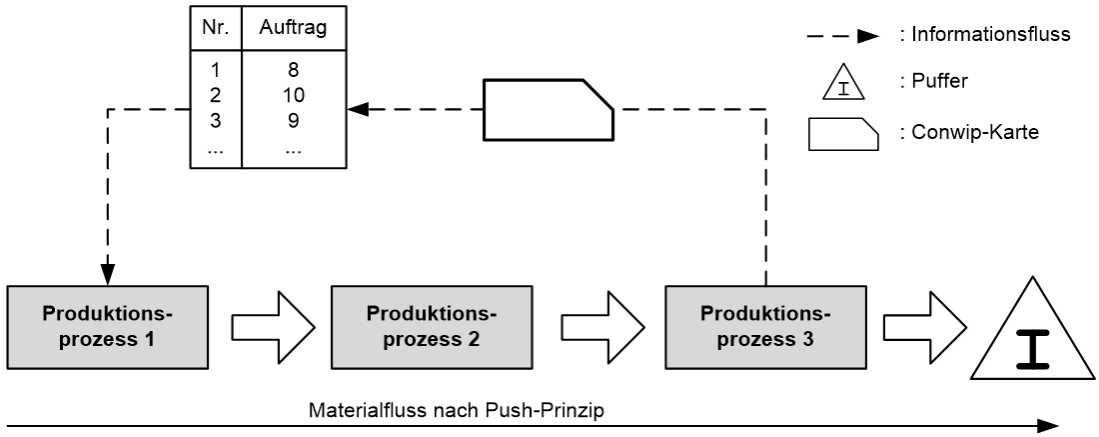

Abbildung 15: Funktionsweise der CONWIP-Steuerung nach [Löd13]

Das dritte Verfahren ist die **dezentrale bestandsorientierte Fertigungsregelung (DBF)**, die auf der dezentralen Umsetzung von CONWIP basiert. Bei DBF bilden die Bestände der Anlagen die Grundlage zur Steuerung der gesamten Fabrik. Die Abbildung 16 zeigt die Funktionsweise der DBF-Steuerung. Sobald der Bestand die Untergrenze des Produktionsprozesses 3 erreicht hat, wird die Freigabe im Produktionsprozess 2 ausgelöst. Derselbe Vorgang gilt ebenso für den Auslösungszyklus vom Produktionsprozess 2 zum Produktionsprozess 1. Der Produktionsprozess 1 entnimmt nach der Fertigstellung eines Auftrags neue Aufträge aus dem Auftragsspeicher [Löd01].

Zwei weitere häufig verwendete Verfahren sind die belastungsorientierte Auftragsfreigabe (BOA) und die Workload Control (WLC). Bei der BOA wird im Rahmen einer Rückwärtsterminierung eine Liste dringlicher Aufträge angefertigt, die in Abhängigkeit des vorhandenen Bestands freigegeben wird. Ist der Belastungsschwellwert einer Anlage unterschritten, wird der nächste Auftrag übermittelt. Das WLC-Verfahren arbeitet ähnlich und hält Aufträge zurück, die auf überlasteten Arbeitsstationen bearbeitet werden.

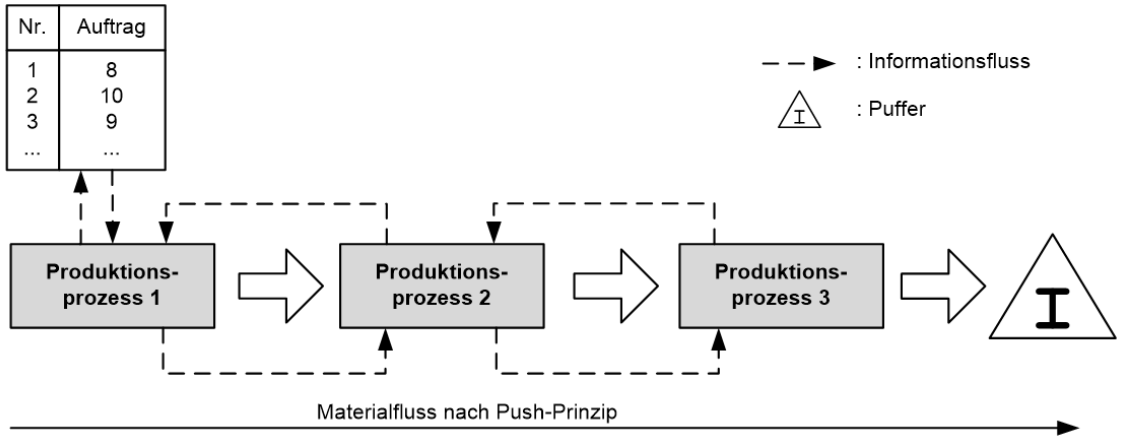

Abbildung 16: Funktionsweise der DBF-Steuerung nach [Löd13]

In der nachfolgenden Tabelle 2 werden häufig eingesetzte Verfahren der PPS zusammengefasst. Auf Grund der Festlegung auf eine dezentrale Organisation der Produktion und der Fokussierung auf Flexibilität, wird bei der Entwicklung der eigenen Verfahren auf die Verfahren DBF, Reihenfolgebildung anhand der Liefertreue und die Rückstandsregelung zurückgegriffen.

Tabelle 2: Aufgaben der PPS und dafür üblicherweise eingesetzte Verfahren

Aufgabe	**Verfahren**
Auftragserzeugung (Produktionsplanung)	MRP
	Kanban
Auftragsfreigabe	Terminbasierte Verfahren
	CONWIP – Constant Work in Progress
	BOA – Belastungsorientierte Auftragsfreigabe
	Engpasssteuerung
	DBF – Dezentrale Bestandsorientierte Fertigung
Reihenfolgebildung	Verfahren nach Zielgrößen: • Liefertreue • Leistung • Operationszeit
Kapazitätssteuerung	Rückstandsregelung
	Terminorientierte Kapazitätssteuerung

3.1.4 Ansätze zur Selbstorganisation

Nachfolgend werden auf Grundlage der Arbeiten von Grundstein und van Brackel [Gru17, Van09] Ansätze zur Selbstorganisation der Produktionssysteme vorgestellt. Diese Ansätze berücksichtigen unter anderem alle drei Aspekte der Produktionssteuerung: die Kapazitätssteuerung, die Reihenfolgebildung und die Auftragsfreigabe.

- Warteschlangenschätzer (Queue Length Estimator/QLE): Bei parallelen Arbeitsstationen wird hierbei die Warteschlange beider sich daraus ergebender Prozesse berechnet und der Auftrag der Anlage mit der kürzeren Warteschlange zugeordnet [SJB08].
- Termintreue-Methode: Aufbauend auf der Warteschlangenmethode werden für alle Stationen die Warteschlangen berechnet und die Aufträge werden nach dem frühesten Fertigstellungstermin priorisiert [SGP09].
- Prioritätsbasierter Ansatz: Hierbei wird die Priorität der Aufträge vor Auftragsfreigabe mit Hilfe der Berechnung der Warteschlange und deren Einfluss auf andere Aufträge berechnet und entsprechend eine Priorität vergeben, die für die anschließende Steuerung verwendet wird [PMT09].
- Pheromonbasierte Ansätze: Die mittlere Durchlaufzeit und weitere Faktoren der letzten Aufträge werden berechnet, um diese als Entscheidungsgrundlage für die Zuordnung der Aufträge zu Anlagen zu verwenden [SDF+08, SJB08].
- Autonomous-Manufacturing-System-based-on-Swarm-of-Cognitive-Agents-Methode: Hierbei handelt es sich um einen besonderen pheromonbasierten Ansatz, um den sogenannten Pheromonwert für Anlagen zu berechnen, also inwieweit die Anlage für den Auftrag geeignet ist. Berücksichtigt werden die Bearbeitungsfähigkeit, die Bearbeitungszeit und die Kosten der Ausführung [PT12].
- Potenzialfeld-Methode: Diese dient ähnlich wie pheromonbasierte Ansätze dazu, die „Anziehungskraft“ von Anlagen zu berechnen und anhand derer die Zuweisung vorzunehmen. Hierbei werden die Auftragstypen, Verfügbarkeit der Maschinen, Warteschlangenobergrenzen und Bearbeitungszeiten berücksichtigt [PBZ+12].
- Bienenfuttersuche-Methode: Über die Berechnung der Wertschöpfungsgewinne wird die Attraktivität einer Anlage berechnet. Von dieser Attraktivität werden die Kosten für die Produktion und die Durchlaufzeit über Faktoren abgezogen [SDF+08].
- Pheromonbasierte Koordination: Hierbei wird in Analogie zu Pheromonen ein kostenbasierter Ansatz verwendet, bei dem die Zuordnung von Aufträgen zu Stationen, die Auslastung der Station sowie die Kosten der Bearbeitung in einer Kostenfunktion berücksichtigt werden. Diese Kostenfunktion umfasst die Bearbeitungskosten, Lagerkosten und mögliche Verspätungskosten. Die Termintreue wird über die Verspätungskosten berücksichtigt [WTG+12].

- Bietermechanismen von Sudo & Matsuda [SM13]: Diese beinhalten Bietermechanismen zwischen Aufträgen und Anlagen, um möglichst kurze Durchlaufzeiten und geringe Transportkosten zu erreichen. Die Anlagen versuchen hierbei sich selbst und damit auch ihre Rüstzeiten zu optimieren.
- Contract Net Protocol: Aufträge werden den Anlagen nach deren Verfügbarkeit und Wahrscheinlichkeit der Termineinhaltung zugeordnet [VV15].
- NEU-Protocol: Aufbauend auf dem Contract Net Protocol wurde ein Protokoll entwickelt, dass es Systemkomponenten ermöglicht, Änderungen vorzunehmen (Upgrade, Anpassung usw.), ohne dass andere Komponenten geändert werden müssen [VV15].

Ferner gibt es noch zahlreiche andere Ansätze, wie zum Beispiel Verfahren zur zeitlichen Einplanung (Scheduling) von Produktionsaufträgen [Van09]. Die Einführung dieser Ansätze zur Selbstorganisation sind jedoch mit einem sehr hohen Aufwand verbunden. Alleine die Bereitstellung aller Informationen des Produktionssystems ist nicht realisierbar. Daher wird in dieser Arbeit ein dezentraler Ansatz zur Steuerung der Produktionssysteme entwickelt. Somit können die Berechnungsprozesse, die Kommunikationsprozesse und der damit verbundene Aufwand erheblich reduziert werden.

3.1.5 Ansätze zur Steuerung von CPPS

Nachfolgend werden Ansätze zur Steuerung von CPPS vorgestellt. Im Rahmen der Forschungsinitiative Industrie 4.0 [KLW11] wurden mehrere Projekte gefördert, die sich mit dem Einsatz und der Steuerung von CPS beschäftigt haben. Im Rahmen des Projektes „ProSense“ (Hochauflösende Produktionssteuerung auf Basis kybernetischer Unterstützungssysteme und intelligenter Sensorik) wurden unter anderem ein Produktionssteuerungssystem und ein Feinplanungssystem entwickelt, die auf Basis neuer Technologien Produktionsszenarien vorhersagen. Diese Vorhersagen dienen dazu, die Datengrundlage zu verbessern und Inkonsistenzen in den Daten zu beseitigen [Sch12]. Im Rahmen vom Projekt „CyPRoS“ wurden Ansätze zur Planung und Steuerung von Abläufen in der Produktion zur Umsetzung von CPPS erarbeitet [Rei13]. Ein wesentliches Ergebnis ist ein situationsbasierter Ansatz für CPPS, der unter anderem in der Dissertation von Engelhardt [Eng15] vertieft wird. Auf Grundlage eines Ansatzes für PPS, der in der Dissertation von Ostgathe [Ost12] entwickelt wurde, wird aufbauend auf der Idee von adaptiven Produktionssystemen ein RFID-gesteuertes Produktions-steuerungssystem konzipiert. Engelhardt setzt die Idee des Werkstücks (bzw. des Auftrags), das seinen Weg durch die Produktion selbst findet, durch ein RFID-gestütztes situationsbasiertes Produktionssteuerungssystem um. Das Thema seiner Dissertationsschrift ist insbesondere das Durchführen durch die Produktion und die Systematik zur Auftragsfreigabe. Die Steuerung der Produktionsprozesse geschieht bei Ostgathe produktbasiert und vereint viele Aspekte automatisierter dezentraler

Produktionssteuerungen. Im Rahmen des SFB 637 – „Selbststeuerung logistischer Prozesse – Ein Paradigmenwechsel und seine Grenzen“ wurden zudem weitere Grundlagen für agentenbasierte Steuerungssysteme geschaffen. Ein wesentliches Ziel des SFB bestand darin, auf Basis von softwarebasierten Agenten, die mit anderen Produkt- und Ressourcenagenten kommunizieren und verhandeln, selbstständig und unter Berücksichtigung der individuellen Zielsetzungen den eigenen Weg durch das Produktionssystem festzulegen [BW07].

Aufbauend auf der Arbeit von Ostgathe hat Hees [Hee17] ein Produktionsplanungssystem für rekonfigurierbare Produktionssysteme entwickelt, das der flexiblen Planung von rekonfigurierbaren Produktionssystemen dient. Motiviert durch Industrie 4.0 sind in Deutschland in den letzten Jahren viele Dissertationen im Bereich der automatisierten Produktionssteuerung entstanden [TWW17]. Meinecke präsentiert zum Beispiel eine Methode für die digitalen Vernetzung von Produktionsmaschinen zur Lastgangglättung im Betrieb [Mei17]. Trzyna entwickelt ein Produktionssteuerungssystem zur besseren Einplanung von Eilaufträgen in der Produktion [Trz15]. Berlak legt Grundlagen für ein modellbasiertes Verfahren zur Berechnung der Termintreue [Ber15]. Grundstein präsentiert mit seinem System der selbstorganisierten Fertigungsregelung eine neuartige Produktionssteuerung für CPPS [Gru17]. Isenberg stellt ein PPS-System für mehrstufige Batchproduktionen vor, dessen verwendete Regeln der Terminierung und Freigabe ebenso eine bedeutsame Basis für diese Arbeit darstellen [Ise16].

Lopitzsch [Lop05] hat mit seiner segmentierten adaptiven Fertigungssteuerung zum Beispiel ein hybrides Auftragserzeugungsverfahren für die variantenreiche Serienproduktion entwickelt. Über den Kanban-Ansatz kombiniert mit einer Materialbedarfsplanung wird hierbei eine dezentrale Produktionssteuerung realisiert. Seibold leitet daraus eine Methodik für den Einsatz von hybriden Verfahren in der Großserienproduktion ab [Sei06]. Im SFB 652 „Gentelligente Bauteile im Lebenszyklus“ werden unter anderem produktbasierte Steuerungsverfahren entwickelt, wodurch eine Weiterentwicklung des Gedankens „das Produkt findet seinen Weg durch die Produktion“ erfolgt. Ein Ansatz bilden hierfür sogenannte gentilligente Bauteile [DLS12].

Zusätzlich zu den aufgeführten Dissertationen existieren zahlreiche Publikationen, die verschiedene Problemstellungen selbstorganisierter Produktionssysteme behandeln [CHM15, WWI+16]. Einen Überblick über den Stand der Forschung in Bezug auf flexible, dezentrale Steuerungssysteme gibt Leitao [Lei09], der die Verfahren zur dezentralen oder hybriden Produktionssteuerung von holonischen Produktionssystemen mit Hilfe von Multiagentensystemen zusammenfasst. Das Werk „Design of the Unexpected“ [VV15] stellt das aktuelle Rahmenwerk zu holonischen Produktionssystemen sowie deren Planung und Steuerung dar. Hierin wird unter anderem die Steuerung von holonischen Produktionssystemen mit Hilfe verteilter Multiagentensysteme beschrieben. Weitere Werke, die eine Umsetzung der Planung und Steuerung von holonischen Produktionssystemen beschreiben, wurden von Jana et al. [JBP+13, JBP+14] und Raileanu [Rai10] verfasst, wobei letzterer die Umsetzung eines

dezentralen Produktionssteuerungsverfahrens in einem Produktionslabor präsentiert. Raileanu setzt einfache Steuerungsregeln in Kombination mit der innerbetrieblichen Logistik des Produktionssystems um. Die Umsetzung ist jedoch nur für die Steuerung des dort beschriebenen Produktionslabors ausgelegt. Eine weitere Umsetzung eines CPPS findet sich an der TU Wien [HRT+19]. In weiteren Veröffentlichungen im Rahmen der verteilten Planung und Steuerung wird allgemein die Herangehensweise [HCC+16] und die Implementierung geeigneter Steuerungssysteme [THT15] erläutert. Die Herangehensweise zur Entwicklung der verteilten PPS-Systeme lässt sich in zwei Problemstellungen untergliedern: die Verhandlung zwischen den Modulen des Produktionssystems und die zeitliche Einplanung auf den jeweiligen Systemen [MSK+16]. Die zeitliche Einplanung von Aufträgen wird in einem umfassenden Sammelwerk von Pinedo [Pin09] behandelt. Lang et al. [LFB16], Monostori et al. [MVD+15] und Thomas et al. [TTV12] bilden verschiedene Ansätze zur Verhandlung zwischen Agenten von dezentralen Steuerungssystemen ab. Monostori et al. fassen aktuelle Systeme zur kooperativen, verteilten Steuerung zusammen. Thomas et al. besprechen den Stand der Forschung zu intelligenten, dezentralen Produktionssteuerungen. Lang et al. entwickeln mit ihrem Ansatz ein konkretes Beispiel einer Agentenverhandlung zur Verteilung von Aufträgen.

Die genannten Publikationen bilden die Forschungsbasis für diese Arbeit. Ihre Vorleistung hat belegt, dass eine dezentrale Produktionssteuerung zur Umsetzung der CPPS realisierbar ist. Daraus ergibt sich für die Entwicklung der dezentralen Produktionssteuerung die Forschungslücke: eine automatisierte dezentrale Produktionssteuerung ist erforderlich, um CPPS zur Selbstorganisation zu befähigen. Die bisherigen Ansätze funktionieren ausschließlich im Rahmen der Produktionssteuerung, so dass die Auftragsumplanung, die als der entscheidende Faktor zur Ausschöpfung der Produktionskapazität gilt, nicht berücksichtigt wurde. Das heißt, wenn eine Auftragsumplanung erforderlich ist, muss diese weiterhin manuell durchgeführt werden. In dieser Arbeit wird daher eine automatisierte dezentrale Produktionssteuerung zur Selbstorganisation von CPPS entwickelt.

3.2 Personaleinsatzplanung in selbstorganisierten CPPS

In Abschnitt 2.3 wird das Handlungsfeld „Personaleinsatzplanung“ beschrieben. Darin wurde abgeleitet, dass Verbesserungsbedarfe bezüglich der Personaleinsatzplanung und der Berücksichtigung von Mitarbeiterbedürfnissen bestehen. Die digitale Repräsentation der Beschäftigten wurde als Lösungsansatz für die Verbesserungsbedarfe ausgearbeitet. In diesem Abschnitt werden Arbeiten zur Einbindung von Beschäftigten in selbstorganisierten Produktionssystemen vorgestellt. Mit der Einführung von CPPS werden sich nicht nur die Organisations- und Steuerungssysteme in der Produktion verändern, sondern auch die Einbindung der Beschäftigten in der Produktion wird einem grundlegenden Wandel unterzogen. Die Untersuchung dieses Wandels wird in

zahlreichen Studien vorgenommen. Ein Hauptuntersuchungsgegenstand bildet dabei die Frage, ob Computer in Zukunft Menschen in der Produktion ersetzen werden.

Einen Ausgangspunkt stellt dabei die Studie von Frey und Osborne [FO13] dar, die das Automatisierungspotenzial von Arbeitsplätzen untersucht haben. Insbesondere durch das Fortschreiten der Computertechnologie in weiteren Anwendungsbereichen besteht die Erkenntnis ihrer Studie darin, dass 47 % aller Arbeitsplätze Gefahr laufen, durch Computertechnologie automatisiert zu werden. Einfache, repetitive Tätigkeiten insbesondere in der Produktion sind stark von diesem Wandel betroffen. Diese Studie ist Gegenstand zahlreicher Diskussionen und auch neuer Studien. Die Studie betrachtet, ob durch Entwicklungen von Industrie 4.0 Arbeitsplätze automatisiert werden. Das Institut zur Zukunft der Arbeit stellt in seiner Studie aus dem Jahr 2015 [SGG+13] zum Beispiel fest, dass Industrie 4.0 wahrscheinlich zu einer wachsenden Lohnspreizung zwischen besser und schlechter entlohnten Berufsgruppen führen wird, da durch die Automatisierung und den verstärkten Einsatz von Informationssystemen die Anforderungen an einfache Tätigkeiten weiter sinken werden. Die Studie geht allerdings nicht davon aus, dass es zu starken Veränderungen in der Anzahl der Beschäftigten durch die Einführung von Industrie 4.0 kommen wird. Graetz und Michaels [GM15] erzielen auf Basis ihrer Untersuchung der Einführung von Industrierobotern von 1993–2007 und der sich daraus ergebenden Auswirkungen auf die Beschäftigung ähnliche Ergebnisse. Zwar habe sich der Einsatz von Robotern stark erhöht, aber dadurch sei die Beschäftigung, also die geleisteten Arbeitsstunden, in diesen Betrieben nicht zurückgegangen.

Aus ingenieurwissenschaftlicher Sicht wird stattdessen versucht, die Frage zu beantworten, welche Gestaltungsmöglichkeiten für das Arbeitsumfeld von Beschäftigten sich durch CPPS ergeben [GGH+13]. Nach Kagermann erfordert die Gestaltung von CPPS bezogen auf die Einbindung von Beschäftigten neue dezentrale Führungs- und Steuerungsformen, neue kollaborative Formen der Arbeitsorganisation mit einem hohen Maß an Selbstorganisation, einen verstärkten Aufbau entsprechender Systemkompetenz sowie eine wachsende technische Unterstützung auf Basis der Anpassung der Arbeit an den Menschen [KWH13]. Seine Erkenntnis hieraus ist, dass menschliche Flexibilität und Kreativität auch zukünftig nicht durch selbstorganisierte Systeme ersetzbar sein werden. Die zukünftige Aufgabe bestehe darin, die Fähigkeiten des Menschen durch den Einsatz von Informationstechnologie besser zu nutzen. Als Mittel hierfür werden neue, kollaborative Formen der Arbeitsorganisation genannt, bei denen der Mensch als aktiver Träger von Entscheidungen und Optimierungsprozessen agieren soll [DSS09].

Dabei ist es besonders auffällig, dass die Kapazität der Beschäftigten in der Forschung im Bereich der Fertigungssteuerung neben sozio-technischen Aspekten lange vernachlässigt wurde [Sch13a]. Für dieses Ergebnis hat der hohe Aufwand zur Einführung der Personalkapazitätssteuerung in der Fertigungssteuerung einen großen Teil beigetragen [SGG+13]. Das wurde in der Umfrage von Spath, et al. [SGG+13] von einem Drittel der befragten Unternehmen bestätigt. Darüber hinaus hat über die Hälfte

der Befragten angegeben, dass die kurzfristigen Kapazitätsschwankungen einen enormen Aufwand allein für die Umplanung der Beschäftigten verursachen können. Daher ist es notwendig, Beschäftigte flexibel einsetzen zu können. Eine Möglichkeit zur Flexibilisierung ist, wenn die Beschäftigten selbst die Steuerung übernehmen können. Botthoff et al. [BH16] sehen in der Selbstverwaltung gekoppelt mit der Einführung selbststeuernder Systeme eine wesentliche Forschungsfrage, die es bei der Einführung von CPPS zu erforschen gilt. Sie formulieren hierfür die folgenden Fragen:

- Wie können selbstorganisierte Produktionssysteme mit Mitbestimmung durch den Beschäftigten in Einklang gebracht werden?
- Wie kann in hochautomatisierten Umgebungen menschliche Kontrolle – Kontrolle des Menschen über seine Umwelt und die dort ablaufenden Prozesse – hergestellt werden?

Es existieren derzeit keine Arbeiten, die die Einbindung des Menschen bzw. Beschäftigten in CPPS vollumfänglich betrachten. Im Forschungsprojekt „Pro Mondi – Prospektive Ermittlung von Montagearbeitsinhalten in der Digitalen Fabrik“ wird zum Beispiel die Entwicklung einer Gesamtsystematik zur ganzheitlichen Unterstützung der integrierten Ermittlung von Montagearbeitsinhalten entlang der digitalen Produktentstehung betrachtet. Eine wesentliche Erkenntnis bildet hierbei die Möglichkeit der Verwendung der Datenanalyse, um eine durchgängige, digitale Planung von Montagearbeitsinhalten zu ermöglichen und dadurch die manuellen Montagetätigkeiten individuell zu optimieren [DEE+12].

Es gibt zudem zwei Forschungsprojekte, die die Integration des Menschen in CPPS betrachten. In dem Forschungsprojekt „CyProS“ ist zum Beispiel ein Arbeitspaket als Realisierung einer innovativen Möglichkeit der Mitarbeiterinteraktion zur Produktionsplanung und -steuerung in der CPS-basierten Produktion benannt. Weiterhin wurde die Mitarbeitereinbindung in CPPS im Rahmen von „CyProS“ untersucht [Ber15, WDF+14]. Ein weiteres Projekt, das sich mit der Einbindung des Menschen in CPPS beschäftigt, ist „Kapaflexcy“. Hierbei geht es vor allem um die Flexibilisierung der mittelfristigen Personal- und Kapazitätsplanung in Produktionssystemen. Die Beschäftigten können im Rahmen von Pilotanwendungen ihre eigenen Wunscheinsatzpläne definieren und mit Hilfe eines Onlinetools zur Terminfindung wird auf diese Wünsche bei der Planung eingegangen.

Die Aufarbeitung des Themas in Publikationen findet in der Regel durch die Betrachtung einzelner Aspekte der menschlichen Arbeitsleistung statt. Zentrale Forschungsthemen umfassen die Verbesserung der zeitlichen Einplanung, der Auftragsterminierung und der Ressourcenzuordnung unter Berücksichtigung menschlicher Einflüsse, wie z. B. Leistungsschwankungen [CDY+16]. Planung und Steuerung bezogen auf den Menschen wurden bereits in mehreren Publikationen auf holonische Produktionssysteme adaptiert [DAA17, EMM17]. Die Entwicklung zu CPPS ermöglicht eine Neugestaltung der Einbindung des Menschen über dessen Betrachtung als Ressource hinaus. Eine

Entwicklung von CPS mit Einbezug des Menschen wird hierbei als sozio-cyber-technisches System bezeichnet [BH16]. Glonegger [Glo14] beschreibt in seiner Dissertation z. B. umfassend die Planung unter Berücksichtigung menschlicher Leistungsschwankungen. Hecklau et al. [HGF+16] erläutern die Veränderungen zum Einbezug des Menschen in die Produktion durch Industrie 4.0. Hochdörffer et al. [HHL18] thematisieren die Zuweisung von Beschäftigten zu Arbeitsplätzen und Aufträgen unter Berücksichtigung ergonomischer Aspekte und der Qualifikation der Beschäftigten. Denkena [DDW+16] stellt darüber hinaus einen simulationsbasierten Ansatz zur Einbindung von Mitarbeiterkompetenzen in die Personaleinsatzplanung von Produktionssystemen vor. Sabar & Zenjair [SZ15] präsentierten bereits einen ersten allgemeinen Ansatz zur Berücksichtigung von Bedürfnissen in der Produktionsplanung. Josifovska et al. [JYE19a, JYE19b] präsentieren erstmalig Referenzmodelle zur Abbildung des Menschen ähnlich dem digitalen Zwilling für technische Systeme. Block et al. [BMD+16] beschreiben einen agentenbasierten Ansatz, wonach Menschen in der Produktion mit Elementen des Produktionssystems kommunizieren können.

Diese Forschungsprojekte und Arbeiten sind als ein erster Schritt in Richtung der Berücksichtigung der Bedürfnisse und Anforderungen von Beschäftigten zu deuten. Sie sind jedoch noch weit entfernt von einer tatsächlichen Implementierung im Rahmen eines PPS-Systems. Das zu entwickelnde PPS-System soll auf formal definierte Wünsche und Anforderungen von Beschäftigten bezüglich des Produktionsplans eingehen und diese in den Planungszyklen berücksichtigen. Beschäftigte sollen zudem die Möglichkeit erhalten, sie betreffende maschinelle Entscheidungsvorgänge zu beeinflussen und bei Entscheidungen zu intervenieren. Es existieren derzeit keine Publikationen, die eine solche Einbindung des Menschen behandeln. Daraus ergibt sich die Zielsetzung dieser Arbeit, ein Werkzeug zu entwickeln, sodass die Mitarbeiterin oder der Mitarbeiter trotz der automatisierten dezentralen Planung und Steuerung weiterhin eine Möglichkeit zur Beobachtung und Beeinflussung der Produktionssteuerung erhält.

3.3 Abgrenzung der Arbeit

In Abschnitt 3.1 wurde ausgehend von der Literaturrecherche die Forschungslücke abgeleitet: Die Entwicklung einer automatisierten dezentralen Produktionssteuerung ist erforderlich, um CPPS zur Selbstorganisation zu befähigen. Die Literaturrecherche war hierbei so aufgebaut, dass zuerst sich mit Hilfe von themenverwandten Lehrbüchern ein Überblick über die Thematik verschafft wurde und anschließend zu den in den Lehrbüchern erwähnten Verfahren ein aktueller Stand der Forschung abgeleitet wurde. Aktuelle Arbeiten zu diesen Themen wurden mit Hilfe von Suchmaschinen, der Durchsicht aktueller Konferenz- und Zeitschriftenbände und den Veröffentlichungen von Wissenschaftlern, die in diesem Themenbereich forschen, identifiziert.

Die Abgrenzung zu anderen Forschungsarbeiten aus diesem Bereich ist in der Abgrenzungsmatrix in Tabelle 3 verdeutlicht. Es wurde hierbei versucht unterschiedliche

Verfahren zu kategorisieren, die in einem direkten Zusammenhang zu dieser Arbeit stehen. Anschließend wurden versucht die aktuellsten, bzw. relevantesten Arbeiten für eine Kategorie zu finden und diese als Arbeit in die Abgrenzungsmatrix aufzunehmen. Die Relevanz wurde dabei inhaltlich und nach der Veröffentlichungsform ermittelt: Es wurde versucht vornehmlich Dissertationen und Beiträge aus bekannten Zeitschriften in die Matrix mit aufzunehmen. Die Abgrenzungsmatrix wird nach den beiden wesentlichen Aspekten des Lösungsansatzes dieser Arbeit – dezentrale Produktionssteuerung und digitale Repräsentation der Beschäftigten – unterteilt. Im Folgenden werden die Bewertungskriterien der Abgrenzungsmatrix vorgestellt.

- Der Aspekt der **dezentralen Produktionssteuerung für CPPS** konzentriert sich darauf, dass **individuelle Produkte** in CPPS mit der dezentralen Produktionssteuerung produziert werden. Diese grundlegende Anforderung an die dezentrale Produktionssteuerung wurde im Abschnitt 2.2 abgeleitet. Daraus ergeben sich die ersten drei Bewertungskriterien: Produktion individueller Produkte, dezentrale Steuerung und Fokussierung auf CPPS.

- Darüber hinaus soll die Produktionssteuerung die drei Hauptaufgaben von Produktionssteuerungen, also **Kapazitätssteuerung, Reihenfolgebildung und Auftragsfreigabe**, automatisiert behandeln. Diese Anforderung an die zu behandelnden Aufgaben der dezentralen Produktionssteuerung wurde in den Abschnitten 3.1.1 bis 3.1.3 vorgestellt. Daraus entsteht das Bewertungskriterium zur Behandlung dieser drei Steuerungsaufgaben: Kapazitätsteuerung, Reihenfolgebildung und Auftragsfreigabe.

- Durch die dezentrale Steuerung, die **regelbasiert** ist, können Produktionssysteme **die Selbstorganisation** erreichen. Die beiden Anforderungen an eine regelbasierte Steuerung und die Erreichung einer Selbstorganisation von Produktionssystemen wurden in den Abschnitten 3.1.4 und 3.1.5 durch eine Literaturrecherche analysiert und daraus abgeleitet. Die beiden Anforderungen entsprechen den letzten beiden Bewertungskriterien der dezentralen Produktionssteuerung.

- Der Aspekt der digitalen Repräsentation der Beschäftigten fokussiert sich darauf, dass **eine Eingriffsmöglichkeit in die Entscheidungen** durch Beschäftigte ermöglicht wird. Diese Anforderung wurde im Abschnitt 2.1 als Begriffsgrundlage schon erwähnt und im Abschnitt 3.2 wurde sie detailliert abgeleitet.

- Gleichzeitig müssen **die Bedürfnisse der Beschäftigten** berücksichtigt werden. Diese Anforderung wurde bereits in den Abschnitten 2.3 und 3.2 im Rahmen der Personaleinsatzplanung erwähnt.

- Um die Eingriffsmöglichkeit der Beschäftigten zu realisieren, muss das **Verhalten der Beschäftigten zur Entscheidungsfindung antizipiert** werden. Diese Anforderung entspricht einer vollständigen Visualisierung des Status von Beschäftigten (siehe Abschnitt 2.2 und 3.2).

- Die digitale Repräsentation muss **in der Realität umgesetzt** werden können.

In Tabelle 3 wird der Erfüllungsgrad der oben abgeleiteten Bewertungskriterien bewertet. Daraus ergibt sich der Neuheitsgrad dieser Arbeit. Hierbei wird nur auf die Forschungsarbeiten von Grundstein [Gru17] und Block et al. [BMD+16], die die meisten Bewertungskriterien der Abgrenzungsmatrix im Vergleich zu anderen Arbeiten erfüllt haben, eingegangen. Bezogen auf die Entwicklung der Produktionssteuerung hat Grundstein [Gru17] mit seiner entwickelten selbstorganisierten Fertigungsregelung eine integrierte Lösung für die Steuerung von CPPS geschaffen. Zudem nutzt er einen regelbasierten Ansatz zur Steuerung. Damit können Produktionssysteme zur Selbststeuerung befähigt werden. Grundstein nimmt jedoch eine klare Trennung zwischen der Steuerung und der Planung des Produktionssystems vor. Das heißt, dass er die Auftragsumplanung, die als der entscheidende Faktor zur Ausschöpfung der Produktionskapazität gilt, in seiner Arbeit nicht betrachtet hat. Das entspricht der in Abschnitt 3.1.5 beschriebenen Forschungslücke, die für die Erreichung der Selbstorganisation des Produktionssystems benötigt wird. Zudem unterscheiden sich die von ihm ausgewählten Verfahren wesentlich von den Verfahren dieser Arbeit. Auch wird keine Betrachtung der Beschäftigten über eine Ressource hinaus vorgenommen. Die einzige Möglichkeit, die Grundstein [Gru17] den Beschäftigten verleiht, besteht in der Veränderung der Steuerungsparameter. Aus diesem Grund sind die Möglichkeiten der Einflussnahme von Beschäftigten auf die Produktionssysteme stark eingeschränkt. Die Eingriffsmöglichkeit durch die Beschäftigten wird in der Arbeit von Block et al. [BMD+16] thematisiert. Diese bildet jedoch nur eine Visualisierung des Status und der Entscheidungen ab und implementiert keine Einflussnahme durch die Beschäftigten. Eine Begründung für die wenigen menschenzentrierten Forschungsarbeiten in diesem Bereich liefert Schuh und konstatiert, dass die Kapazität der Beschäftigten neben sozio-technischen Aspekten in der Forschung im Bereich der Fertigungssteuerung lange vernachlässigt worden sei [Sch12]. Aus diesen Forschungslücken, welche auch in der Abgrenzungsmatrix in Tabelle 3 dargestellt sind, wurden die Forschungsfragen abgeleitet. Mit Hilfe der vorliegenden Arbeit sollen die folgenden Forschungsfragen untersucht und beantwortet werden:

1. Wie muss ein Verfahren aufgebaut sein, mit dem eine Selbststeuerung von CPPS erreicht wird?

2. Wie lassen sich durch ein dieses Verfahren insbesondere Umplanungen von der Produktionssteuerung automatisiert bewerkstelligen?

3. Wie kann bei einem solchen automatisierten Planungs- und Steuerungssystem den Beschäftigten eine Eingriffsmöglichkeit in die von den Computersystemen getroffenen Entscheidungen gegeben werden?

4. Wie können die Bedürfnisse von Beschäftigten im Rahmen der Planung und Steuerung automatisiert berücksichtigt werden?

Tabelle 3: Abgrenzungsmatrix

		Dezentrale Produktionssteuerung für CPPS						Digitale Repräsentation der Beschäftigten in PPS			
		Produktion individueller Produkte	Dezentrale Steuerung	Kapazitätssteuerung, Reihenfolgebildung, Auftragsfreigabe	Fokussierung auf CPPS	Regelbasierte Steuerung	Erreichung einer Selbstorganisation	Eingriffsmöglichkeit durch die Beschäftigten	Berücksichtigung von Bedürfnissen der Beschäftigten	Antizipation des Verhaltens zur Entscheidungsfindung	Umsetzung einer digitalen Repräsentation
PPS	[SDC+18]	◐	●	◐	◐	—	—	—	—	—	—
	[Van08]	◐	○	—	—		—	—	—	—	—
	[Eng15]	●	◐	●	◐	◐	—	—	—	—	—
	[Ise17]	○	◐	○	—	◐	—	○	—	—	
	[Ost12]	○	◐	●	—	—	—	—	—	—	—
	[Löd13]	◐	◐	●	○	◐	—	○	—	—	—
	[DLS12]	○	○	—	—	—	—	—	—	—	—
	[Rai10]	◐	●	—	—	◐	—	—	—	—	—
	[JBP+14]	○	○	—	○	—	—	—	—	—	—
	[LFB16]	○	○	—	—	—	—	—	—	—	—
	[Men17]	◐	○	—	—	—	—	—	—	—	—
	[Trz15]	◐	○	—	—	—	—	—	—	—	—
	[Nie16]	◐	○	●	◐	◐	—	○	—	—	—
	[Gru17]	◐	○	●	●	●	—	○	—	—	—
Repräsentation	[Glo15]	—	—	—	—	—	—	○	—	—	—
	[Tie17]	—	—	—	—	—	—	○	—	—	—
	[Den16]	—	—	—	—	—	—	○	—	—	—
	[SZ15]	—	—	—	—	—	—	○	—	—	—
	[Ber15]	—	—	—	—	—	—	○	○	—	—
	[BMD+16]	—	—	—	—	—	—	◐	—	—	—
Neuheitsgrad dieser Arbeit							X	X	X	X	X

— Nicht behandelt
○ Übertragung möglich
◐ Teilweise behandelt
● Vollumfänglich behandelt
× Neuheitsgrad

4 Entwicklung einer automatisierten dezentralen Produktionssteuerung für CPPS

In Abschnitt 3.3 wurde der Forschungsbedarf durch zwei Aspekte beschrieben: automatisierte dezentrale Produktionssteuerung für CPPS und digitale Repräsentation der Beschäftigten. In diesem Kapitel wird auf die Entwicklung der automatisierten dezentralen Produktionssteuerung für CPPS eingegangen. Dafür werden zuerst Anforderungen definiert, die auf den Bewertungskriterien in der Abgrenzungsmatrix beruhen. Auf dieser Grundlage wird das neue Verfahren zur automatisierten dezentralen Steuerung eines Produktionssystems entwickelt.

Die vier logistischen Zielgrößen der PPS sind eine hohe Termintreue, kurze Durchlaufzeiten, hohe Auslastung und geringe Bestände (siehe Abschnitt 2.2). Hieraus werden die folgenden Anforderungen abgeleitet:

- Die logistischen Zielgrößen werden hierbei als die übergeordnete Zielstellung für die Entwicklung der Produktionssteuerung festgelegt: Die Fertigstellungstermine müssen eingehalten werden, während der Bestand auf einem vordefinierten Niveau geregelt wird. Durchlaufzeiten und Auslastung werden durch die Bestandsregelung indirekt beeinflusst.
- Eine hohe Termintreue setzt eine umsichtige Terminplanung voraus. In Abschnitt 3.1.2 wurde die Auswirkung der Reihenfolge der zu bearbeitenden Aufträge auf die Termintreue erläutert. Abhängigkeiten zwischen Aufträgen sollen hierbei berücksichtigt werden. Dies können zum Beispiel auftragsabhängige Rüstzeiten sein.
- Für möglichst kurze Durchlaufzeiten und eine möglichst hohe Auslastung soll die Arbeitsverteilung in der Produktion möglichst gleichmäßig auf parallele Stationen erfolgen.

In Abschnitt 3.1 wurden die drei wesentlichen Aufgaben einer Produktionssteuerung vorgestellt: Kapazitätssteuerung, Reihenfolgebildung und Auftragsfreigabe. Im Folgenden werden die Anforderungen an diese vorgestellt:

- Die Kapazitätssteuerung zielt auf die Kompensation von Überlast und Unterlast der Arbeitsstationen. Sie wird sowohl periodisch als auch ereignisorientiert angestoßen. Periodisch errechnet die Kapazitätssteuerung, ob die Fertigstellungstermine der Aufträge an den jeweiligen Stationen eingehalten werden. Auf Grundlage dieser Kalkulationsergebnisse wird die Kapazität jeder Station iterativ bis zum jeweiligen Minimum/Maximum abgesenkt/erhöht, sodass alle Termine mit der geringstmöglichen Kapazität eingehalten werden. Zudem werden kapazitätsverändernde Maßnahmen erst ergriffen, wenn Produktionskonflikte nicht durch eine Umplanung beseitigt werden können.
- Für die Bildung der Reihenfolge (siehe Abschnitt 3.1.2) werden die vergangenen Bearbeitungszeiten berücksichtigt. Bezüglich der Reihenfolge der Warteschlange vor

einer Arbeitsstation wird geprüft, ob bereits bei diesen Aufträgen Terminabweichungen entstehen.

- Die Auftragsfreigabe gibt die Aufträge frei, sobald der geplante Freigabetermin erreicht ist oder der Bestand der Aufträge an einer Arbeitsstation unter eine vorher festgelegte Bestandsgrenze fällt. Hierbei sind die Freigabetermine vorher definiert oder diese werden anhand eines geplanten Fertigstellungstermins berechnet. Die Aufträge ohne Wunschtermine oder Eilaufträge werden mit entsprechender Priorisierung anderen Aufträgen vorgezogen.

Ein entscheidender Neuheitsgrad in dieser Arbeit liegt darin, dass die Produktionssteuerung die Auftragsumplanung ermöglicht. Die Anforderung an die Auftragsumplanung ist hierbei wie folgt:

- Bei Umplanungen sollen keine angefangenen Aufträge unterbrochen oder aufgeteilt werden. Unterbrechungen oder Änderungen an Aufträgen können nur manuell vorgenommen werden.

Aufbauend auf diesen Anforderungen wird in dem nächsten Abschnitt das grundlegende Konzept der entwickelten Produktionssteuerung vorgestellt. Zusätzlich werden diese Anforderungen als Referenz für die Validierung in Kapitel 7 genommen.

4.1 Konzept der automatisierten dezentralen Produktionssteuerung

In diesem Abschnitt wird das Konzept der automatisierten dezentralen Produktionssteuerung entwickelt. Im Folgenden werden zuerst die Ansätze zur Erreichung der Zielstellung vorgestellt. Daraus werden die zwei übergeordneten Hauptaufgaben der automatisierten dezentralen Produktionssteuerung zusammengefasst und Lösungsansätze hierfür vorgestellt. Anschließend wird das Verfahren zur „Auftragsumplanung“, das einen wesentlichen Neuheitsgrad der entwickelten dezentralen Produktionssteuerung darstellt, detailliert beschrieben. Darauf aufbauend wird das Gesamtkonzept der automatisierten dezentralen Produktionssteuerung vorgestellt.

Übergeordnete Zielstellung der dezentralen Produktionssteuerung

Die übergeordnete Zielstellung der dezentralen Produktionssteuerung liegt in der **Einhaltung von Fertigstellungsterminen**. Bei der Berechnung der Termineinhaltung werden Vergangenheitsdaten, wie z.B. die Bearbeitungszeiten, berücksichtigt. Die Arbeitsverteilung in der Fertigung erfolgt unter Berücksichtigung der Fertigstellungstermine, mit dem Ziel, die Arbeit gleichmäßig auf parallele Stationen zu verteilen und so kurze Durchlaufzeiten zu erreichen. Aufträge werden blockiert, falls beispielsweise durch Rückflüsse die Bestandsgrenze einer Station überschritten wird. In einem solchen Fall wird die Kapazitätssteuerung angestoßen, um die Überlastung zu kompensieren. Die Kapazitätssteuerung wird ebenso bei geringer Auslastung angestoßen,

um eine möglichst hohe Auslastung zu gewährleisten. Die Reihenfolgebildung zielt auf die Einhaltung der Fertigstellungstermine, berücksichtigt aber auch reihenfolgeabhängige Rüstzeiten und versucht, diese zu reduzieren, solange Fertigstellungstermine dadurch nicht gefährdet werden. Bei der Berechnung von Bearbeitungs- und Rüstzeiten werden Vergangenheitsdaten ebenso wie Plandaten aus den von der Produktionsplanung berechneten Arbeitsplänen berücksichtigt. Die Reihenfolge der Warteschlange vor einer Arbeitsstation wird zusammen mit den voraussichtlichen Terminabweichungen bei jedem Zu- und Abgang eines Auftrags berechnet.

Um die Einhaltung von Fertigstellungsterminen zu gewährleisten, sollen Engpässe möglichst hoch ausgelastet werden. Die Bestände von Aufträgen werden auf einem gleichbleibenden Niveau geregelt. Alle Berechnungen werden von den einzelnen Stationen selbst vorgenommen, wobei die dort hinterlegten Vergangenheitsdaten als Grundlage für die Berechnung herangezogen werden. Über die Produktionsplanung sind die Aufträge bereits den Stationen zugewiesen. Sie werden zuerst auf den Stationen verteilt und anschließend eingelastet. Ein neuer Auftrag wird nach dem Freigabetermin an der Station freigegeben. Die Freigabe geschieht, sobald die Aufträge unter eine bestimmte Bestandsgrenze fallen und alle Voraussetzungen für die Ausführung des Auftrags erfüllt sind. Alle Aufträge unterhalb der Bestandsgrenze einer Station werden unmittelbar freigegeben und die Auftragsreihenfolge wird geprüft und gegebenenfalls geändert. Gleichartige Stationen sprechen sich hierbei vor der Freigabe eines neuen Auftrags untereinander ab, um eine möglichst gleichmäßige Verteilung von Aufträgen zu erreichen. Nach der Freigabe des Auftrags wird dieser lokal in die Auftragsreihenfolge der Station einsortiert. Anschließend wird die Erfüllung der Anforderungen im Rahmen einer ständigen Kapazitätssteuerung überwacht. Die Maßnahmen, die hieraus ergriffen werden können, umfassen eine Umplanung der Aufträge oder die Erhöhung der Kapazität (zum Beispiel durch die Veranlassung einer Überstunde).

Übergeordnete Aufgaben der dezentralen Produktionssteuerung

Zur Erreichung dieser Zielstellung muss die automatisierte dezentrale Produktionssteuerung Aufträge automatisiert auf Stationen einlasten und anschließend die Ausführung kontinuierlich überwachen.

Ausgangspunkt der Produktionssteuerung ist, dass bereits Reihenfolgepläne für die einzelnen Stationen vom ERP-System oder MES erstellt wurden. Die Einlastung dieser Reihenfolgepläne auf den Stationen wird von der in dieser Arbeit entwickelten Produktionssteuerung automatisiert. Bei der Einlastung werden die Auftragsreihenfolgepläne überprüft und gegebenenfalls angepasst bzw. umgeplant. Nach der Einlastung findet anschließend die automatisierte Steuerung statt, also die automatisierte Ausführung der Aufträge und die eigenständige Überwachung der Produktion. Sollten kurzfristige Veränderungen (z. B. auf Grund von Störungen) an den Auftragsreihenfolgeplänen notwendig werden, wird eine Umplanung eingeleitet. Basierend auf der Automatisierungspyramide (siehe Abschnitt 2.2) ist der

Betrachtungsraum der dezentralen Produktionssteuerung in der folgenden Abbildung 17 veranschaulicht. Die Aufgaben von der dezentralen Steuerung sind hierbei in grau markiert.

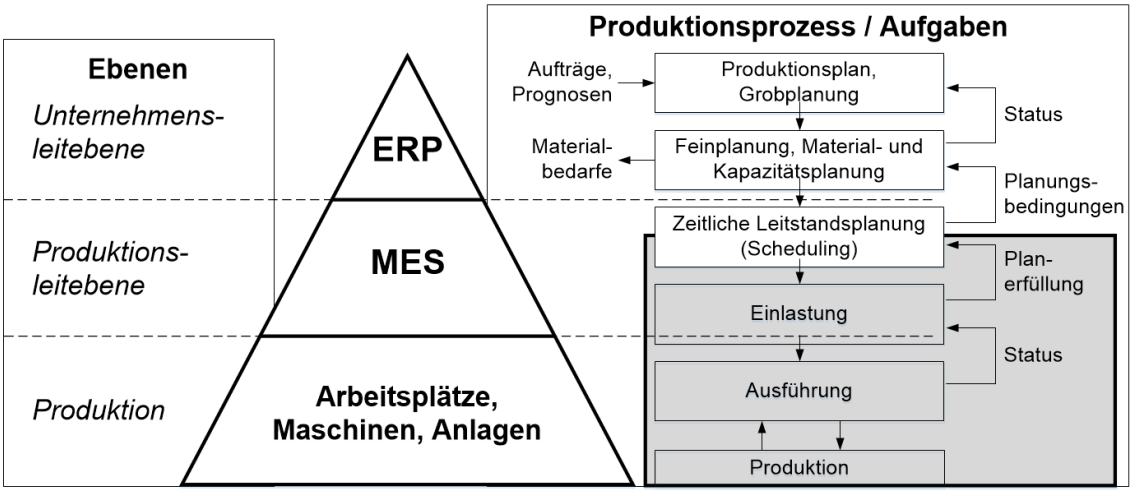

Abbildung 17: Betrachtungsraum der dezentralen Produktionssteuerung

Die Vorstellung der Steuerungsverfahren findet am Beispiel unverketteter Produktionssysteme statt. Die Potenziale der in dieser Arbeit entwickelten Produktionssteuerung haben in unverketteten Produktionssystemen deutlich größere Auswirkungen als in verketteten Produktionssystemen. Prinzipiell ließe sich das Steuerungssystem in allen Produktionsorganisationsformen einsetzen. Bei stark verketteten Produktionssystemen wie zum Beispiel der Fließfertigung ergeben sich jedoch nur geringe Spielräume für Produktionsoptimierungen. Die Restriktionen hinsichtlich der Verkettung sind zu groß, um wirksame Umplanungen umsetzen zu können. So können Aufträge bei verketteten Systemen in der Regel nicht auf andere Stationen verteilt werden. Erst bei voneinander abgekoppelten Systemen wie z.B. bei CPPS, in denen die einzelnen Produktionsanlagen selbstständig agieren können, ergeben sich Potenziale für den Einsatz einer dezentralen Steuerung. Dies ist zum Beispiel bei Produktionssystemen der Fall, die nach dem Werkstattprinzip organisiert sind. Hierin können die einzelnen Produktionselemente Aufträge weitgehend unabhängig voneinander ausführen, weshalb eine dezentrale Optimierung entsprechend große Potenziale bietet.

Nachfolgend wird der Betrachtungsraum der dezentralen Produktionssteuerung vorgestellt. Die Produktionsplanung erstellt eine Auftragsreihenfolge und gibt diese zur Ausführung an die Produktionssteuerung weiter. Die Produktionssteuerung lastet diese Auftragsreihenfolgen anschließend entsprechend auf den Stationen ein. Nach einem gewissen Zeitraum wird die Produktion erneut geplant und die daraus resultierenden Auftragsreihenfolgepläne erneut eingelastet. In Abbildung 18 oben werden diese Neuplanungen als vier Planungsperioden dargestellt. Im Rahmen der Produktionssteuerung werden jedoch nur kurzfristig auszuführende Aufträge betrachtet.

Dies ist in Abbildung 18 als Steuerungshorizont dargestellt. Aufträge, die sich im Rahmen des Steuerungshorizonts befinden, werden an die Arbeitsstationen weitergegeben. Alle Aufträge danach werden zuerst im Auftragsspeicher belassen. Dadurch kann der Berechnungsaufwand stark reduziert werden.

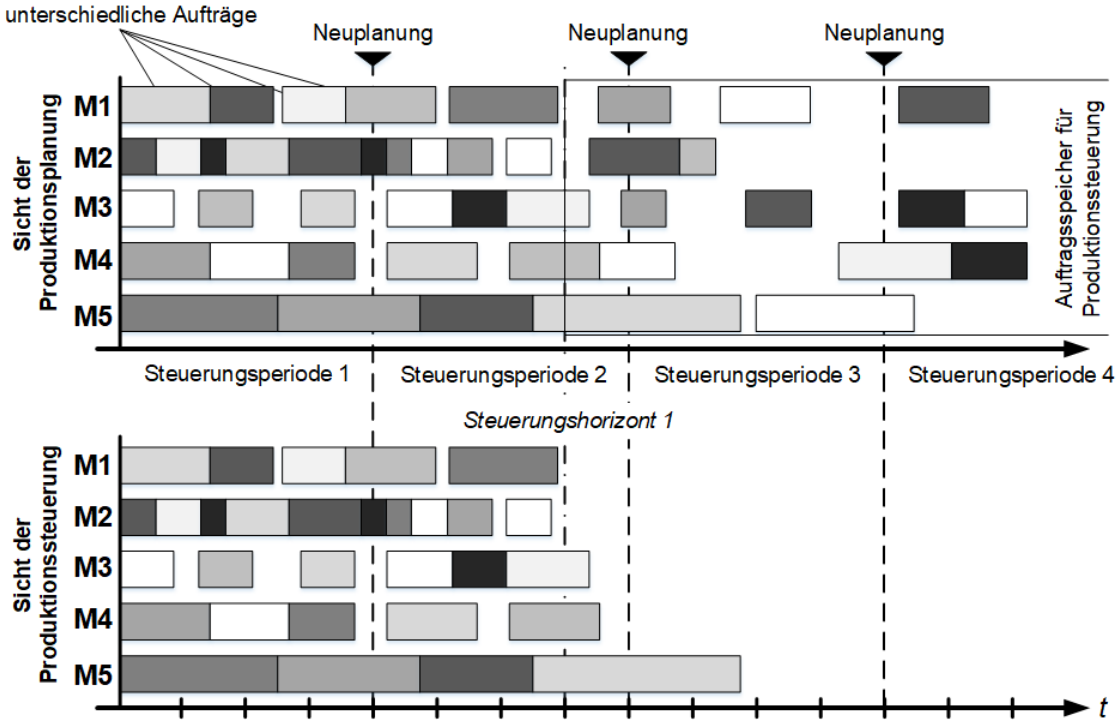

Abbildung 18: Betrachtungszeitraum der Produktionssteuerung

Sobald ein Auftrag an einer Station abgeschlossen wurde oder sich Veränderungen an der Produktionssituation ergeben, wird geprüft, ob eine Umplanung notwendig ist. Veränderungen, die zu einer Umplanung führen, sind im Wesentlichen die Folgenden:

- Fertigstellungszeitpunkt eines Auftrags wird nicht eingehalten
- Eine Anlage fällt aus oder benötigt länger als geplant für einen Auftrag und erzeugt dadurch eine Verzögerung
- Allgemeine Verzögerung im Produktionsablauf
- Ungeplante Pausenzeiten
- Verzögerungen durch Leistungsschwankungen, z. B. hervorgerufen durch unterschiedliche Montagezeiten von Beschäftigten

Um den Ansatz zur Umplanung genauer vorzustellen, wird nachfolgend der gesamte Produktionssteuerungsprozess anhand eines Produktionsszenarios beschrieben, in dem eine Umplanung durchgeführt wurde.

Szenariobeschreibung

Zuerst wird das Produktionsszenario anhand des Produktionsauftrags A10 in der Abbildung 19 veranschaulicht. Die Auftragsstruktur des Auftrags A10 erklärt den gesamten Produktionsprozess: zwei Fräskomponenten werden von den beiden Produktionsaufträgen A4 und A5 gefertigt. Dann folgt die Vormontage, die dem Auftrag A10 entspricht. Nach dem Abschluss von A10 wird die Endmontage durch den Produktionsauftrag A12 ausgeführt.

Beschreibung des Steuerungsprozesses für den Produktionsauftrag A10

In diesem Fall übernimmt die dezentrale Steuerung die Aufgabe, den Auftrag A10 auf Stationen einzulasten, auszuführen und den Fortschritt zu überwachen, um den Fertigungstermin einzuhalten. Falls der Fertigungstermin von Auftrag A10 nicht eingehalten werden kann, wird eine automatisierte Umplanung durch die dezentrale Produktionssteuerung durchgeführt. Dafür muss die dezentrale Produktionssteuerung den Fertigungszeitpunkt des Auftrags A4 und A5, den frühesten Startzeitpunkt des Auftrags A10, den spätesten Fertigstellungszeitpunkt des Auftrags A10 und den Starttermin von Auftrag A12 ständig berücksichtigen. Die folgende Abbildung 19 zeigt, dass der Auftrag A10 sich an der Arbeitsstation „Vormontage_2“ befindet. Dabei könnte der Produktionsauftrag A10 ebenso auf den anderen beiden Vormontagestationen 1 und 3 ausgeführt werden. Allerdings sind die Aufträge A9 und A11 bereits an der Station Vormontage 1 und Vormontage 3 zugewiesen, die gleichzeitig mit dem Produktionsauftrag A10 ausgeführt werden. All diese Randbedingungen müssen von der dezentralen Steuerung eingehalten werden.

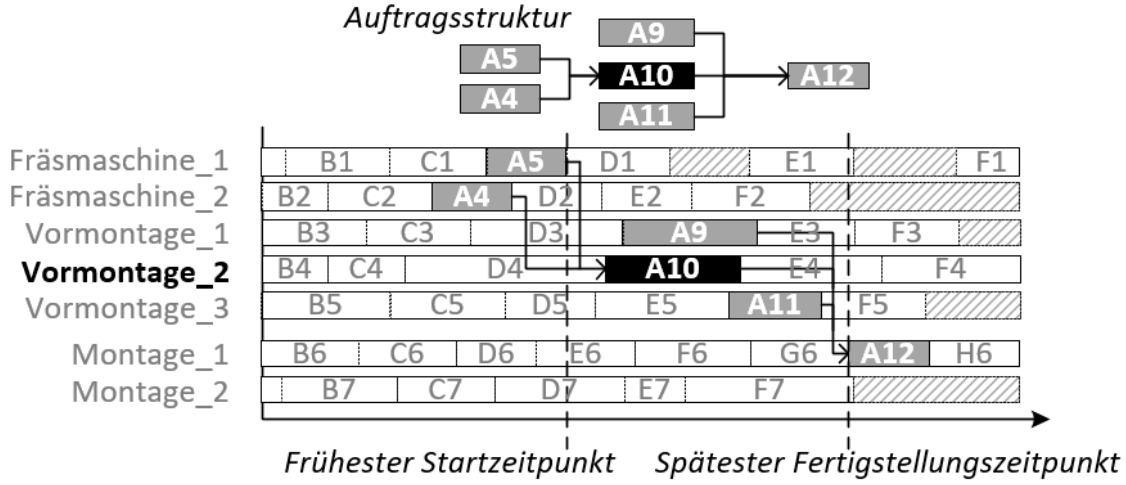

Abbildung 19: Einflüsse von anderen Produktionsaufträgen auf Produktionsauftrag A10

An der Station „Vormontage_2“ muss gewährleistet werden, dass sich alle Materialien für den Auftrag A10 an der Station befinden, also A5 und A4 abgeschlossen sind und deren Komponenten zu A10 transportiert sind. Gleichzeitig muss sichergestellt werden, dass der späteste Fertigstellungszeitpunkt von A10 nicht überschritten wird. Vor der

Einlastung auf der Station muss der Auftrag A10 für die Bearbeitung freigegeben werden. Nach der Freigabe von A10 auf die Station „Vormontage_2“ wird eine neue Reihenfolgebildung (siehe Abschnitt 3.1.2) an der Station durchgeführt. Im laufenden Betrieb findet dann im Rahmen der Kapazitätssteuerung (siehe Abschnitt 3.1.1) eine ständige Überwachung der Aufträge und deren Randbedingungen statt. Treten kurzfristige Änderungen auf, benötigt also Fräsmaschine_1 beispielsweise länger für den Produktionsauftrag A5, muss schnell eine Entscheidung darüber getroffen werden, ob eine Umplanung notwendig ist. Zwischen dem frühesten Starttermin und dem spätesten Fertigstellungszeitpunkt kann die Arbeitsstation den Produktionsauftrag eigenständig verschieben und anpassen. Darüber hinaus muss eine Koordination mit den vorhergehenden bzw. nachfolgenden Stationen erfolgen. Dies bildet die Grundidee des Konzepts der automatisierten dezentralen Produktionssteuerung.

Entwicklung des Konzepts der Produktionssteuerung

Die dezentrale Produktionssteuerung baut somit auf den Vorgaben der Produktionsplanung auf. Die Vorgaben von der Produktionsplanung sind Produktionspläne, in denen die Zuordnung von Aufträgen zu Arbeitsstationen vorgegeben wird. Darüber werden ebenso die einzelnen Start- und Endzeitpunkte der einzelnen Operationen bestimmt. Zusätzlich werden die prozessbezogenen Informationen wie z.B. Start- und Fertigstellungszeitpunkte der einzelnen Aufträge und die produktbezogenen Informationen wie z.B. Stücklisten und Arbeitspläne von der Produktionsplanung weitergegeben. Für die Produktionssteuerung ist es unerheblich, welche Methode zur Erstellung des Produktionsplans verwendet wird, solange diese Informationen zur Verfügung gestellt werden.

Nach dieser Übergabe aller Planwerte übernimmt die dezentrale Produktionssteuerung die folgenden Prozesse: Auftragseinlastung, ständige Überwachung, Auftragsausführung und unter Umständen Auftragsumplanung an einer Arbeitsstation. Die Einlastung der Aufträge wird in der Regel bereits durch die Planung vorgegeben. Im Rahmen der Produktionssteuerung wird die Plausibilität der Pläne noch einmal geprüft. Die dezentrale Produktionssteuerung führt hierbei eine ständige periodische Überwachung nach einem vordefinierten Steuerungshorizont durch (siehe Abbildung 20 oben links). Die Aufträge außerhalb des Steuerungshorizonts werden in dem Auftragsspeicher belassen. Bei Störungen oder Verzögerungen wird die dezentrale Steuerung Maßnahmen zur Umplanung ergreifen, wie z.B. die Optimierung der Auftragsverteilung bzw. der Kapazitätsanpassung (siehe Abschnitt 3.1.1). Anschließend folgt die Auftragsausführung bis zur Fertigstellung des Auftrags. Während der Auftragsausführung findet ein kontinuierlicher Austausch zwischen den Arbeitsstationen statt, um sicherzustellen, dass alle Anforderungen und insbesondere die Fertigstellungszeitpunkte eingehalten werden (siehe Abbildung 20 rechts). Gleichzeitig versuchen die einzelnen Arbeitsstationen dabei, die Produktionspläne, die bereits auf den einzelnen Stationen freigegeben sind, weiter zu optimieren und zum Beispiel unerwünschte Pausen- oder Instandhaltungszeiten zu

eliminieren. Daraus entsteht das Gesamtkonzept der dezentralen Produktionssteuerung, das in Abbildung 20 veranschaulicht ist.

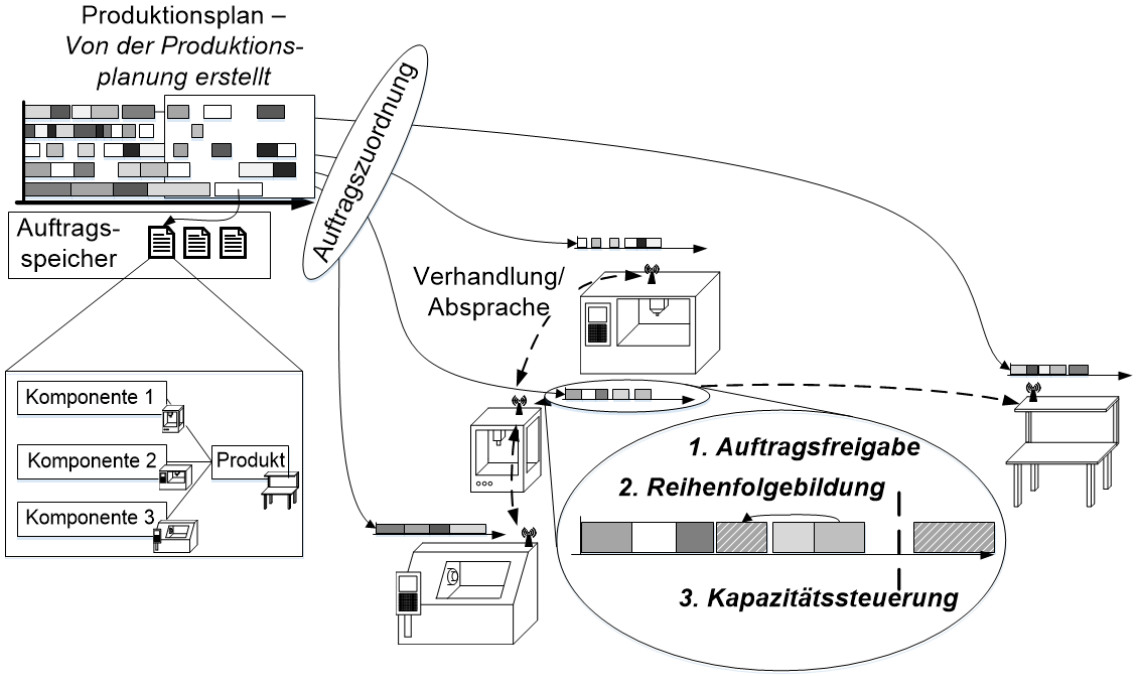

Abbildung 20: Gesamtkonzept der dezentralen Produktionssteuerung

4.2 Entwicklung der automatisierten dezentralen Produktions-steuerung in CPPS

Die dezentrale Produktionssteuerung besteht aus den folgenden Bausteinen.

- Automatisierter Prozess zur Einlastung von Aufträgen:

  Die Einlastung beschreibt den Prozess von der Freigabe eines Auftrags an einer Arbeitsstation über die Auftragsüberwachung bis hin zu einer eventuellen Umplanung. Die dezentrale Produktionssteuerung übernimmt den Produktionsplan von der Produktionsplanung. Darin ist die Zuweisung der Aufträge zu den Arbeitsstationen enthalten. Die Einlastung wird von der dezentralen Produktionssteuerung auf Plausibilität geprüft. Insbesondere wird geprüft, ob alle Fertigstellungstermine eingehalten werden können. Für den Einlastungsprozess spielen die Reihenfolgebildung und Auftragsfreigabe eine bedeutende Rolle zur Umsetzung der automatisierten Einlastung.

- Das Alleinstellungsmerkmal der dezentralen Produktionssteuerung: automatisierte Umplanung der Auftragsreihenfolge:

  Die Besonderheit der in dieser Arbeit entwickelten dezentralen Produktionssteuerung ist die Umsetzung der automatisierten Auftragsumplanung. Dadurch ist die Produktionssteuerung in der Lage, wenn z.B. Störungen oder sich ändernde

Rahmenbedingungen auftreten, eine automatisierte Umplanung zur Optimierung der Auftragsreihenfolge bzw. der Auftragsverteilung durchzuführen. Die Umplanung wird dabei entweder bei der Einlastung eines neuen Auftrags oder durch die Kapazitätssteuerung ausgelöst.

- Realisierung einer dezentralen Produktionsstruktur zur Ermöglichung der untereinander stattfindenden Kommunikation bzw. Verhandlungen zwischen den Arbeitsstationen:

  Die dezentrale Umsetzung der Produktionssteuerung spielt eine wesentliche Rolle in der Ausschöpfung der Potenziale von CPPS. Dadurch können Arbeitsstationen z.B. eigenständig Prozesszeiten ermitteln und diese für eine selbstständige Planung nutzen.

In diesem Abschnitt werden die Verfahren zur Behandlung der oben vorgestellten Bausteine entwickelt, aus denen die automatisierte dezentrale Produktionssteuerung besteht.

4.2.1 Das entwickelte Verfahren zur Einlastung

Die dezentrale Produktionssteuerung übernimmt den Produktionsplan und überprüft im ersten Schritt dessen Plausibilität. Die Plausibilitätsprüfung umfasst dabei den Abgleich zwischen den von der Produktionsplanung vorgegebenen und den von der Produktionssteuerung berechneten Fertigstellungszeiten. Wenn keine Fertigstellungstermine von der Produktionsplanung vorgegeben werden, wird die dezentrale Produktionssteuerung diese durch Kommunikation unter den Arbeitsstationen berechnen. Werden alle Fertigstellungszeiten eingehalten, werden die Aufträge einfach übernommen. Wenn nicht, wird bereits bei der Einlastung der Umplanungsprozess ausgelöst. Diese Plausibilitätsprüfung findet über die Betrachtung des frühesten Startzeitpunkts und des spätesten Endzeitpunkts jedes Auftrags statt. Dieser Zusammenhang wird in der folgenden Abbildung 21 veranschaulicht.

Zuerst werden eine Bestandsgrenze und der Steuerungshorizont für jede Arbeitsstation festgelegt. Beides sind Zeitwerte, die angeben, wie viele Aufträge betrachtet werden. Die Bestandsgrenze dient dazu, den Bestand an einer Arbeitsstation und damit indirekt den Bestand an Aufträgen in dem Produktionssystem auf einem definierten Niveau zu regeln. Wird die Bestandsgrenze niedrig gesetzt, verringern sich die Umlaufbestände und dadurch in der Regel auch die Durchlaufzeiten und auch die Bestandskosten. Zu geringe Bestandsgrenzen können jedoch dafür sorgen, dass nachfolgende Arbeitsstationen auf vorgehende oder nachfolgende Prozesse warten müssen, da nicht genügend Bauteile oder Baugruppen zur Weiterverarbeitung an den Arbeitsstationen vorhanden sind. Zur Gewährleistung einer hohen Liefertreue eignen sich deswegen höhere Bestandsgrenzen. Es ist daher eine Abwägung, die richtigen Bestandsgrenzen zu wählen. Der Steuerungshorizont definiert hingegen, bis wohin die Steuerung Aufträge betrachtet und

ggf. vorab eingeplant werden. Der Steuerungshorizont muss daher stets länger als die Bestandsgrenze gewählt werden. Beide Werte können zwar für alle Arbeitsstationen einzeln definiert werden, es wird jedoch davon ausgegangen, dass für alle Arbeitsstationen dieselben Werte eingetragen werden.

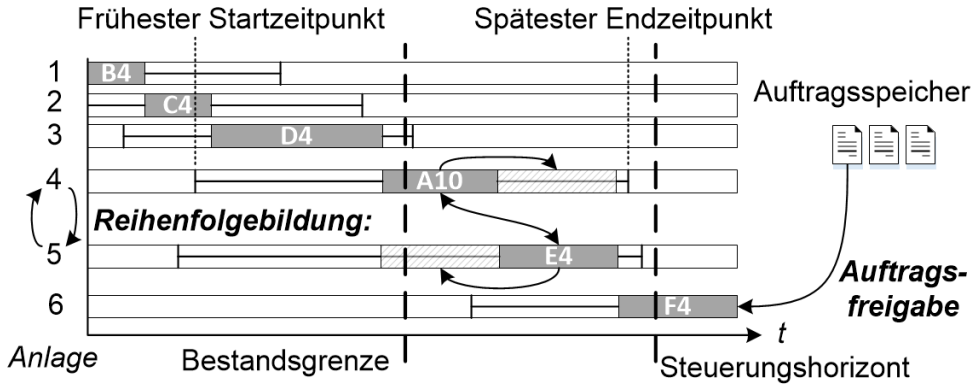

Abbildung 21: Prozess zur Auftragsfreigabe von dem Auftragsspeicher auf die Warteschlange einer Arbeitsstation

Die Abbildung 21 veranschaulicht den Prozess zur Auftragsfreigabe von dem Auftragsspeicher auf die Warteschlange einer Arbeitsstation. Dabei wird der Produktionsauftrag A10 als Zielauftrag betrachtet. Die in der Abbildung gezeigte Warteschlange der Arbeitsstation ist mit der ursprünglichen Auftragsreihenfolge, die von der Produktionsplanung weitergegeben wurde, dargestellt. Darin sind zudem die folgenden wesentlichen Informationen zu finden: der früheste Startzeitpunkt für A10, der späteste Endzeitpunkt für A10, die Bestandsgrenze von Aufträgen an der Arbeitsstation und der Steuerungshorizont.

Die erste Möglichkeit der Auftragsfreigabe ist, dass keine Veränderungen an dieser Arbeitsstation vorkommen. Dann wird der Produktionsauftrag A10 nach der von der Produktionsplanung vorgegebenen Reihenfolge eingeplant. Gleichzeitig wird ein neuer Produktionsauftrag vom Auftragsspeicher an das Ende der Warteschlange freigegeben.

Die zweite Möglichkeit beschreibt die Verschiebung des Produktionsauftrags A10. Die Verschiebung des Auftrags A10 darf nur in dem Bereich zwischen dem vorgegebenen frühesten Startzeitpunkt und dem spätesten Endzeitpunkt geschehen. In der Abbildung ist die Verschiebung nach hinten als Tausch mit dem Produktionsauftrag E4 illustriert. Der gesamte Prozessablauf zeigt, dass die Verschiebung der Aufträge nur unter Berücksichtigung der vorhandenen Restriktionen bzw. Randbedingungen von der Produktionssteuerung durchgeführt werden darf. Bereits gestartete Aufträge dürfen hierbei nicht von der automatisierten dezentralen Produktionssteuerung angepasst werden.

Das entwickelte Verfahren zur Auftragsfreigabe wird in der nachfolgenden Abbildung 22 verdeutlicht.

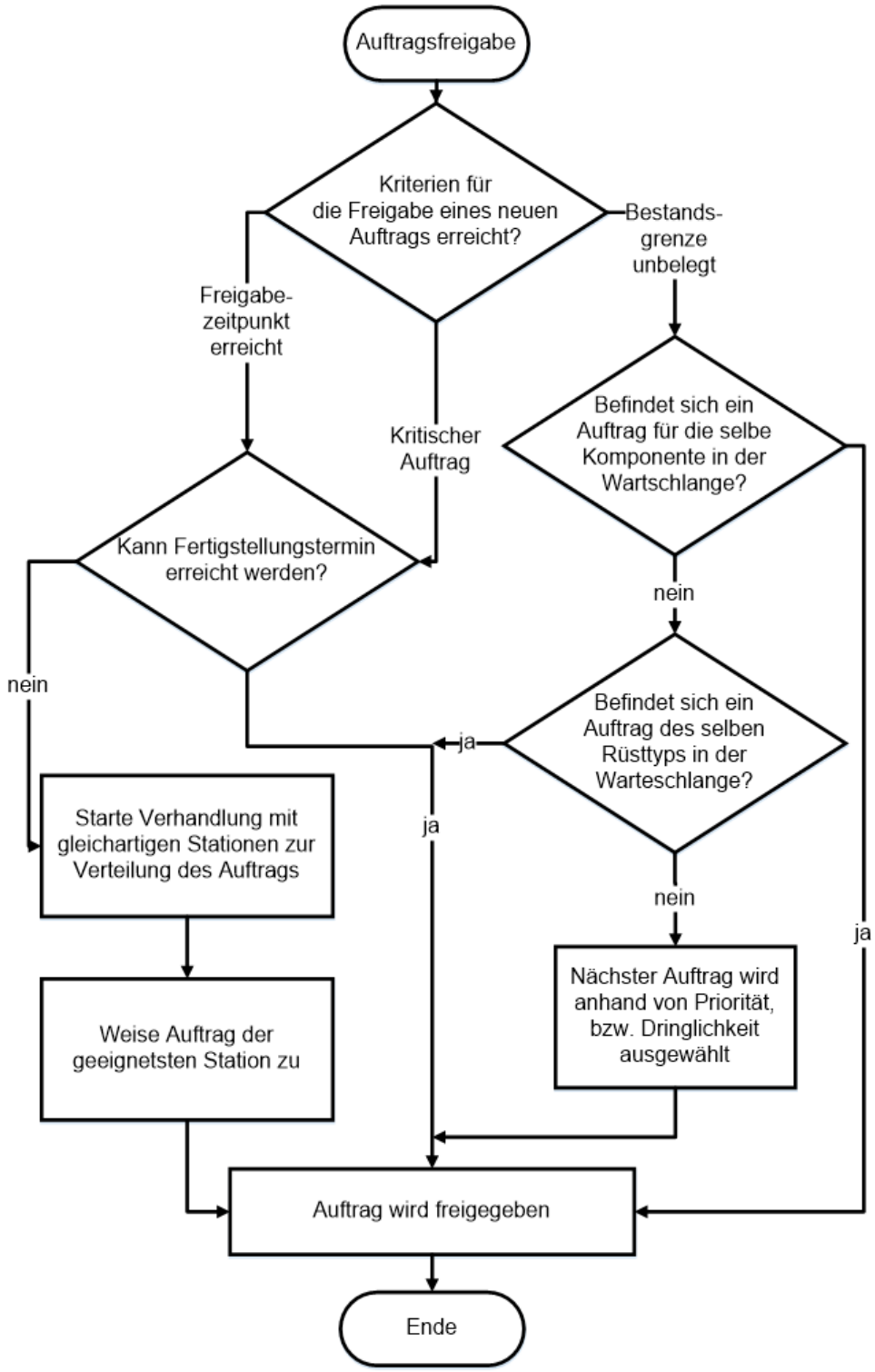

Abbildung 22: Ablaufdiagramm der Auftragsfreigabe

Bei der Auftragsfreigabe wird zuerst geprüft, ob die Kriterien zur Freigabe eines Auftrags erfüllt werden. Hierbei gibt es interne Kriterien wie z.B. ob genug Ausgangsmaterialien für die Produktion des Auftrags vorhanden sind, und externe, auftragsbezogene Kriterien. Die auftragsbezogenen Kriterien sind, ob der Freigabezeitpunkt erreicht ist oder ob der

Auftrag ein kritischer Auftrag ist. Kritische Aufträge bezeichnen solche Aufträge, bei denen Anforderungen nicht mehr erfüllt werden können, wie z.B. durch Überschreiten des vorgegebenen spätesten Freigabetermins. Wenn der Freigabetermin erreicht ist oder wenn der Auftrag bereits ein kritischer Auftrag geworden ist, werden die Aufträge sofort freigegeben. Dann findet eine Verhandlung mit gleichartigen Arbeitsstationen zur Verteilung des freigegebenen Auftrags statt, um eine optimale Auftragsverteilung unter den Arbeitsstationen zu erreichen. Nach der Verhandlung wird der Auftrag der am besten geeigneten Arbeitsstation zugewiesen. Nach diesem Absicherungsprozess wird die Reihenfolgebildung an der Arbeitsstation gestartet.

- In der Regel wird der **Freigabezeitpunkt** von der Produktionsplanung an die Produktionssteuerung übertragen. Wird keine Freigabezeitpunkt von der Produktionsplanung definiert, werden sie von der Arbeitsstation selbst anhand der Fertigstellungszeitpunkte berechnet. Die übergeordnete Zielstellung ist die Einhaltung des spätesten Fertigstellungszeitpunkts. Das heißt, dass der dringlichste Auftrag als Nächstes freigegeben werden muss. Die Dringlichkeit wird dabei über den Startzeitpunkt und die Durchlaufzeit gemessen und dann bezogen auf den Fertigstellungszeitpunkt berechnet.
- Für die **Freigabe eines kritischen Auftrags** auf die Warteschlange einer Arbeitsstation muss zuerst überprüft werden, ob Aufträge in der Warteschlange oder dem Auftragsspeicher kritisch werden. Ist ein Auftrag kritisch, muss dieser Auftrag eventuell vorgezogen und andere Aufträge an der Arbeitsstation verschoben werden. Dabei hat der Schritt „die Verhandlung mit gleichartigen Stationen zur Verteilung des freigegebenen Auftrags“ eine entscheidende Bedeutung, da versucht wird, kritische Aufträge bestmöglich unter gleichartigen Arbeitsstationen aufzuteilen. Gleichartige Arbeitsstation bezeichnen hierbei die Arbeitsstationen, die diesen Auftrag bearbeiten können. Bei der Verhandlung mit anderen gleichartigen Arbeitsstationen werden die folgenden Überprüfungsfragen gestellt:
  - Kann der Auftrag auf der Arbeitsstation produziert werden?
  - Sind freie Kapazitäten bei dieser Arbeitsstation vorhanden?
  - Können Randbedingungen des Auftrags (frühester Startzeitpunkt, frühester Endzeitpunkt) eingehalten werden?
  - Was ist die kürzeste Bearbeitungszeit (Berechnung unter Zuhilfenahme der Rüstreihenfolge)?

  Sind mehrere geeignete Arbeitsstationen vorhanden, wird die Arbeitsstation mit der kürzesten Bearbeitungszeit ausgewählt. Dadurch kann die Effizienz des Produktionsprozesses optimiert werden.
- Gibt es **mehrere kritische Aufträge**, findet eine Verhandlung mit anderen, gleichartigen Arbeitsstationen statt. Diese werden angefragt, die kritischen Aufträge

zu übernehmen. Die anderen Arbeitsstationen prüfen daraufhin, ob sie den Auftrag übernehmen können und dadurch der Fertigstellungszeitpunkt eingehalten werden kann. Hierbei existieren verschiedene Verhandlungsdurchläufe. Kann keine andere Arbeitsstation den Auftrag übernehmen, wird mit den nachfolgenden Arbeitsstationen im Produktionsprozess verhandelt. In diesem Rahmen wird geprüft, ob der Auftrag weiter nach hinten geschoben werden kann. Anschließend werden Maßnahmen zur Kapazitätserhöhung überprüft. Kann hierdurch auch keine Auftragsreihenfolge mit Einhaltung des Fertigstellungstermins erzeugt werden, findet an der Station intern eine Umplanung hinsichtlich der Prioritäten statt. Der Auftrag mit der niedrigsten Priorität wird anschließend zurückgestellt, um die Erreichung des Fertigstellungszeitpunkts zu bewerkstelligen. Dieser Prozess wird so lange durchgeführt, bis eine ausführbare Auftragsreihenfolge an der Station erreicht wird. Planer und Werker erhalten eine Meldung, sobald die Anforderungen an einen Auftrag nicht mehr erfüllt werden können. Diese entscheiden daraufhin eigenständig, ob sie weitere Maßnahmen zur Erfüllung des Auftrags umsetzen oder ob diese für den Auftrag nicht eingehalten werden. Dieses Verfahren zielt somit primär auf die Einhaltung von Terminen.

- Die **Bestandsgrenze** wird mit Hilfe eines Zielbestands berechnet. Der Zielbestand wird anhand des gewünschten Umlaufsbestands berechnet und vorgegeben. Gleichzeitig muss der Vorlauf von vorhergehenden Arbeitsstationen des Auftrags berücksichtigt werden. Dadurch wird vermieden, dass sich ein Produktionsstau an dieser Arbeitsstation bildet.
- Die Priorität eines Auftrags ist statisch und setzt sich aus Faktoren zusammen, die die Wichtigkeit des Auftrags beschreiben. Die Festlegung der Priorität ist dabei unternehmensspezifisch, wobei Einflussfaktoren wie Auftragsvolumen oder Kundenbewertung ausschlaggebend sind. Wenn keine Priorität vorhanden ist, werden die Aufträge ausschließlich nach der Dringlichkeit freigegeben.

Das entwickelte Verfahren zur Auftragsverteilung

Es wird im Rahmen der Auftragsfreigabe davon ausgegangen, dass alle Aufträge bereits den Arbeitsstationen zugeordnet sind. Ein zugeordneter Auftrag wird anschließend freigegeben. Allerdings gibt es ebenso Aufträge, die noch nicht von der Produktionsplanung zu einer Arbeitsstation zugeordnet sind. Die Steuerung solcher Aufträge wird heutzutage manuell durchgeführt. Genau hier setzt die in dieser Arbeit entwickelte dezentrale Produktionssteuerung an, um die Auftragsverteilung zu automatisieren. Neue Produktionsaufträge, die noch keiner Arbeitsstation zugeordnet sind, werden durch die Verhandlung zwischen den Arbeitsstationen zugeordnet. Dann werden die Aufträge auf die Arbeitsstationen aufgeteilt und nach vordefinierten Regeln freigegeben. Zudem wird das Verfahren zur Auftragsverteilung für die Optimierung der von der Produktionsplanung vorgegebenen Auftragsverteilung eingesetzt, um die Liefertreue einzuhalten und die Produktionskapazität auszuschöpfen. Dieses Verfahren

besteht aus zwei Grundbausteinen: das **Verfahren zur Verhandlung** zwischen den Arbeitsstationen und die vordefinierten **Regeln zur Freigabe** der vorher nicht zugewiesenen Aufträge.

Auch bei den nicht zugewiesenen Aufträgen gilt, dass zuerst die Aufträge freigegeben werden, deren Freigabezeitpunkt erreicht ist. Sollten keine Freigabezeitpunkte definiert sein, werden die **Regeln zur Freigabe** der Aufträge als Referenz genommen. Zuerst werden kritische Aufträge freigegeben. Existieren keine kritischen Aufträge, wird bei der Unterschreitung des Auftragsvorrats an einer Arbeitsstation der dringlichste Auftrag freigegeben. Die Aufträge werden dabei von den Arbeitsstationen selbst freigegeben. Ziel ist es, möglichst für jeden Auftrag die Fertigstellungszeitpunkte einzuhalten und voraussichtlich entstehende Rückstände zu minimieren.

Im Folgenden wird das Verfahren zur **Verhandlung zwischen den Arbeitsstationen** vorgestellt. Die Verhandlung zwischen den Arbeitsstationen folgt dem in Abbildung 23 dargestellten Szenario. Das Szenario beschreibt den Verhandlungsprozess zur Verteilung eines kritischen Auftrags mit dem entsprechenden Ablaufdiagramm und dem Prozessablauf. Die Zwischenschritte der Absprache sind dabei nicht verdeutlicht, auf diese wird vertiefend in Abschnitt 6.2 eingegangen. Die Arbeitsstationen verhandeln darüber, welche Arbeitsstation für die Erfüllung des Auftrags am besten geeignet ist. Das wesentliche Kriterium stellt die Erfüllung des Fertigstellungszeitpunktes dar. Zudem werden die Bearbeitungszeiten, Rüstzeiten und gleichartige Aufträge bei der Verteilung der Aufträge berücksichtigt.

In Abbildung 23 wird die Verteilung eines kritischen Auftrags dargestellt, für den auf der Arbeitsstation keine Kapazitäten mehr vorhanden sind. Deswegen werden im ersten Schritt Arbeitsstationen angefragt, die diesen Auftrag übernehmen könnten. Falls es hiervon mehrere gibt, findet eine Auswahl statt. Falls es nur eine geeignete gibt, wird diese direkt ausgewählt. Sollte keine Station geeignet sein, wird geprüft, ob der Auftrag an der Folgearbeitsstation weiter nach hinten geschoben werden kann und dabei trotzdem der Fertigstellungstermin des Kundenauftrags erfüllt werden kann. Sollte keine andere Arbeitsstation geeignet sein und auch der Produktionsauftrag nicht nach hinten geschoben werden können, wird direkt eine Meldung an das Planungssystem erzeugt, damit notfalls eine manuelle Umplanung durchgeführt wird. Gleichzeitig werden jedoch weitere Maßnahmen zur Erfüllung des Auftrags geprüft. Im ersten Schritt wird hierbei eruiert, ob eine Kapazitätserhöhung an einer Station zur Erfüllung des Auftrags vorgenommen werden kann. Sollte auch dies nicht möglich sein, werden alle konfliktären Aufträge ermittelt. Der Auftrag mit der niedrigsten Priorität wird anschließend verworfen, um die Liefertreue höher priorisierter Aufträge zu gewährleisten.

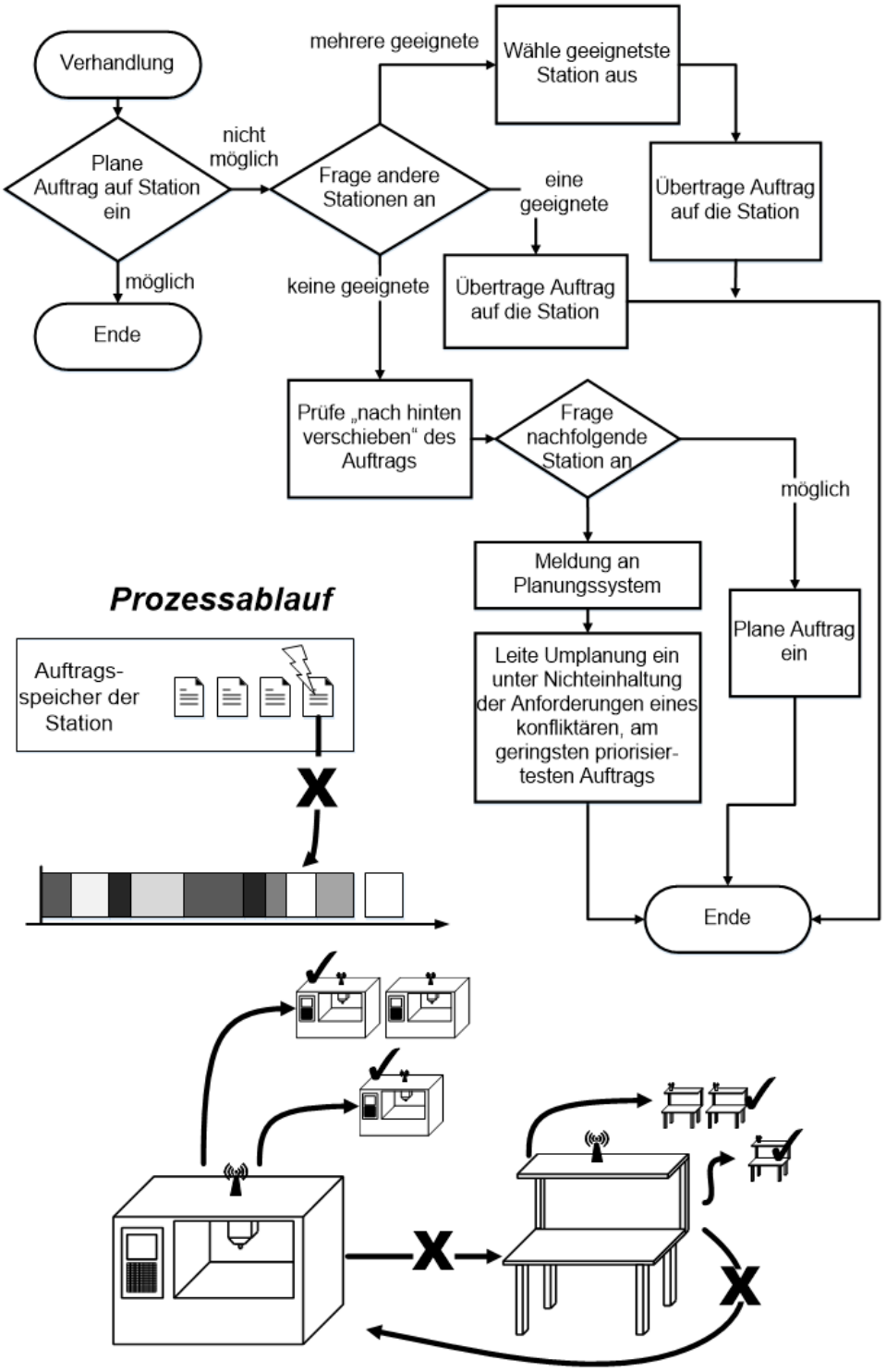

Abbildung 23: Verhandlung zur Verteilung eines Auftrags

Reihenfolgebildung an einer Arbeitsstation

Nach der Zuordnung eines Auftrags zu einer Arbeitsstation wird die Reihenfolgebildung an der Arbeitsstation angestoßen. Dieser Prozess zur Reihenfolgebildung ist in Abbildung 24 dargestellt. Die Reihenfolgebildung wird bei der Freigabe eines neuen Auftrags auf

der Arbeitsstation oder kurz vor der Fertigstellung eines Auftrags auf der Arbeitsstation durchgeführt. Bei der Reihenfolgebildung wird analog zur Auftragsfreigabe anhand der Kriterien „Dringlichkeit“ und „Priorität“ die Reihenfolge der Aufträge bestimmt. Zudem findet eine Optimierung hinsichtlich Losgrößen und Rüstzeiten statt, wobei geprüft wird, ob sich ein Auftrag desselben Rüsttyps bzw. derselben Komponente in der Warteschlange befindet. Ist dies der Fall, wird versucht, diese Aufträge nacheinander zu platzieren. Das Hauptziel der Steuerung bildet somit die Einhaltung der Liefertreue.

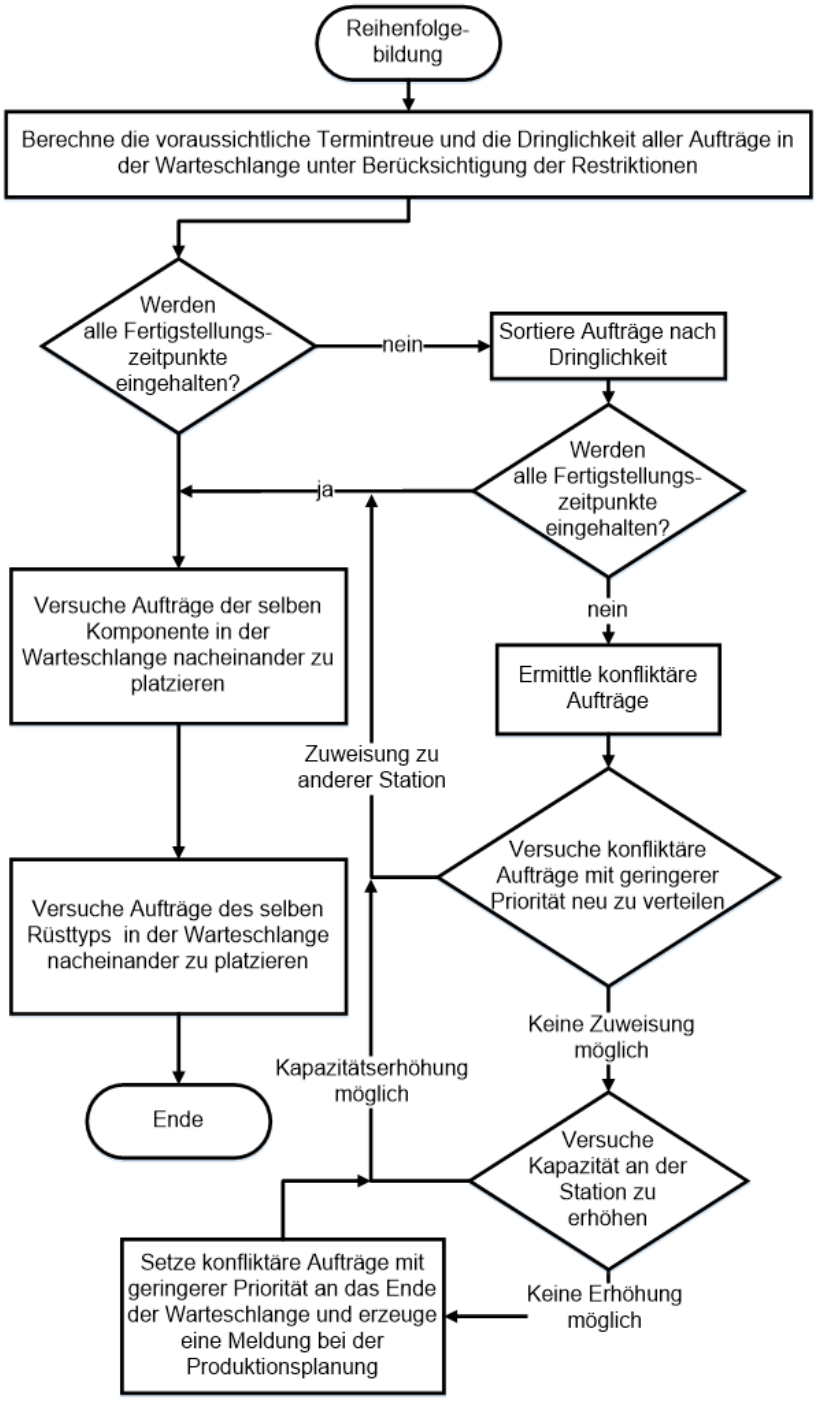

Abbildung 24: Ablaufdiagramm der Reihenfolgebildung

Im Rahmen der Reihenfolgebildung werden bei keiner Erreichung eines ausführbaren Plans wiederum die anderen Arbeitsstationen angefragt, ob sie diesen Auftrag übernehmen können und ob eine Verschiebung des konfliktären Auftrags nach hinten möglich ist. Daraufhin findet eine Verhandlung zwischen den Arbeitsstationen statt. Erst wenn dabei keine Lösung gefunden wird, wird das Kriterium der Priorität eingesetzt, um die Aufträge mit der geringeren Priorität nach hinten zu sortieren. Mit Abschluss der Reihenfolgebildung ist der Prozess der Einlastung neuer Aufträge beendet. Erst wenn neue Aufträge eingespielt werden oder ein neuer Produktionsplan an die Steuerung übergeben wird, wird der Einlastungsprozess neu gestartet.

4.2.2 Das entwickelte Verfahren zur Umplanung

Das in dieser Arbeit entwickelte Verfahren zur Umplanung stellt das Alleinstellungsmerkmal bzw. den Neuheitsgrad dar. Umplanungen werden jedoch nicht von allen Veränderungen verursacht. Es werden Schwellwerte für den Eingriff der Steuerung definiert, in denen keine Veränderung vorgenommen wird. Für jede Überschreitung bzw. Unterschreitung innerhalb dieses Schwellwerts wird keine Umplanung eingeleitet. Diese Schwellwerte sind für alle Prozesse auf allen eingesetzten Arbeitsstationen mit einem Zeitwert zu parametrisieren. Umplanungen werden in der Regel von der Kapazitätssteuerung angestoßen. Die Kapazitätssteuerung findet dabei sowohl periodisch als auch ereignisorientiert statt. Periodisch erfolgen hierbei jeweils ein Abgleich zwischen den Start- und Endterminen, sowie die Prüfung der Einhaltung der Kapazitätsgrenzen. Gleichzeitig wird geprüft, ob eine Kapazitätserhöhung oder Kapazitätsverringerung notwendig bzw. machbar ist. Ziel ist es, die Fertigstellungstermine mit einem minimalen Kapazitätsaufwand einzuhalten bzw. genügend Kapazität anzubieten, damit die Fertigstellungstermine eingehalten werden können. Das entsprechende Vorgehen hierfür ist in Abbildung 25 des Ablaufdiagramms der Kapazitätssteuerung dargestellt.

Die Kapazitätsteuerung zielt vorrangig auf die Einhaltung der Fertigstellungstermine. Wenn die Fertigstellungstermine eingehalten werden können, versucht die Kapazitätssteuerung die eingeplante Produktionskapazität abzusenken bzw. zu optimieren. Dabei wird die Möglichkeit zur Kapazitätssenkung um eine vordefinierte Kapazitätsstufe geprüft. Gleichzeitig wird die Einhaltung der Fertigstellungstermine kontrolliert. Wenn nicht alle Fertigstellungstermine erreicht werden können, wird versucht, die Kapazität wieder zu erhöhen, bzw. auf dem alten Wert zu belassen. Wenn alle Fertigstellungstermine eingehalten werden können, wird geprüft, ob die automatisierte Umsetzung der Kapazitätssenkung durchgeführt werden kann. Anschließend wird die Prüfung der Kapazitätssenkung noch einmal durchgeführt, bis die Kapazität nicht weiter gesenkt werden kann. Sollten bereits bei der Ausgangsprüfung die Fertigstellungstermine nicht eingehalten werden können, wird versucht, eine Kapazitätserhöhung um eine vordefinierte Stufe umzusetzen. Anschließend wird geprüft, ob dadurch die Fertigstellungstermine eingehalten werden können. Wenn die Kapazität

nicht automatisiert erhöht werden kann, werden Meldungen an das Plansystem erzeugt, um die Kapazitätserhöhung manuell durchführen zu lassen oder eine Umplanung des gesamten Produktionssystems zu veranlassen.

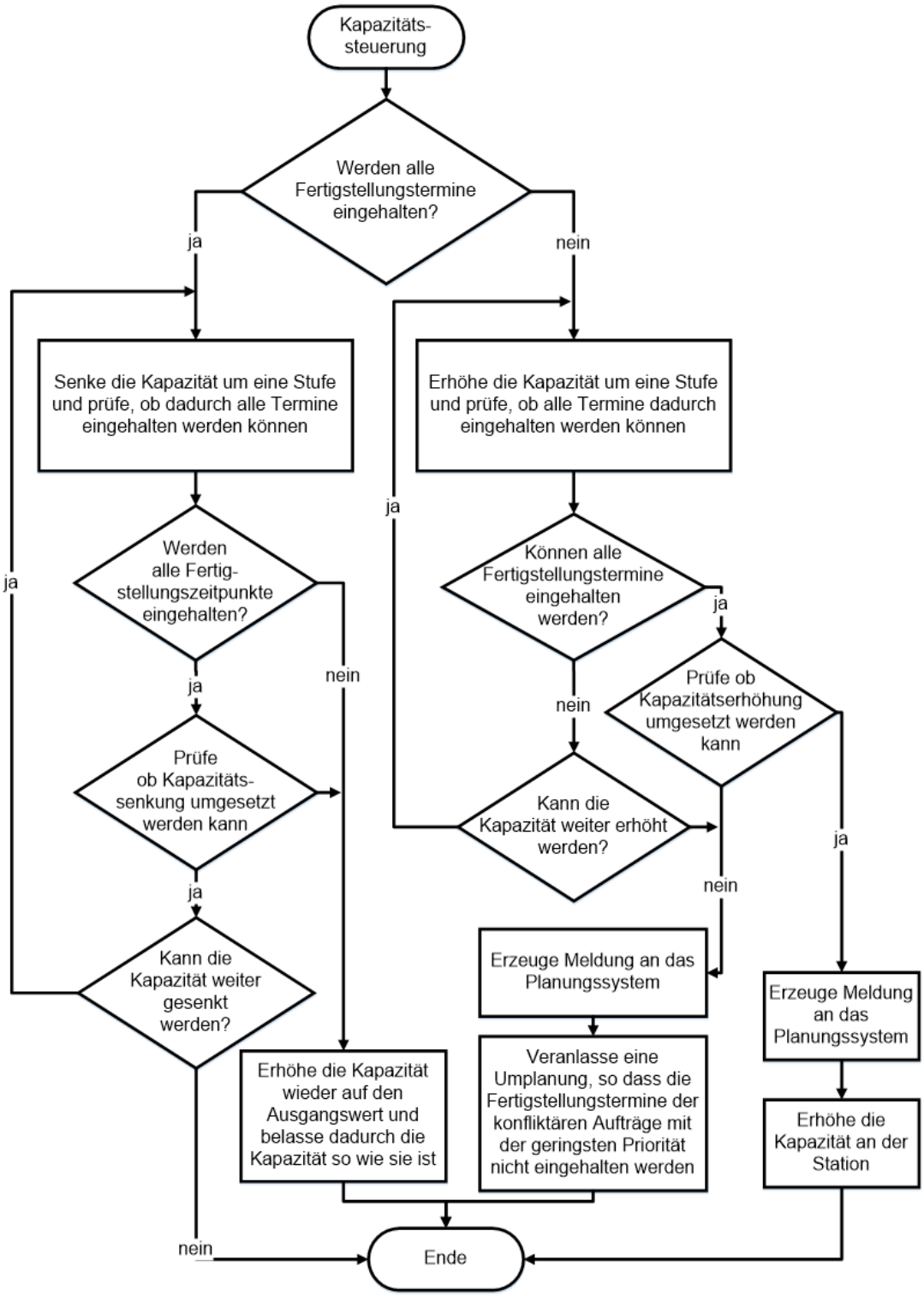

Abbildung 25: Ablaufdiagramm der Kapazitätssteuerung

Da Maßnahmen zur Kapazitätserhöhung oder -senkung mit Aufwänden verbunden sind, werden diese nur in Ausnahmefällen eingeleitet. Das vorrangige Ziel besteht in der Erfüllung der Fertigstellungszeitpunkte und dem ständigen Abgleich zwischen den Start- und Endterminen. Ist dieses Ziel gefährdet, wird im ersten Schritt eine neue Reihenfolgebildung eingeleitet, in deren Rahmen auch die Kommunikation mit den anderen Stationen erfolgt.

Erst wenn das Ziel zur Erfüllung der Fertigstellungszeitpunkte fehlschlägt, werden Maßnahmen zur Kapazitätserhöhung eingeleitet, die jedoch unternehmensbezogen unterschiedliche Ausprägungen haben. Häufig müssen vor Einsatzbeginn etwaige Überstunden geklärt sein. Der Betrachtungshorizont müsste dafür über die Schicht hinausgehen. Die bedeutendste Maßnahme umfasst das Veranlassen von Überstunden bzw. den Abbau von Überstunden. Das Modell, das mit Hilfe der digitalen Repräsentation hier angewendet wird, beinhaltet ein Abfragen der Beschäftigten, ob diese einer entsprechenden Erhöhung bzw. Senkung zustimmen. Es werden die Zeiten ermittelt, die benötigt werden, und diese werden inkrementell um eine Überperiode (in der Regel eine Stunde) erhöht. Alternativ können auch vorgehaltene Kapazitäten für eine Kapazitätserhöhung verwendet werden. Setzt ein Unternehmen zum Beispiel Springer ein oder besitzt noch ungenutzte Arbeitsstationen, können diese ebenso zur Kapazitätssteuerung eingesetzt werden. Für die Kapazitätssteuerung müssen Kapazitätsgrenzen, also ab welcher Unter- bzw. Überdeckung eingeschritten wird und die Kapazitätsstufen festgelegt werden.

4.2.3 Kommunikation zwischen den Arbeitsstationen innerhalb der CPPS

Die andere Kernfunktion der in dieser Arbeit entwickelten automatisierten dezentralen Produktionssteuerung ist die Kommunikation zwischen den Arbeitsstationen innerhalb eines dezentralen Produktionssystems. Dadurch werden der Informationsaustausch bzw. die Verhandlung zwischen den Arbeitsstationen ermöglicht. Die in den Abschnitten 4.2.1 und 4.2.2 vorgestellte Einlastung und Umplanung setzt diese Fähigkeit zur Kommunikation voraus.

In Abbildung 26 wird die Kommunikation zwischen den Arbeitsstationen veranschaulicht. Dabei wird zusätzlich der Informationsaustausch zu übergeordneten Diensten dargestellt. Der Informationsaustausch mit anderen Arbeitsstationen bezieht sich auf den Status der einzelnen Arbeitsstationen sowie ihre Änderungen. Bei gleichartigen Arbeitsstationen dient diese Kommunikation zudem zur Verhandlung über die Übernahme von Aufträgen. Die Kommunikationsfähigkeit gilt gleichzeitig als die Voraussetzung zur Umplanung an den Arbeitsstationen. Über die hier dargestellten Arbeitsstationen hinaus gibt es weitere Arbeitsstationen, die im Rahmen der dezentralen Steuerung der Produktion angefragt werden müssen, wie zum Beispiel Logistikeinheiten zum innerbetrieblichen Transport.

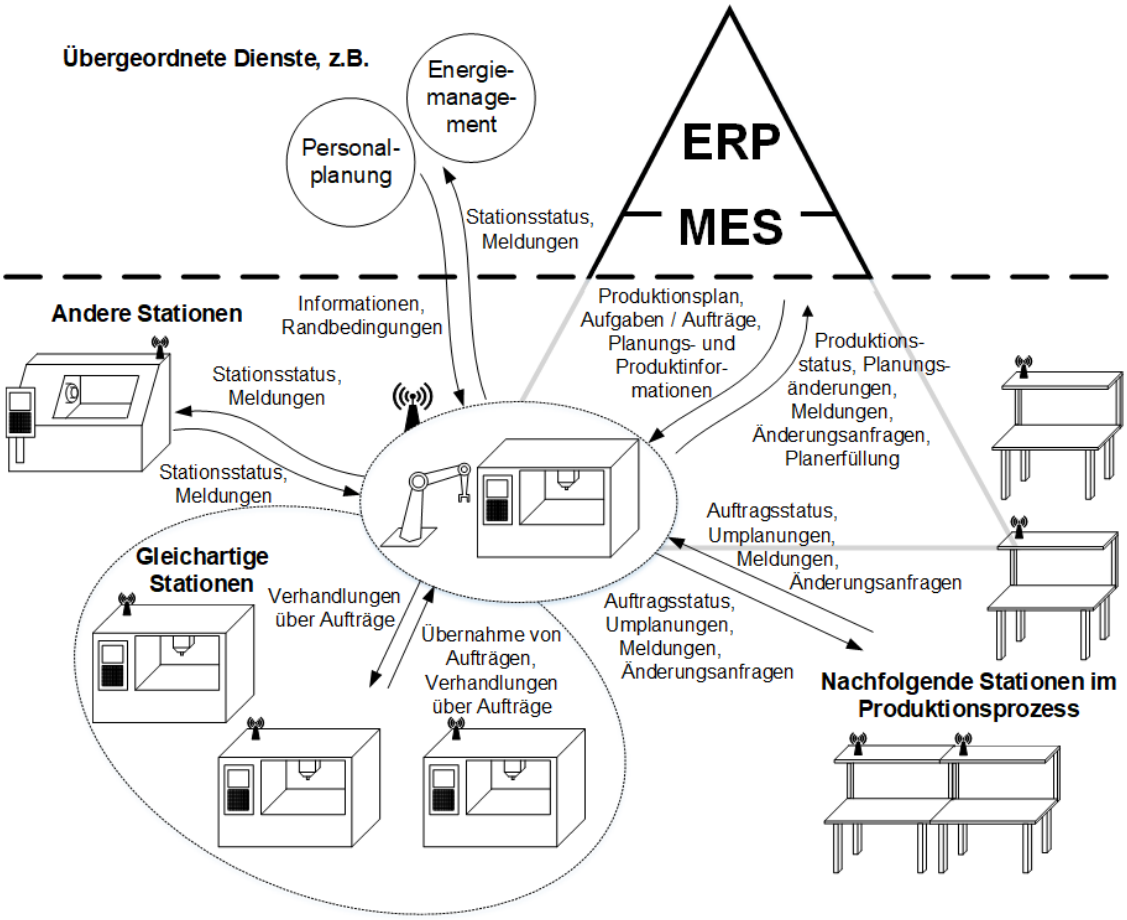

Abbildung 26: Kommunikation zwischen den Arbeitsstationen innerhalb des CPPS

Die Umsetzung des Kommunikationssystems wird in Abschnitt 6.2 näher erläutert. Nachfolgend werden vornehmlich die Regeln zur Kommunikation beschrieben. Es ergeben sich zahlreiche Fälle, in denen Arbeitsstationen untereinander verhandeln müssen, ob Aufträge geschoben oder von anderen Arbeitsstationen übernommen werden können. Die entsprechende Übersicht hierzu ist in Abbildung 26 enthalten. Dahingehend werden zwei Arten der Kommunikation unterschieden:

- Mitteilung: Wenn eine Veränderung keine unmittelbaren Auswirkungen auf andere Arbeitsstationen hat, aber z. B. den Status eines Auftrags verändert, geschieht dies über eine einfache Mitteilung an die betroffenen Arbeitsstationen. Wird zum Beispiel ein Auftrag um zehn Minuten nach hinten verschoben, muss den nachfolgenden Arbeitsstationen mitgeteilt werden, dass der früheste Startzeitpunkt ebenso um zehn Minuten nach hinten verschoben wurde. Hat dies keine unmittelbaren Auswirkungen auf die nachfolgenden Arbeitsstationen, reicht eine einfache Mitteilung aus und es muss keine weitere Absprache mit den betroffenen Arbeitsstationen erfolgen.
- Verhandlung / Absprache: Wenn andere Arbeitsstationen von möglichen Änderungen betroffen sind, müssen Absprachen zwischen den Arbeitsstationen getroffen werden.

Dies können zum Beispiel Verhandlungen darüber sein, welche Arbeitsstation einen Auftrag übernimmt, oder Anfragen über das Verschieben eines Auftrags.

Für Mitteilungen werden einfache JSON-Nachrichten mit den entsprechenden Änderungen versendet. Die Nachrichten sind dabei klassifiziert und die Beschreibung des Auftrags ist fest definiert. Bei der Absprache sind weitere Mechanismen erforderlich. Sobald ein Abspracheprozess von einer Arbeitsstation gestartet wurde und andere Arbeitsstationen angefragt wurden, werden die betroffenen Parameter auf der Arbeitsstation „eingefroren“. In diesem Status können dann keine Änderungen an den Parametern oder bezogen auf die Parameter vorgenommen werden. Falls zum Beispiel die Übernahme eines Auftrags angefragt wird, wird die gesamte Auftragssituation an der Arbeitsstation nicht weiterbearbeitet, solange nicht alle Rückmeldungen von anderen Arbeitsstationen hierzu eingetroffen sind. Kann zum Beispiel eine andere Arbeitsstation den Auftrag zu diesem Zeitpunkt übernehmen, verändert diese vor der Rückmeldung noch nicht ihre eigene Auftragsreihenfolge. Stattdessen friert sie entsprechend auch die eigene Auftragsreihenfolge ein, leitet also zu diesem Zeitpunkt keine Umplanung ein. Die angefragte Arbeitsstation meldet dies inklusive aller Parameter wie Bearbeitungszeiten, Rüstzeiten etc. zurück. Die Arbeitsstation, die den Prozess gestartet hat, wählt daraufhin die am besten geeignete Arbeitsstation für die Umplanung aus. Voraussetzung hierfür ist, dass alle Anforderungen an den Auftrag erfüllt werden. Die Arbeitsstation wird dabei auf Grundlage der folgenden Faktoren ausgewählt.

1. Erhöhung der Auslastung/Verhinderung von Pausen: Sollte im angefragten Zeitraum eine Arbeitsstation bisher keine Aufträge besitzen, wird diese Arbeitsstation vorrangig ausgewählt, um Wartezeiten möglichst zu vermeiden.
2. Gleiche Komponente: Sollte der Auftrag in einem Zeitraum geschoben werden können, bei dem direkt vor oder nach dem Auftrag die gleiche Komponente zu produzieren ist, wird diese Arbeitsstation ausgewählt, um entsprechend die Losgröße zu erhöhen.
3. Bearbeitungszeit: Häufig ergibt sich an Arbeitsstationen eine geringere Bearbeitungszeit als an anderen, da diese zum Beispiel mit einer höheren Bearbeitungsgeschwindigkeit den Auftrag ausführen können. Vorrangig werden die Arbeitsstationen mit der geringsten Bearbeitungszeit ausgewählt.
4. Rüstzeit: Sollte der Auftrag in einen Zeitraum geschoben werden können, bei dem direkt vor oder nach dem Auftrag Komponenten des gleichen Rüsttyps oder mit einer geringeren Rüstzeit produziert werden, wird diese Arbeitsstation ausgewählt, um entsprechend die Rüstzeiten zu verringern.
5. Auftragsvorrat: Sollte keines der oben genannten Kriterien zutreffen, wird die Station mit dem geringsten Auftragsvorrat ausgewählt.
6. Zusätzlich können weitere manuelle Kriterien zur Bevorzugung gewisser Arbeitsstationen ausgewählt werden.

In der nachfolgenden Abbildung 27 wird der Abspracheprozess bzw. Verhandlungsprozess zwischen den Arbeitsstationen dargestellt.

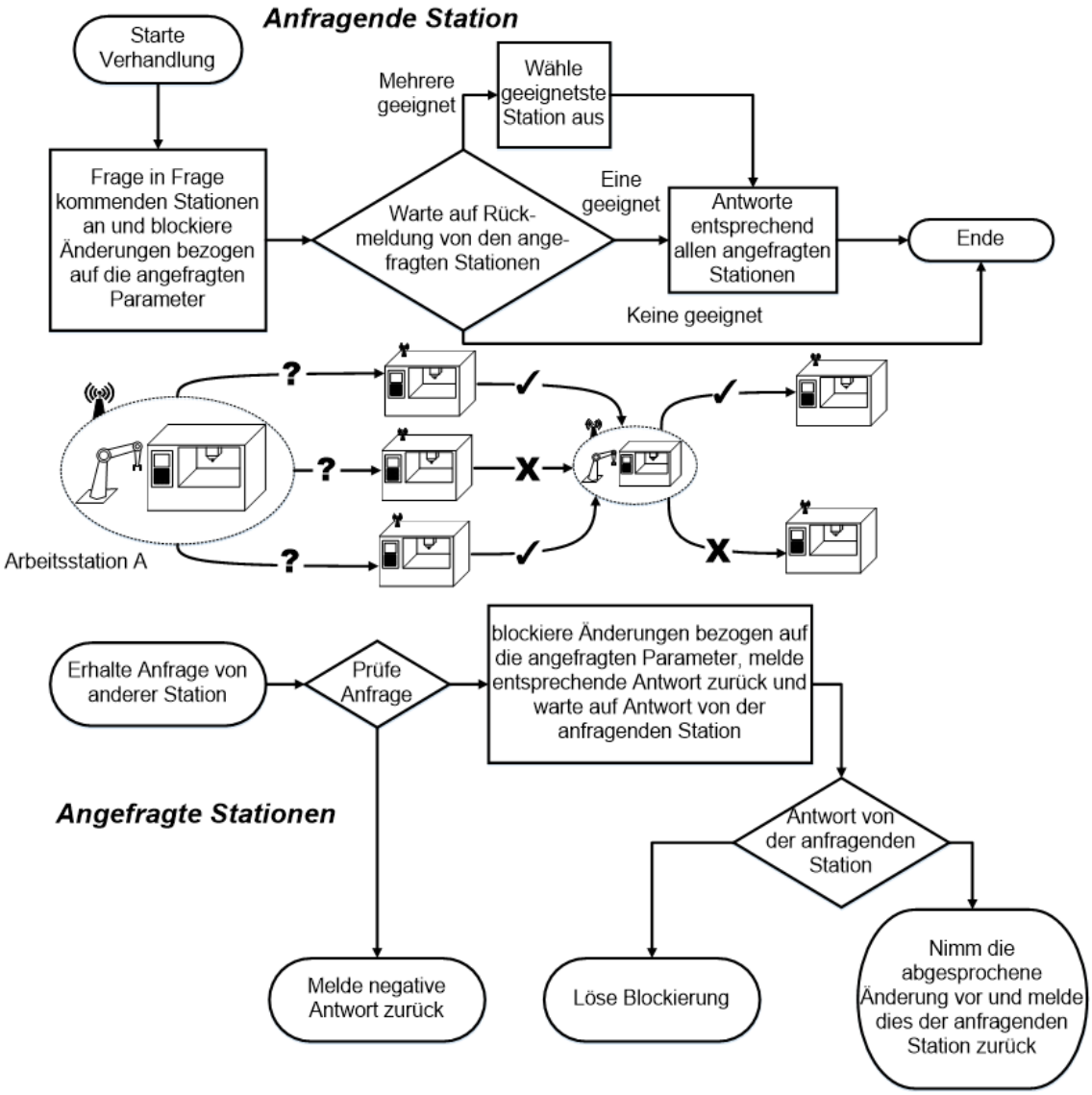

Abbildung 27: Verhandlungsablauf zwischen Arbeitsstationen

Der in der Abbildung dargestellte Verhandlungsprozess beginnt mit der Anfrage zur Übernahme eines Auftrags, die von der Arbeitsstation A an drei andere Arbeitsstationen verschickt wurde. Gleichzeitig werden die Änderungen bezogen auf die angefragten Parameter von der Arbeitsstation A blockiert. Anschließend wartet die Arbeitsstation A auf die Rückmeldungen von anderen Arbeitsstationen. Es existieren drei verschiedene Antwortmöglichkeiten. Eine Möglichkeit ist eine Antwort mit mehreren positiven Rückmeldungen. Die Arbeitsstation A wählt anschließend die am besten geeignete Arbeitsstation aus und sendet den anderen Arbeitsstationen eine negative Antwort zurück. Diese Möglichkeit wird in der mittleren Prozessdarstellung illustriert. Zwei Arbeitsstationen haben mit positiven Rückmeldungen geantwortet. Eine Arbeitsstation

hat die Anfrage abgelehnt. Die Arbeitsstation A hat die obere Arbeitsstation ausgewählt und schickte gleichzeitig der unteren Arbeitsstation eine negative Antwort. Bei der zweiten Möglichkeit gibt es nur eine positive Rückmeldung. Dann wird die entsprechende Arbeitsstation ausgewählt. Bei der dritten Antwortmöglichkeit gibt es keine positiven Rückmeldungen.

In diesem Kapitel wird das in dieser Arbeit entwickelte Verfahren zur automatisierten dezentralen Produktionssteuerung vorgestellt. Zuerst wurden die Anforderungen zur Entwicklung des Verfahrens abgeleitet. Darauf basierend wurde das Gesamtkonzept zur Veranschaulichung des Verfahrens vorgestellt. Zur Realisierung des Gesamtkonzepts müssen die drei wesentlichen Aufgaben behandelt werden: die Einlastung, die Umplanung und die Kommunikation zwischen den Arbeitsstationen. Für die Entwicklung des Verfahrens zur Einlastung wurden Verfahren zur Auftragsfreigabe und Auftragsverteilung entwickelt. Das Verfahren zur Umplanung wurde basierend auf einem Ablaufdiagramm vorgestellt. Für die Kommunikation zwischen Arbeitsstationen wurden Abspracheregeln definiert. Die Umsetzung der automatisierten dezentralen Produktionssteuerung wird in Kapitel 6 vorgestellt, die anschließende Validierung in Kapitel 7.

5 Digitale Repräsentation der Beschäftigten

In Kapitel 4 wurde die in dieser Arbeit entwickelte automatisierte dezentrale Produktionssteuerung vorgestellt. Zur Einbindung der Beschäftigten in diese automatisierte dezentrale Produktionssteuerung wird nachfolgend eine digitale Repräsentation für die Personaleinsatzplanung und -steuerung entwickelt.

Bei von Beschäftigten ausgeführten Prozessen schwanken die Bearbeitungszeiten und Arbeitsqualität erheblich. Dies erschwert die Planung und Steuerung der Produktion und führt zu ungenauen und teilweise falschen Planungsvorhersagen und damit zu Fehlplanungen. Um diese Fehlplanungen zu beseitigen, wird nachfolgend das Thema automatisierte Personaleinsatzplanung und -steuerung in der Produktion tiefergehend untersucht.

In Abschnitt 2.3 wurde der Handlungsbedarf abgeleitet. Durch die Einführung von CPPS werden nicht nur die Steuerungssysteme, sondern auch die Einbindung der Beschäftigten in die Produktion verändert. Diese Veränderungen wurden durch zahlreiche Studien untersucht (siehe Abschnitt 3.2). CPPS führen zu einer Automatisierung der Produktionsplanung und -steuerung und damit einer automatischen Zuweisung von Aufgaben an Beschäftigte. Daraus entsteht die Anforderung, dass Beschäftigte in einem automatisierten dezentralen Produktionssteuerungssystem über Steuerungsentscheidungen bestimmen können. Daher ist ein neues Verfahren für die Personaleinsatzplanung und zur Einbindung der Beschäftigten in Produktionssysteme erforderlich. Dafür wurde eine digitale Repräsentation der Beschäftigten als Lösungsansatz in dieser Arbeit entwickelt.

Die digitale Repräsentation der Beschäftigten ist in der Lage, den Status und die Fähigkeiten von Beschäftigten bezogen auf die Produktionssteuerung zu visualisieren. Das bietet eine umfangreiche Informationsbasis über die Beschäftigten für die automatisierte dezentrale Produktionssteuerung. Somit können die Planung bzw. Zuweisung von Beschäftigten zu Anlagen und Aufträgen durch die Integration der digitalen Repräsentation der Beschäftigten optimiert und automatisiert werden.

Die Entwicklung der digitalen Repräsentation der Beschäftigten basiert auf einer digitalen Abbildung der Beschäftigten. Die digitale Abbildung der Beschäftigten fasst alle für die Produktionsplanung und -steuerung relevanten, messbaren Daten von Beschäftigten zusammen. Die digitale Repräsentation greift auf diese Abbildung zu, um für die Prozessausführung relevante Entscheidungen und Parameter zu prognostizieren und so Steuerungsentscheidungen zu treffen.

5.1 Randbedingungen zur Entwicklung der digitalen Repräsentation der Beschäftigten

Die digitale Repräsentation der Beschäftigten überträgt Planungsentscheidungen der automatisierten dezentralen Produktionssteuerung auf manuell ausgeführte Prozesse

überträgt. In den nachfolgenden Abschnitten 5.2 bis 5.4 wird die technische Entwicklung der digitalen Repräsentation der Beschäftigten und ihre Implementierung in ein Produktionsumfeld detailliert vorgestellt. In diesem Abschnitt werden die Randbedingungen zur Entwicklung der digitalen Repräsentation der Beschäftigten abgeleitet.

Die Grundlage der digitalen Repräsentation bilden daher die Daten der Arbeitsprozesse, die durch die Mitarbeitenden ausgeführt werden. Die Bereitschaft der Beschäftigten diese Daten aufzunehmen, wird für die Entwicklung und den Einsatz der digitalen Repräsentation vorausgesetzt. Menschen sind jedoch keine technischen Systeme, auf welche einfach von außen Instrumentarien angewendet werden können. Die Einführung einer solchen digitalen Repräsentation der Beschäftigten benötigt daher die Erlaubnis, das Verständnis und eine aktive Beteiligung der Mitarbeitenden. Hierbei sind unter anderem die Vorstellungen der Beschäftigten und ihre Bedenken bei der Einführung der digitalen Repräsentation sehr wichtig. Methoden des Veränderungsmanagements eignen sich hierbei zur Unterstützung der Einführung. Beispiele von Veränderungsprozessen, die hierfür angewendet werden können, finden sich in [Grä15, Din17, Har11]. Zudem sind bei der Einführung der digitalen Repräsentation der Beschäftigten zahlreiche Randbedingungen im Rahmen der Personaleinsatzplanung zu beachten. Nachfolgend werden diese Randbedingungen aufgezeigt.

Die Personaleinsatzplanung benötigt eine Abstimmung zwischen Arbeitgebern und Arbeitnehmern. Das generelle Arbeitgeberinteresse ist auf den bestmöglichen wirtschaftlichen Erfolg allen unternehmerischen Handelns ausgerichtet. Diesem Interesse wird auch die Personaleinsatzplanung untergeordnet [ScSt11]. Die Personaleinsatzplanung ergibt sich aus der übergeordneten Unternehmensplanung sowie Planungen anderer Funktionsbereiche im Unternehmen. Diese werden ihrerseits wiederum durch externe Rahmenbedingungen wie z. B. Gesetze, Markteinflüsse, Technologieentwicklung beeinflusst. Die Vertreter des Human Resources Management (HRM)-Ansatzes gehen davon aus, dass sich das strategische und operative HRM mit denjenigen Voraussetzungen des Unternehmenserfolgs beschäftigen muss, in denen das Mitarbeiterinnen- und Mitarbeiterpotenzial eine entscheidende Rolle spielt [Thi11].

Zudem haben Beschäftigte in Bezug auf ihr Beschäftigungsverhältnis eine Reihe sehr unterschiedlicher und auch einander widersprechender Interessen [LR14]:

- gute, humane Arbeitsbedingungen
- einen sicheren Arbeitsplatz
- angemessene Bezahlung
- eine anspruchsvolle Tätigkeit
- Vereinbarkeit von Familie und Beruf
- berufliche Entwicklungsmöglichkeiten

Eine gute Personaleinsatzplanung kann dazu beitragen, dass diese Interessen der Beschäftigten besser mit den wirtschaftlichen Anforderungen des Unternehmens bzw. Betriebes in Übereinstimmung gebracht werden können. Sie kann somit eine Versachlichung und Verstetigung der Personalpolitik fördern und helfen, um unvermeidliche, betriebliche Konflikte aufgrund der unterschiedlichen Interessenlagen im Wege der Kompromissfindung leichter und besser zu lösen [LR14].

Neben den Anforderungen von Beschäftigten, die bei der Personaleinsatzplanung berücksichtigt werden sollen, existieren darüber hinaus gesetzliche Anforderungen an die Personaleinsatzplanung. Dabei spielen unter anderem die Aufnahme und die Verarbeitung personenbezogener Daten eine besonders wichtige Rolle für die Entwicklung bzw. Einführung der digitalen Repräsentation. Im Folgenden werden die wesentlichen gesetzlichen Anforderungen beschrieben.

Eine Verarbeitung personenbezogener Daten ist grundsätzlich verboten und erst dann zulässig, wenn es einen Rechtfertigungsgrund dafür gibt. Solche Rechtfertigungsgründe können sich aus dem Bundesdatenschutzgesetz oder aus einer Einwilligung der Betroffenen ergeben. Die Begriffe der „Verarbeitung“ und des „Personenbezugs“ werden dabei sehr weit gefasst, so dass eine Vielzahl von Anwendungen in der Industrie 4.0 darunterfallen kann: Ein Verarbeiten liegt z.B. bereits vor, wenn personenbezogene Daten übermittelt oder gespeichert werden. Für den Personenbezug genügt es sogar schon, wenn zum Beispiel eine Person über ein Datum ermittelt werden kann. Daraus entsteht ein großes Hindernis zur Einführung der digitalen Repräsentation der Beschäftigten. Für die Aufnahme, Speicherung und Verarbeitung der Daten ist stets eine Einwilligung von den Mitarbeitenden erforderlich. Allerdings muss diese Einwilligung strenge Anforderungen erfüllen. Zum Beispiel müssen die Aufnahme und die Verwendung der Daten vorab geklärt sein und es dürfen nur für die Ausführung der Tätigkeit notwendige Daten erfasst werden. Eine generelle Verarbeitung von Daten zur Bestimmung der Mitarbeiterproduktivität, wie sie mit Hilfe der digitalen Repräsentation der Beschäftigten angestrebt wird, ist z.B. gesetzlich nicht zulässig [BH06].

Daher erfordert eine Einführung der digitalen Repräsentation der Beschäftigten in Unternehmen zahlreiche Anpassungen, um die gesetzlichen und mitarbeiterbezogenen Anforderungen zu erfüllen. Dafür müssen die Beschäftigten gemeinsam mit dem Betriebsrat in Lösungen vor Ort eingebunden werden. Ein Mechanismus, um die Grundanforderungen an die Verarbeitung personenbezogener Daten zu erfüllen, wurde bereits im Rahmen dieser Arbeit in die digitale Repräsentation der Beschäftigten implementiert. Alle personenbezogenen Daten werden von der digitalen Repräsentation nur lokal gespeichert und ein Teilen der Daten muss zwingend von den Beschäftigten freigegeben werden. Dadurch können die Beschäftigten selbst entscheiden, welche Informationen über Sie an die Produktionsanlagen weitergegeben werden dürfen. Möchte eine Mitarbeiterin / ein Mitarbeiter keine Daten teilen, werden von den Produktionsanlagen die Vorgabedaten verwendet.

5.2 Ausgangssituation zur Entwicklung der digitalen Repräsentation der Beschäftigten

Mit der in dieser Arbeit entwickelten dezentralen Produktionssteuerung, können CPPS selbstorganisiert gesteuert werden. Das heißt, dass die einzelnen Arbeitsstationen sich selbst gegenseitig koordinieren, welche Arbeitsstation wann welchen Auftrag ausführt. Dabei ist ein wesentliches Problem heutiger Produktionsplanungs- und -steuerungssysteme, dass Beschäftigte als Bediener einer Arbeitsstation (zum Beispiel bei Fertigungsanlagen) oder als ausführende Produktionsanlage (zum Beispiel in der manuellen Montage) nicht betrachtet werden. Die Beschäftigten werden derzeit planungstechnisch nur als ein Bestandteil einer Arbeitsstation angesehen. Die Prozessbearbeitungszeiten von Beschäftigten werden lediglich aus durchschnittlichen Werten abgeschätzt. Selbst die Zuweisung der Beschäftigten zu Arbeitsstationen werden als gegeben angenommen und in der Regel nicht mehr verändert. Die Bedürfnisse und Wünsche von Beschäftigten werden dabei kaum berücksichtigt.

Daraus ergibt sich **die grundlegende Anforderung** an die digitale Repräsentation der Beschäftigten, Steuerungsentscheidungen durch die Beschäftigten zu ermöglichen und eine individuelle Planung für die Mitarbeitenden zu realisieren. Dafür werden die einzelnen Arbeitsstationen und die digitalen Repräsentationen der Beschäftigten so betrachtet, dass sie sich als dezentrale Produktionsanlagen selbstständig koordinieren können. Dabei agiert die digitale Repräsentation für die Beschäftigten als Verbindungsglied zu dem dezentralen Produktionssteuerungssystem. Daraus ergeben sich die zwei Hauptaufgaben der digitalen Repräsentation. Zum einen muss die digitale Repräsentation das Verhalten der Beschäftigten nachbilden und die Entscheidungen im Verhandlungsprozess, also z.B. wer wann welchen Auftrag ausführt, für die Beschäftigten treffen. Zum anderen muss ein Verfahren für die Zuweisungsprozesse zwischen den Arbeitsstationen und der digitalen Repräsentationen der Beschäftigten entwickelt werden. Für die Zuweisung von Aufträgen zu Beschäftigten besteht das Grundprinzip darin, dass ein Abgleich zwischen den Fähigkeitsprofilen der Beschäftigten, der Arbeitsstationen und der Aufträge erfolgt. Zusätzlich soll hierbei Rücksicht auf Wünsche und Bedürfnisse der Beschäftigten genommen werden.

5.3 Entwicklung eines digitalen Abbilds der Beschäftigten

Das digitale Abbild der Beschäftigten bildet die Grundlage zur Entwicklung der digitalen Repräsentation der Beschäftigten. Das digitale Abbild der Beschäftigten fasst alle relevanten messbaren Daten über den Arbeitsprozess zusammen und vermittelt somit einen Überblick über den aktuellen Zustand. Digitale Abbilder werden eigentlich für technische Systeme erstellt und verwendet. Bei technischen Systemen stellen digitale Abbilder virtuelle Modelle von realen Systemen dar [SFL19]. Durch ein Modell, das bestimmte Funktionen des realen Systems nachahmt, können Vorhersagen abgeschätzt und Analysen durchgeführt werden. Insbesondere bei Themen wie Zustandsüberwachung

und vorausschauender Instandhaltung ist dieses Konzept von digitalen Abbildern weit verbreitet [WiGeP20]. Eine Verbindung zwischen dem digitalen Abbild und dem realen System kann über Sensoren und Kommunikationsschnittstellen erfolgen. Diese Idee des digitalen Abbilds wird in dieser Arbeit auf den Beschäftigten in der Produktion übertragen.

Dadurch können zwei grundlegende Faktoren der dezentralen Produktionssteuerung optimiert werden:

- Zeiten, die durch die Beschäftigten beeinflusst werden, insbesondere Tätigkeitszeit, aber auch Pausen- und Rüstzeiten. Diese werden für Prognosen eingesetzt und können dadurch die Planung und Steuerung optimieren.

  Das digitale Abbild von Beschäftigten hilft dabei, die Prognose von Prozesszeiten erheblich zu verbessern. Zuerst werden hierdurch Daten gesammelt und aggregiert. Die anschließende Analyse dient der Ermittlung individueller Bearbeitungszeiten.

- Abschätzungen und Planung der Einsatzmöglichkeiten der Beschäftigten ohne personenbezogene Daten zu übertragen. Hiermit sollen flexible Zuweisungen von Beschäftigten realisiert werden.

Das digitale Abbild von Beschäftigten ermöglicht es, eine automatisierte Zuweisung von Beschäftigten zu Arbeitsstationen vornehmen zu können. Zu diesem Zweck wird untersucht, wer welchen Prozess ausführen kann und welche Zeit hierzu wahrscheinlich benötigt wird.

Für die Ermittlung der Zeiten, wie z.B. die Bearbeitungszeit und die Pausenzeit, reicht ein einfaches Nachverfolgen der Tätigkeiten aus. Zudem muss für die Zuweisung zu Arbeitsstationen ein entsprechendes Profil der Beschäftigten erstellt werden. Im Folgenden wird auf die beiden Aspekte – Prozesszeiten und Zuweisung der Beschäftigten – detailliert eingegangen.

Abschätzung der Prozesszeiten

Zur Abschätzung der Prozesszeiten werden alle Einflussfaktoren auf die Ausführung des Prozesses durch den Menschen aufgenommen. Am relevantesten ist die Einschätzung der Leistung, also der wahrscheinlich zu erreichenden Prozesszeit durch die Beschäftigten. Die Leistung der Beschäftigten entsteht durch deren Leistungsangebot. Dieses Leistungsangebot wird von vielen Faktoren beeinflusst, die sich unterschiedlich kategorisieren lassen. Nachfolgend werden drei Kategorien vorgestellt: die Unterteilung des Leistungsangebots in Leistungsvoraussetzungen, die Unterteilung des Leistungsangebots in Merkmale und die Unterscheidung des Leistungsangebots in gruppenbezogene und individuelle Leistungsvoraussetzungen.

Das Leistungsangebot wird von verschiedenen Rahmenbedingungen beeinflusst. In der folgenden Abbildung 28 werden diese Voraussetzungen schematisch dargestellt. Dabei erfolgt eine zusätzliche Differenzierung in sachliche und menschliche

Leistungsvoraussetzungen. Die menschlichen Leistungsvoraussetzungen enthalten die Leistungsfähigkeit und die Leistungsbereitschaft. Unter der Leistungsfähigkeit sind erworbene (situativ) und angeborene (individuelle) Fähigkeiten zu verstehen. Die Leistungsbereitschaft bezieht sich auf physiologische und psychologische Aspekte. Sachliche Leistungsvoraussetzungen lassen sich in technische und organisatorische Rahmenbedingungen unterteilen. Die technischen Rahmenbedingungen bezeichnen die aufgaben- und arbeitsplatzbezogenen Aspekte, wie z.B. die Komplexität einer Aufgabe und die Ausstattungssituation der Arbeitsplätze. Die organisatorischen Rahmenbedingungen lassen sich in Arbeitsziel, Arbeitsvorbereitung und Arbeitsstrukturierung unterscheiden. Bei dieser Unterteilung werden sowohl die Aufgaben, die Rahmenbedingungen als auch die Beschäftigten berücksichtigt.

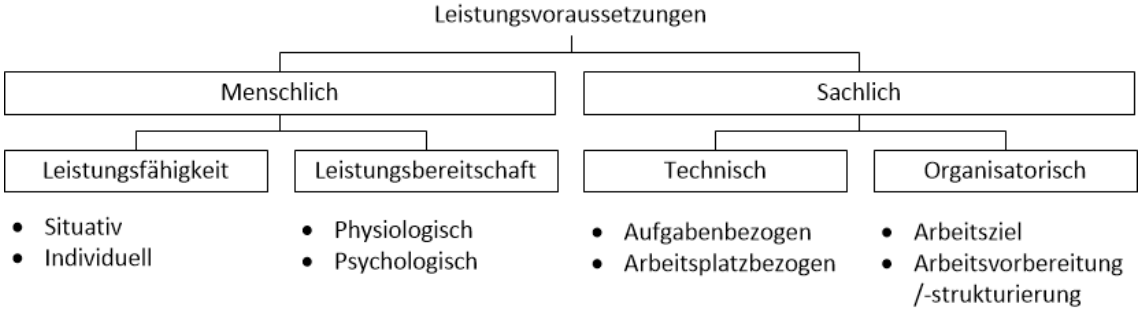

Abbildung 28: Unterteilung in menschliche und sachliche Leistungsvoraussetzungen [SLB09]

Abbildung 29 zeigt die Unterteilung des Leistungsangebots nach Merkmalen. Das Leistungsangebot wird hierbei nach Anpassungs-, Qualifikations- und Kompetenz-, Konstitutions- und Dispositionsmerkmale charakterisiert [GOS+13]. Die Anpassungsmerkmale beschreiben unter anderem die Beanspruchung, die Motivation und die Müdigkeit [FBH08].

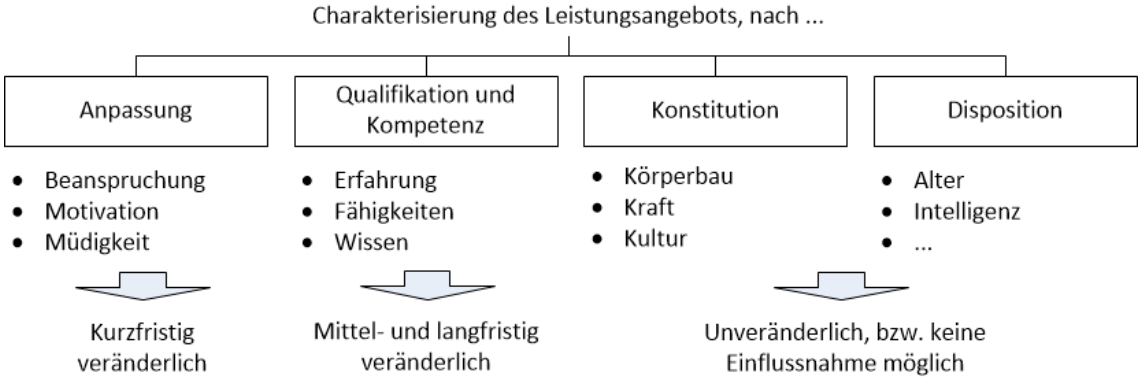

Abbildung 29: Unterteilung nach Charakterisierung des Leistungsangebots [SLB09]

Diese Merkmale können von außen kurzfristig beeinflusst werden. Im Vergleich dazu sind die Qualifikations- und Kompetenzmerkmale nur mittel- und langfristig veränderlich. Diese Merkmale beziehen sich auf Erfahrung, Fähigkeiten und Wissen von

Menschen. Die Konstitutions- und Dispositionsmerkmale können nicht verändert bzw. beeinflusst werden. Diese Unterteilung ist nur auf die Mitarbeiterin bzw. den Mitarbeiter bezogen und nicht auf unternehmensbezogene Rahmenbedingungen.

Die letzte Unterteilung des Leistungsangebots geschieht nach Leistungsschwankungen. Diese Leistungsschwankungen können Einzelpersonen- oder Personengruppenbezogen sein. Dies ist in der nachfolgenden Abbildung 30 verdeutlicht.

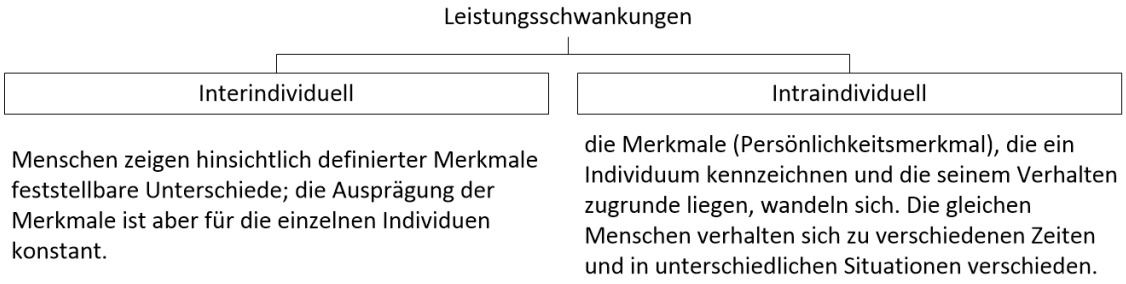

Abbildung 30: Unterteilung nach inter- und intraindividuellen Leistungsschwankungen [SLB09]

In Untersuchungen hat sich gezeigt, dass zwar Korrelationen zwischen gemessenen Daten und Arbeitsleistungen gefunden werden können, dass diese jedoch nicht ausreichend sind, um eine Abschätzung der Arbeitsleistung vornehmen zu können. Ein wesentliches Problem hierbei ist, dass die Auswirkung einzelner Faktoren auf die Prozessausführung nicht eindeutig ist. [RGE+11, FS99]

Es wird daher versucht, alle messbaren Indikatoren zusammenzutragen und diese direkt bei der Personaleinsatzplanung zu berücksichtigen. Hierbei handelt sich hauptsächlich um die Bearbeitungszeit und die Qualität des Arbeitsprozesses. Dies sind die einzigen Faktoren, die als messbare Indikatoren identifiziert werden konnten, die eine direkte Korrelation zu den Prozesszeiten und -ausführungen besitzen. Bei allen anderen Faktoren konnten keine direkten Korrelationen in ein Modell überführt werden. Ziel zukünftiger Arbeiten könnte somit die Ermittlung von Korrelationen zwischen Messgrößen und Leistungsmerkmalen von Beschäftigten sein. Im Rahmen der Produktionsplanung und -steuerung hat sich jedoch gezeigt, dass alle notwendigen Faktoren über die Leistung von Beschäftigten sich über die Prozesszeiten und das Arbeitsergebnis ermitteln lassen. Aus diesem Grund ist eine weitere Betrachtung individueller Aspekte im Rahmen dieser Arbeit nicht erforderlich.

Die Arbeitsleistung muss nicht nur abgeschätzt werden, sondern auch sachlich bewertet werden. In dieser Arbeit werden die sachlichen Leistungsbewertungen über die Prozesszeiten definiert. Somit können alle Einflussfaktoren des Leistungsangebots über die gemessenen Zeiten zurückgeführt, abgeschätzt sowie bewertet werden. Zum Beispiel können die menschlichen Leistungsvoraussetzungen über die Prozessausführungszeit eines Mitarbeiters erfasst werden. Die anderen Einflussfaktoren des Leistungsangebots wie z.B. Konstitutions-, Dispositions-, Qualifikations- und Kompetenzmerkmale werden

für die sachlichen Leistungsbewertungen nicht weiterverfolgt. Sie finden ausschließlich bei der Zuordnung von Arbeitsprozessen zur jeweiligen Person Anwendung. Daher wird die Zeit als einziger messbarer Faktor in dieser Arbeit für das digitale Abbild bzw. die digitale Repräsentation der Beschäftigten verwendet. Anhand der gemessenen Zeiten wird versucht, zukünftige Prozesszeiten zu prognostizieren.

Die Aufteilung der Bearbeitungszeit wird in dieser Arbeit nach der in Abschnitt 2.3 vorgestellten Aufschlüsselung in Rüst-, Tätigkeits-, Warte-, Erholungs- und Verteilzeit (siehe Abbildung 10) vorgenommen. Zudem wird geprüft, ob die Zeiten unmittelbar dem Auftrag und Arbeitsstationen zugeordnet werden können. Daraus ergeben sich die Verbindungen, die zur Prognose der Bearbeitungszeiten der Beschäftigten verwendet werden. Somit kann das Abbild über die Tätigkeit und deren Dauer für die Beschäftigen abgeleitet werden.

Daneben bilden die Anwesenheits- und Abwesenheitszeiten einen wesentlichen Aspekt für die Berücksichtigung der Bedürfnisse und Wünsche der Beschäftigten. In der Regel werden diese Zeiten in der Produktion eines Unternehmens strikt vorgegeben. Das widerspricht dem Ziel der Einbindung der digitalen Repräsentation der Beschäftigten in CPPS. Der Vorteil der Einführung für die Beschäftigten wird nachfolgend anhand von zwei Szenarien dargestellt.

Szenario 1:

„Annika ist in der Qualitätskontrolle eins Lebensmittelherstellers angestellt. Es ist 11 Uhr. Annika prüft gerade eine Charge für den wichtigsten Kunden des Unternehmens. Da erhält sie einen Anruf aus der Kita ihres Sohnes. Ihr Sohn habe Fieber und solle direkt abgeholt werden. Über ihr Handy teilt sie dem Produktionssteuerungssystem mit, dass sie schnellstmöglich die Arbeit für die nächsten 2 Stunden unterbrechen möchte. Unmittelbar erhält sie die Bestätigung vom System und fährt los zur Kita, damit sie ihren Sohn bei der Großmutter abgeben kann. Der Prüfauftrag wird unterbrochen, da die Verzögerung von zwei Stunden keine Gefahr für eine fristgerechte Lieferung bedeutet. In der Kita angekommen, erfährt Annika jedoch, dass ihr Sohn starkes Fieber hat und zum Arzt gebracht werden muss. Annika teilt per Smartphone dem Produktionssteuerungssystem mit, dass sie heute nicht mehr zur Arbeit erscheinen wird. Daraufhin reagiert das Produktionssteuerungssystem und fragt Walter an, einen Kollegen von Annika, den Auftrag abzuschließen. Walter willigt ein und entscheidet sich damit den höherpriorisierten Auftrag auszuführen und ermöglicht damit eine fristgerechte Versandabwicklung an den Kunden.“

Das Szenario zeigt, dass durch die digitale Repräsentation eine Möglichkeit geschaffen wird, Produktionspläne den individuellen Bedürfnissen von Beschäftigten anzupassen. Es können Vorgaben zur Planung definiert werden, welche in der Planung berücksichtigt werden. Gleichzeitig wird eine automatisierte Umplanung von Kapazitäten realisiert, welche eine kurzfristige Abwesenheit der Mitarbeiterin ermöglicht.

Szenario 2:

„Christian ist nach seiner Mechatroniker-Ausbildung als Schweißer in der Produktion von Sondermaschinen eines Mittelständlers eingestiegen. Nach über einem Jahr Anstellung möchte Christian sich nun weiterentwickeln und über Arbeiten in seinem Bereich hinaus die Arbeit in anderen Bereichen erlernen und teilt dies seiner Repräsentation im dezentralen Produktionssteuerungssystem mit. Diese schafft daraufhin zunächst Freiräume, in denen Christian sich in die neue Tätigkeit einlesen kann. Nach einer Woche wird Christian zur Unterstützung von Technikern anderer Bereiche hinzugezogen. Als ein Engpass in der Montage eintritt, bei der Christian bereits dreimal unterstützt hat, wird Christian angefragt und erledigt zum ersten Mal selbstständig diese Tätigkeit tadellos. Dies geschieht in den nächsten 2 Monaten häufiger. Als jedoch in dem Schaltschrankbau mehrere Mitarbeiter auf Grund langwieriger Krankheiten fehlen, wird Christian vom System angefragt, ob dieser sich auch in diese Tätigkeit einarbeiten kann. Nach einem weiteren Monat ist Christian nun in alle Tätigkeiten eingearbeitet. Da Christian es langweilt eine Tätigkeit den gesamten Tag auszuführen und dieser nun sehr vielseitig einsetzbar ist, achtet seine Repräsentation darauf spätestens nach der Mittagspause einen Wechsel der Tätigkeiten zu erreichen und hat bereits zwei neue herausfordernde Bereiche zur Einarbeitung vorgeschlagen.“

In dem zweiten Szenario wird dargestellt, dass die formelle Äußerung von Präferenzen von Tätigkeiten durch die digitale Repräsentation zu einer Aufwertung des Arbeitsplatzes führen kann. Es kann eine automatisierte Umsetzung von Job Rotation realisiert werden und gleichzeitig wird eine Möglichkeit zur systematischen Verfolgung von Maßnahmen zur Personalentwicklung geschaffen.

Entwicklung des Verfahrens für die Zuweisungsprozesse

Derzeit erfolgt die Zuweisung von Beschäftigten zu Arbeitsstationen so, dass diese vorab einer Kapazitätsgruppe zugeordnet sind. Über die Kapazitätsgruppe werden den Beschäftigten Eigenschaften zugewiesen. Eine Mitarbeiterin oder ein Mitarbeiter, der eigentlich für die Montage eingesetzt wird, muss zum Beispiel zunächst angelernt werden, bevor ein Einsatz in der Vorfertigung erfolgen kann. Eine Formalisierung dieser Fähigkeiten oder Eigenschaften wird heutzutage in der Regel nicht vorgenommen und auch nicht für die Planung berücksichtigt. Dies führt dazu, dass einzelne Beschäftigte nur langfristig, also wenn ein kompletter Tätigkeitswechsel vorgenommen wird, anderen Arbeitsstationen zugewiesen werden. Kurzfristige Tätigkeitswechsel werden nur in sehr seltenen Fällen und dann manuell vorgenommen. Die modulare dezentrale Produktionsstruktur zusammen mit der dezentralen Produktionssteuerung, ermöglicht einen komplett neuen Ansatz in der Zuweisung und Einplanung der Beschäftigten: das dezentrale Produktionssystem wird auf die einzelnen Produktionsanlagen heruntergebrochen und aufgelöst. Anhand des Informationsaustausches erfolgt die Zuweisung zwischen den Arbeitsstationen und Beschäftigten.

Für die Zuweisungsprozesse muss analysiert werden, welche Prozesse die Beschäftigten überhaupt ausführen können und in welcher Qualität und Zeit dies erfolgen kann. Als Grundlage für die Ermittlung der Prozesszeiten dienen, wie bereits beschrieben, die historischen Prozesszeiten. Außerdem werden die historischen Daten auch dafür verwendet, ob eine bestimmte Tätigkeit von Beschäftigten ausgeführt werden kann. Hat eine Mitarbeiterin oder ein Mitarbeiter einen Arbeitsprozess an einer Arbeitsstation bereits ausgeführt, so existieren bereits Prozesszeiten und Informationen über die Ausführung des Prozesses. Daraus ergibt sich die Aussage, dass diese Person diesen Prozess ausführen und diese Arbeitsstation bedienen kann. Dadurch wird die Zuweisung von Aufträgen und Beschäftigten zu Arbeitsstationen ermöglicht. Existieren solche Daten nicht, z. B. weil die Mitarbeiterin oder der Mitarbeiter noch nie an der Arbeitsstation gearbeitet hat oder ein Auftrag mit einem neuen Produkt angekommen ist, werden andere Prozeduren benötigt. In solchen Fällen wird auf das Fähigkeitsprofil der digitalen Repräsentation zurückgegriffen.

Im Fähigkeitsprofil werden alle für die Planung und Steuerung relevanten Fähigkeiten der Beschäftigten formalisiert. Damit kann ein vollständiger Abgleich zwischen dem Fähigkeitsprofil und den Anforderungen von Arbeitsstationen und Aufträgen durchgeführt werden. Für den Abgleich werden die Arbeitsprozesse in kleinste Arbeitsschritte unterteilt. Im Folgenden wird beispielhaft das Verfahren zum Aufbau des Fähigkeitsprofils vorgestellt. Tabelle 4 stellt exemplarisch die benötigten Fähigkeiten für eine Montageoperation dar.

Tabelle 4: Exemplarische Beschreibung einer Fügeoperation

Operation	Fernsteuerungsauto - 17 Steuerungskabel crimpen
Grundoperation	Fügen
Prozess	Crimpen – Anbringen einer Aderendhülse an eine Litze
Grundprozess	Herstellung einer elektrischen Verbindung zwischen der Aderendhülse und der Steuerungslitze
Beschreibung	Steuerungslitze abisolieren und anschließend durch Aderendhülse führen und mit der Crimpzange vercrimpen

In diesem Beispiel wird die Operation auf die Grundoperation „Fügen“ heruntergebrochen. Dann folgt eine genaue Beschreibung dieser Fähigkeit. Basierend auf den Informationen dieser Tabelle kann festgestellt werden, ob Beschäftigte die benötigten Fähigkeiten für die Ausführung des Auftrags vorweisen. Die Fähigkeiten der Beschäftigten werden durch die historischen Daten berechnet. Im ersten Schritt werden diese manuell definiert, damit Werte für Fähigkeiten existieren, falls dieser Prozess bisher noch nicht bearbeitet wurde. Anschließend findet ein Abgleichprozess zwischen den

Fähigkeiten von Beschäftigten und den benötigen Fähigkeiten statt. Dieser Abgleichprozess ist entscheidend für die Zuweisung der Beschäftigten zu Arbeitsstationen.

Der Abgleich basiert auf der Formalisierung der Fähigkeiten von Beschäftigten. Daraus ergeben sich die Entscheidungen darüber, ob die Mitarbeiterin oder der Mitarbeiter durch bereits erlernte Erfahrungen für die Ausführung eines Prozesses geeignet ist oder ob ein Anlernen erfolgen muss. Im ersten Schritt wird die formelle Qualifikation von Beschäftigten geprüft. Wenn für einen Arbeitsprozess eine formelle Qualifikation erforderlich ist (Ausbildung, Prüfung oder Ähnliches), dann wird dieser Arbeitsprozess nur den hierfür zugelassenen Beschäftigten zugewiesen. Nach der Prüfung formeller Qualifikationen werden die Anforderungen an Anlagen, Arbeitsplätze und Aufgabenfeld überprüft. Danach folgt die Prüfung über besondere Operationen und die Benutzung von Betriebsmitteln (siehe unten).

1. Prüfung formeller Qualifikationen (Ausbildung, Prüfung usw.)
2. Prüfung formeller Anforderungen (für Anlage, Arbeitsplatz zugelassen)
3. Prüfung besonderer Operationen (z. B. Schweißen, Löten, Verkabelung)
4. Prüfung Benutzung von Betriebsmitteln (z. B. Schweißgerät, Prüfgerät, Werkzeuge)

Der gesamte Montageprozess für das Fernsteuerungsauto setzt sich z. B. aus den folgenden Grundoperationen nach Refa [Refa93] zusammen: Greifen, Bringen, Hinlangen, Fügen, Loslassen, Drücken, Drehen, Trennen, Blicken, Prüfen, Körperdrehung, Fußbewegung, Beinbewegung, Bücken. Es existieren darüber hinaus jedoch besondere Operationen, die sich aus der Zusammensetzung von Grundoperationen, in der Regel im Zusammenschluss mit der Verwendung von Betriebsmitteln, ergeben, für die die Beschäftigten entsprechend geschult sein müssen. Im Rahmen der Montage des Fernsteuerungsautos sind dies die Verkabelung der Elektronik und das Verlöten der Motorenkabel. Bei beiden Operationen kann nicht erwartet werden, dass Beschäftigte diese bei der ersten Ausführung des Montageprozesses beherrschen. Zusätzlich kann es besondere Betriebsmittel geben, mit denen Beschäftigte bisher noch nicht umgehen können. Dies könnte beispielsweise eine besondere Schweißanlage sein, für die eine Einweisung erforderlich ist. Für die Montage des Fernsteuerungsautos wird somit im ersten Schritt überprüft, ob die formellen Anforderungen erfüllt werden. Bei Tätigkeitsfeldern, wie zum Beispiel Staplerfahren, können Zulassungen oder besondere Qualifikationen notwendig sein, die eine Zuweisung ausschließen. Anschließend wird geprüft, ob auf Grund besonderer Operationen Anforderungen an die Fähigkeiten der Beschäftigten bestehen. Die Mitarbeiterin oder der Mitarbeiter, mit den bestbewerteten benötigten Fähigkeiten und welche gleichzeitig freie Kapazität haben, wird diesem Auftrag zugewiesen.

Die Abbildung 31 stellt das digitale Abbild mit den Bestandteilen dar. Die Statusdaten beinhalten unter anderem die Zeiten (z.B. die Bearbeitungszeiten und die

Abwesenheitszeiten), die Fähigkeiten und Kenntnisse bezogen auf z.B. die formelle Qualifikation, die Anlagen und Aufzeichnungen im Wesentlichen über Zustände, Prozesszeiten und Prozessqualität. Kenntnisse beschreiben hierbei das Wissen über ein Thema, währenddessen Fähigkeiten deren Anwendung widerspiegeln. Auf die Auswertung dieser Daten wird im folgenden Abschnitt im Rahmen der Vorstellung der digitalen Repräsentation vertiefend eingegangen.

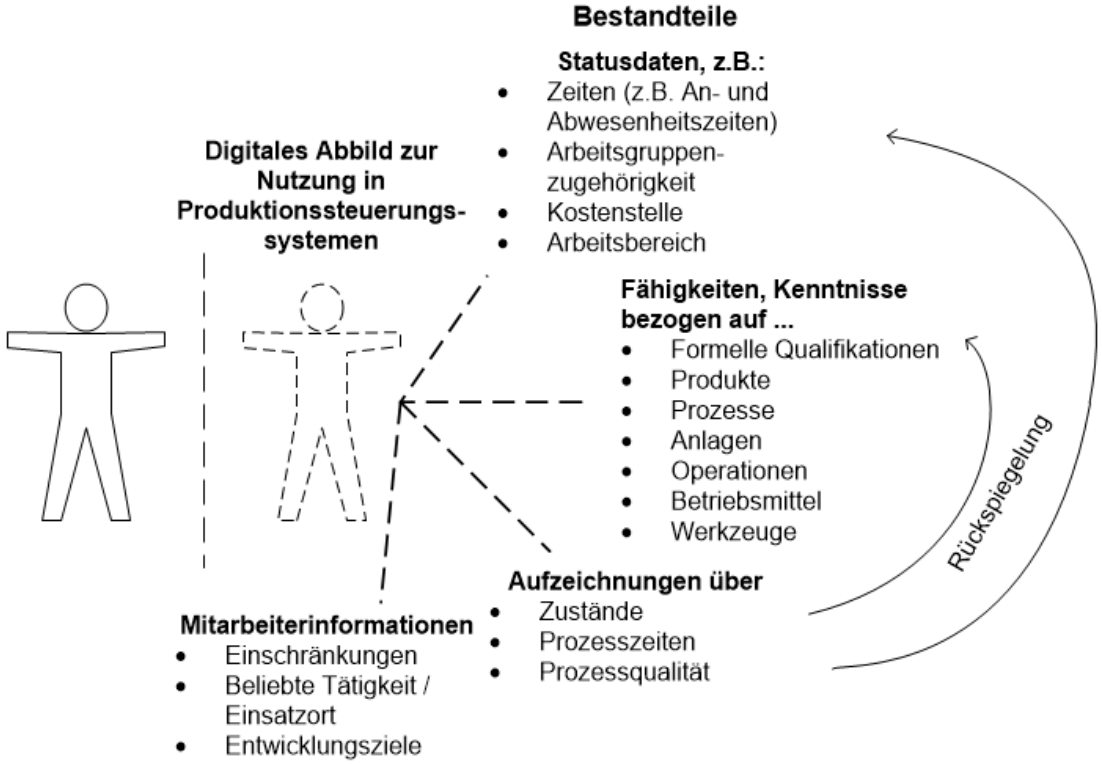

Abbildung 31: Informationen über den Menschen im digitalen Abbild

5.4 Entwicklung einer digitalen Repräsentation zur Einbindung in CPPS

Das Abbild der Beschäftigten erfasst alle für die Produktionssteuerung notwendigen Daten und Informationen über die Beschäftigten. Die digitale Repräsentation greift auf das Abbild zu und analysiert das Verhalten von Beschäftigten, um für die Prozessausführung relevante Entscheidungen und Parameter zu prognostizieren. Ein Ziel ist es hierbei, die Planungs- und Steuerungsentscheidungen anhand genauerer Daten und der Vorhersage des Verhaltens von Beschäftigten zu optimieren. Zudem soll für die Beschäftigten die Eingriffsmöglichkeiten in die Entscheidungen ermöglicht werden, so dass die Beschäftigten ihre eigenen Wünsche bezüglich des Produktionsplans an das Produktionssystem richten können. Daher wurde in dieser Arbeit das Verfahren der digitalen Repräsentation der Beschäftigten entwickelt, die dann zusammen mit der dezentralen Produktionssteuerung in das CPPS integriert wird. Im Folgenden wird auf die entwickelte digitale Repräsentation der Beschäftigten vertiefend eingegangen.

Konzept der digitalen Repräsentation von Beschäftigten

Der Neuheitsgrad der digitalen Repräsentation besteht darin, dass diese für die Beschäftigten im Rahmen der dezentralen Produktionssteuerung agiert. Das heißt, dass diese mit allen anderen Arbeitsstationen und digitalen Repräsentationen über Steuerungsentscheidungen verhandelt. Dafür wird auf die Daten vom digitalen Abbild zurückgegriffen (siehe Abbildung 32). Darauf basierend werden z.B. das Verhalten von Beschäftigten oder die Bearbeitungszeiten an bestimmten Aufträgen prognostiziert. Somit können Verhandlungen zwischen allen involvierten digitalen Repräsentationen der Beschäftigten und Arbeitsstationen stattfinden. Dadurch werden die Beschäftigten zu den entsprechenden Aufträgen und Arbeitsstationen zugewiesen. Das Konzept der digitalen Repräsentation von Beschäftigten wird in Abbildung 32 visualisiert.

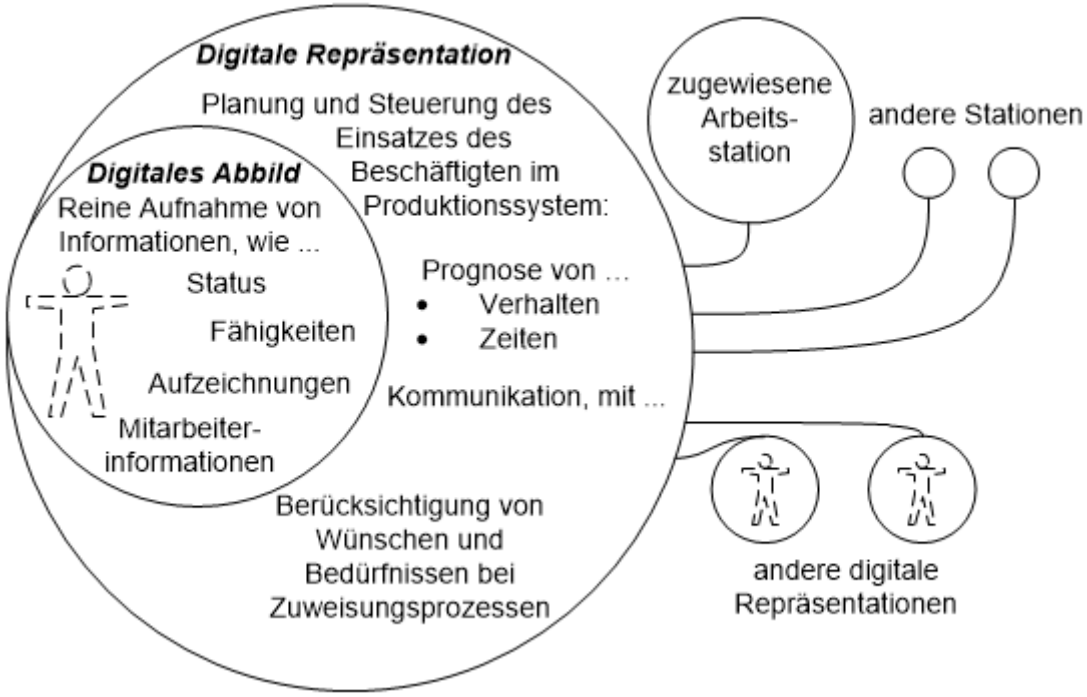

Abbildung 32: Konzept der digitalen Repräsentation der Beschäftigten

Die Verhandlungsfrage „wer arbeitet an welcher Arbeitsstation“ entspricht der Aufgabe der Personaleinsatzplanung. Es werden normalerweise Zuweisungsregeln von der Produktionsplanung festgelegt, die beispielsweise vorgeben, dass bestimmte Beschäftigte an bestimmten Arbeitsstationen arbeiten. Diese Personaleinsatzplanung wird von dem Produktionsplan an die Produktionssteuerung initial übertragen. Nach dem heutigen Stand der Technik wird die Personaleinsatzplanung während der Ausführung der Produktion in der Regel nicht mehr verändert. Allerdings kommen Anpassungs- bzw. Änderungsbedarfe bezüglich der Personaleinsatzplanung häufig vor. Zum Beispiel passiert dies, wenn Beschäftigte neu eingeplant werden müssen oder wenn an ihren eigentlichen Arbeitsstationen gerade keine Aufträge vorhanden sind. Erkrankt ein Mitarbeiter kurzfristig oder fällt anderweitig aus, muss für Ersatz gesorgt werden. Daraus

resultierende Änderungsaufwände stellen große Herausforderungen dar. Der Einsatz der digitalen Repräsentation automatisiert solche Einsatzfälle während der Produktionsausführung.

Dieser automatisierte Zuweisungsprozess bildet das Hauptelement der digitalen Repräsentation. Dabei wird die oder der am besten geeignete Mitarbeiterin oder Mitarbeiter für die Auftragsbearbeitung ausgewählt und einer entsprechenden Arbeitsstation zugewiesen. Dafür werden die Eigenschaften der Beschäftigten aus der digitalen Repräsentation auf die Arbeitsstation übertragen. Die Bearbeitungszeiten werden dann abgeschätzt und damit werden die Planung und Steuerung an dieser Arbeitsstation optimiert. Im Folgenden wird vertiefend auf den Zuweisungsprozess eingegangen.

Konzept des Zuweisungsprozesses mithilfe der digitalen Repräsentation

Derzeit werden Beschäftigte einfach Arbeitsstationen zugeordnet und im Rahmen der Produktionsplanung als ein Teil der Arbeitsstation behandelt. Die digitale Repräsentation setzt hier an, so dass sie für Beschäftigte in der dezentralen Produktionssteuerung mit anderen Produktionselementen agiert. Das ist ähnlich wie bei der Verteilung von Aufträgen zu Arbeitsstationen im Rahmen der dezentralen Produktionssteuerung (siehe Kapitel 4). Bei der digitalen Repräsentation wird hierfür zuerst auf die Daten vom digitalen Abbild zugegriffen, um unter anderem das Verhalten und die Bearbeitungszeiten von Beschäftigten abschätzen zu können. Gleichzeitig werden die Bedürfnisse und Wünsche bezüglich z.B. Pausenzeiten von der digitalen Repräsentation berücksichtigt.

Die Abbildung 33 veranschaulicht das Konzept des Zuweisungsprozesses mithilfe der digitalen Repräsentation. Hierin wird veranschaulicht, dass ein Mitarbeiter, der eigentlich einer Montagestation zugewiesen wurde, gerade ausfällt. Dieser Ausfall wird von seiner digitalen Repräsentation der Montagestation mitgeteilt. Die Montagestation benötigt für die Ausführung des Produktionsprozesses einen neuen Beschäftigten. Sie schickt dann die entsprechende Anfrage an alle digitalen Repräsentationen. Diese digitalen Repräsentationen melden ihre eigenen Zeitpläne und ihre derzeitigen Zuweisungen der Montagestation zurück. Die Montagestation startet dann den Verhandlungs- bzw. Auswahlprozess. Das entsprechende Ablaufdiagramm wird in Abbildung 33 dargestellt. Falls keine passenden Beschäftigten gefunden werden können, wird der Auftrag an dieser Arbeitsstation auf andere Arbeitsstationen verteilt. Gleichzeitig wird versucht, andere Beschäftigte, die anderen Arbeitsstationen zugewiesen sind, der Station zuzuweisen. Wenn dabei trotzdem keine Beschäftigten mit freier Kapazität vorhanden sind, werden Verhandlungen mit anderen zugewiesenen Arbeitsstationen über diese Beschäftigten geführt.

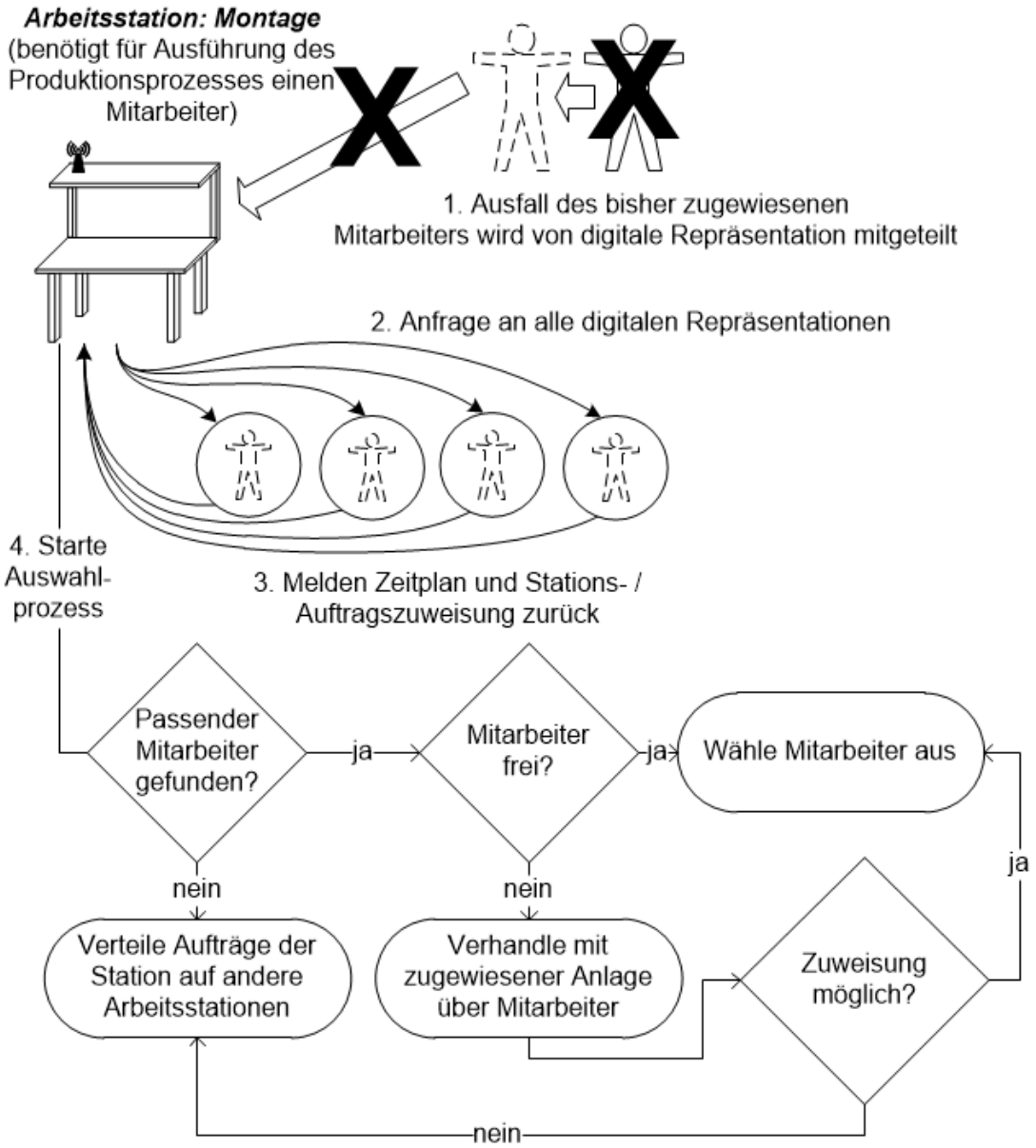

Abbildung 33: Zuweisung eines Mitarbeiters zu einer Arbeitsstation mithilfe der digitalen Repräsentation von Beschäftigten

Die Arbeitsstationen verhandeln untereinander über Zuweisungen der digitalen Repräsentationen und damit über Aufträgen, die an der Arbeitsstation ausgeführt werden. Sobald unvorhergesehene Ereignisse bei Beschäftigten auf Arbeitsstationen eintreten, wird eine solche Verhandlung gestartet. Ein unvorhergesehenes Ereignis kann zum Beispiel das Verlassen eines Arbeitsplatzes sein. Die Mitarbeiterin oder der Mitarbeiter muss anschließend angeben, wie lange er abwesend sein wird, damit die Arbeitsstation prüfen kann, ob für die Abwesenheitszeit Ersatz eingeplant werden soll.

Im Folgenden wird anhand eines Szenarios der Verhandlungsprozess, der das Grundelement des Zuweisungsprozesses bildet, konkretisiert. Die Abbildung 34

veranschaulicht diesen Verhandlungsprozess. Erkrankt zum Beispiel ein Mitarbeiter während des Betriebs und dessen Arbeitsstation weist einen Engpass in der Produktion auf, muss schnell für Ersatz gesorgt werden.

Zuerst fragt die Arbeitsstation alle anderen digitalen Repräsentationen an, ob jemand diesen Prozess ausführen kann. Ist eine passende Mitarbeiterin oder ein passender Mitarbeiter gefunden, wird der Zeitplan und die Situation von der Anlagen- / Auftragszuweisung dieses Beschäftigten geprüft, ob bei ihm entsprechend freie Kapazitäten vorhanden sind. Wenn nicht, wird die Verhandlung an eine andere, baugleiche Arbeitsstation weitergeleitet. Es wird verhandelt, ob an dieser Arbeitsstation Prozesse zuerst nach hinten geschoben werden können, um den Beschäftigten der Arbeitsstation zuzuweisen. Sobald diese Anfrage von der Arbeitsstation identifiziert ist, teilt sie dies der digitalen Repräsentation des Beschäftigten mit. Dieses Szenario zeigt, wie die digitale Repräsentation in die dezentrale Produktionssteuerung eingebunden ist. Die digitale Repräsentation repräsentiert die Beschäftigten in den Verhandlungen zwischen den Arbeitsstationen.

In dem Zuweisungsprozess basiert die Auswahl der Beschäftigten auf dem Prüfprozess für die Eignung der Beschäftigten. Dafür werden die historischen Daten und Fähigkeitsprofile als Grundlage genommen (siehe Abschnitt 5.2). Wurde der geforderte Prozess bereits einmal ausgeführt, wird angenommen, dass auch eine Eignung für diesen Prozess besteht. Die Daten werden anschließend für die Berechnung der Bearbeitungszeiten verwendet. Dabei wird der Median der letzten 50 Ausführungen des Prozesses als Berechnungsgrundlage genommen. Wurde der Prozess bisher noch nie an der Arbeitsstation ausgeführt, wird geprüft, ob ein ähnlicher Prozess bereits ausgeführt wurde. Dies kann zum Beispiel die Ausführung des gleichen Prozesses an einer anderen Arbeitsstation sein. Hierbei werden die Arbeitspläne zwischen den Prozessen verglichen. Existieren keine historischen Daten, werden die Fähigkeitsprofile der Beschäftigten und die Anforderungen der Aufträge miteinander verglichen. Das Fähigkeitsprofil von Beschäftigten wird einmalig festgelegt und danach basierend auf den historischen Daten kontinuierlich erweitert. In dem Fähigkeitsprofil werden die einzelnen Arbeitsschritte aus der Produktion mit Fähigkeiten der Mitarbeiterin, bzw. des Mitarbeiters verglichen. Anhand der Häufigkeit der Ausführung der Arbeitsschritte werden die Fähigkeiten bewertet. Bei der Prüfung der Eignung werden die Arbeitsprozesse auf einzelne Arbeitsschritte heruntergebrochen. Somit wird ein Vergleich zwischen dem Fähigkeitsprofil und den Anforderungen von Aufträgen ermöglicht. Damit kann eine Abschätzung darüber getroffen werden, ob die Mitarbeiterin oder der Mitarbeiter für diesen Arbeitsprozess geeignet ist.

Sobald eine Mitarbeiterin oder ein Mitarbeiter zugewiesen ist, werden die Daten bzw. die Eigenschaften von den Beschäftigten an die Arbeitsstation übergeben. Dann werden die Bearbeitungszeiten für die Produktionssteuerung an der Arbeitsstation berechnet. Diese Zeiten basieren zuerst auf historischen Daten der Beschäftigten. Die Daten der digitalen

Repräsentation müssen somit von der Arbeitsstation interpretiert werden, um daraus die Bearbeitungszeiten für die Anlagen bzw. die Prozesse zu berechnen.

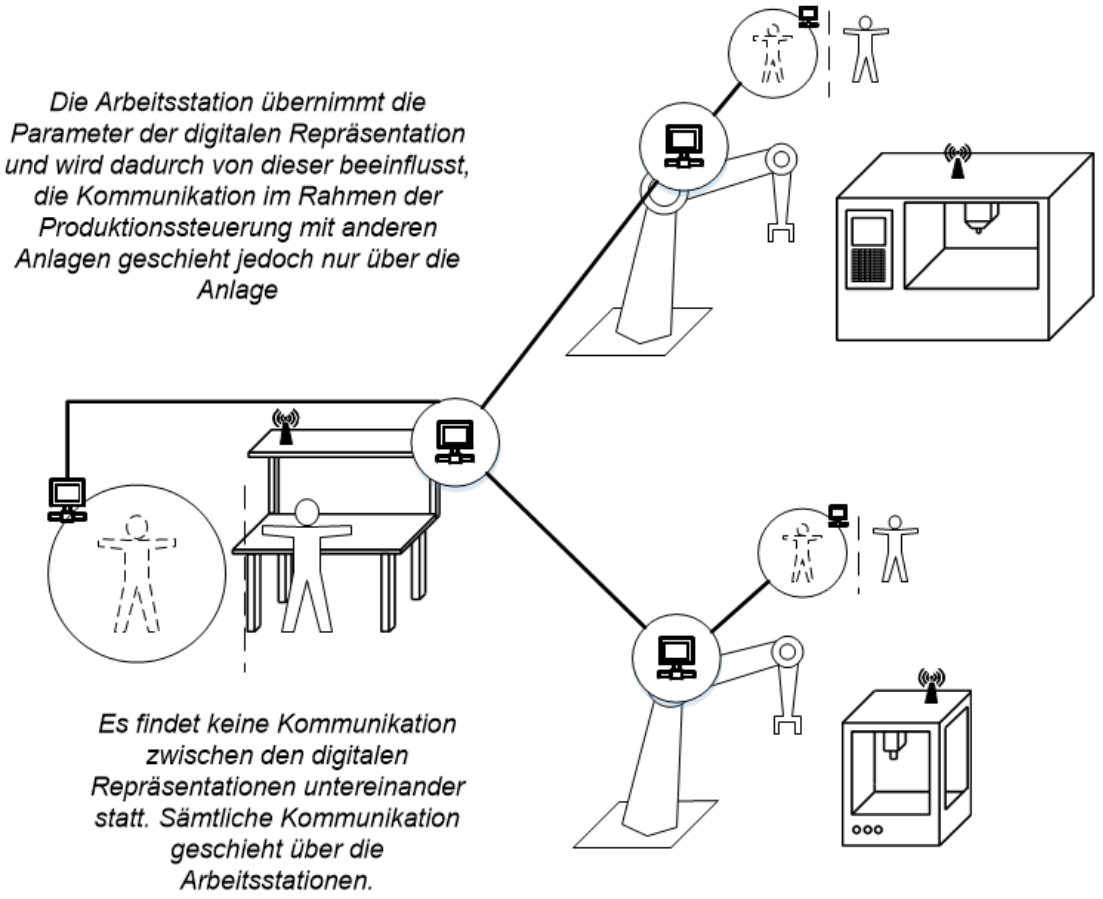

Abbildung 34: Kommunikation zwischen Arbeitsstationen und digitaler Repräsentation

Einsatz der digitalen Repräsentation der Beschäftigten in der Anwesenheitsplanung

Im Rahmen der Anwesenheitsplanung kommen häufig kurzfristige Änderungen an den Schichtplänen der Beschäftigten vor. Derzeit kann eine hierfür notwendige Umplanung nur manuell stattfinden. Das erzeugt einen enormen Aufwand. Durch den Einsatz der digitalen Repräsentation der Beschäftigten wird die aufwändige Umplanung automatisiert. Darauf wird im Folgenden eingegangen.

Die Schichtplanung und die langfristigen Änderungen werden weiterhin im Rahmen der Produktions- bzw. Kapazitätsplanung behandelt. Kommt eine Mitarbeiterin oder ein Mitarbeiter jedoch kurzfristig später zur Arbeit oder muss auf Grund eines Ereignisses dessen Planung geändert werden, führt dies im realen Betrieb zu Störungen bzw. zu manuellen Interventionen. Die Funktion der Anwesenheitsplanung der digitalen Repräsentation kann solche kurzfristigen Änderungen abfangen. Die Beschäftigten können hierzu über ihre digitale Repräsentation die Anwesenheitszeiten anpassen. Dies kann zum Beispiel nach Absprache mit dem Vorgesetzten erfolgen. Die digitale Repräsentation teilt diese Veränderung der Anwesenheit (z. B. Beginn der Tätigkeit um

8:30 statt 8:00 Uhr) der Arbeitsstation mit, die dann darüber entscheidet, ob hierfür weitere Maßnahmen eingeleitet werden müssen.

Im ersten Schritt wird in diesem Zuge geprüft, ob durch diese Änderung ein Auftrag als kritisch einzustufen ist, wenn z.B. dessen Fertigstellung zur Terminfrist gefährdet wird. Können trotz des verspäteten Anfangs alle Aufträge erreicht werden, wird keine Umplanung eingeleitet. Werden Aufträge nicht erreicht, wird versucht, eine Umplanung der Aufträge an der Arbeitsstation umzusetzen. Dementsprechend können Aufträge nach hinten geschoben werden. Ist dies nicht möglich, werden andere Arbeitsstationen angefragt, ob sie die kritischen Aufträge übernehmen können. Im zweiten Schritt werden freie Ressourcen gesucht, indem andere Beschäftigte angefragt werden, ob sie in dieser Zeit einspringen können. Diese Anfrage wird von den kontaktierten digitalen Repräsentationen an ihre Arbeitsstationen weitergeleitet. Die Arbeitsstationen prüfen die Anfrage. Der letzte Schritt besteht im Rückgriff auf weitere kapazitätssteigernde Maßnahmen. Hierzu wird die Funktion der Kapazitätssteuerung durchgeführt.

Diese Anwesenheitsplanung wird durch aktive Maßnahmen der Beschäftigten, also durch eine Meldung an die digitale Repräsentation, hervorgerufen. Gleichzeitig werden Abwesenheitszeiten durch Rückfragen überprüft. Verlässt eine Mitarbeiterin oder ein Mitarbeiter zum Beispiel eine Station, fragt die digitale Repräsentation nach, wie lange die Abwesenheitszeit voraussichtlich andauern wird. Diese Abwesenheitszeit wird dann entsprechend an die Arbeitsstation weitergegeben. Erfolgt keine Rückmeldung, wird davon ausgegangen, dass die Mitarbeiterin, bzw. der Mitarbeiter unmittelbar zur Arbeitsstation zurückkehren wird. Geplante Abwesenheitszeiten werden bereits im Rahmen der Kapazitätsplanung berücksichtigt und sind entsprechend in der digitalen Repräsentation hinterlegt. Für diese Zeiten (z. B. Pausenzeiten) müssen somit keine gesonderten Prozeduren durchgeführt werden.

Einsatz der digitalen Repräsentation zur Berücksichtigung der Wünsche von Beschäftigten

Die digitale Repräsentation ermöglicht die Berücksichtigung der Präferenzen der Beschäftigten bei der Aufgabenzuweisung. Hierbei können Beschäftigten die Arbeitsstationen mit entsprechenden Präferenzen bewerten. Im Rahmen des Abgleichs der Fähigkeiten werden diese Beschäftigten dann im ersten Schritt bevorzugt ausgewählt. Erst wenn kein anderer Beschäftigter mit ähnlichem Qualifikationsprofil für diese Arbeitsstation gefunden wird, wird jemand ohne Präferenzauswahl ausgewählt. Dadurch wird den Beschäftigten ermöglicht, Einfluss auf die Auswahl der Arbeitsstationen zu nehmen.

Eine andere Möglichkeit zur Realisierung der Einflussnahme von Beschäftigten ist die Rückfrage der digitalen Repräsentation nach jeder Entscheidung, die vom Produktionssystem getroffen wurde. Es geht hierbei in erster Linie darum, Feedback zu erhalten, wie Entscheidungen von den jeweiligen Beschäftigten wahrgenommen werden. Insbesondere handelt es sich hierbei um Rückfragen zur Entscheidung bezogen auf die

Zuweisung zu einer Arbeitsstation und zur Übernahme von neuen und unbekannten Aufträgen. Basierend auf den Antworten von Beschäftigten werden dann weitere Präferenzen gebildet, wie zum Beispiel, welche Arbeitsstationen bevorzugt werden und welche Aufträge der Mitarbeiter präferiert. Ein weiteres Element bilden Anfragen von der digitalen Repräsentation an die Beschäftigten. Treten Prozesse zum ersten Mal auf, stellt die digitale Repräsentation zuerst die Frage, ob diese Entscheidung akzeptabel ist. Dadurch wird verhindert, dass Beschäftigte überfordert werden, z. B. wenn Beschäftigte einer neuen Arbeitsstation zugewiesen werden, die noch nicht bedient wurde. Gleichzeitig wird dadurch die Eingriffsmöglichkeit in die Entscheidung des Produktionssystems bzw. CPPS ermöglicht. Ähnlich wie bei Rückfragen wird der Umfang der Anfrage immer geringer. Die ersten drei Male wird eine Anfrage vollständig und ausführlich gestellt. Wird die Anfrage die ersten drei Male angenommen, findet eine vereinfachte Anfrage statt. Erst wenn eine Anfrage abgelehnt wird, erfolgt erneut eine ausführliche Anfrage.

In diesem Kapitel wurde die Entwicklung der digitalen Repräsentation der Beschäftigten vorgestellt. Im Abschnitt 5.1 wurde die Umwandlung der Personaleinsatzplanung durch die Einführung von CPPS vorgestellt. Dafür wurde der Ansatz der digitalen Repräsentation in dieser Arbeit entwickelt (siehe Abschnitt 5.3). Dieser Ansatz basiert auf dem digitalen Abbild von Beschäftigten, welches die steuerungstechnisch relevanten Daten und Informationen speichert (siehe Abschnitt 5.2). Dadurch wird die Personaleinsatzplanung kurzfristig während des Betriebs angepasst. Diese Anpassung findet automatisiert statt. Durch die digitale Repräsentation werden gleichzeitig die Wünsche und Bedürfnisse der Beschäftigten berücksichtigt und ihre Einflussnahme auf die Produktionssteuerung bzw. das Produktionssystem realisiert.

6 Implementierung der Produktionssteuerung im Smart Automation Labor

Die Entwicklung der automatisierten dezentralen Produktionssteuerung wurde in Kapitel 4 und die Entwicklung der digitalen Repräsentation der Beschäftigten in Kapitel 5 vorgestellt. Nachfolgend wird die Implementierung im Smart Automation Labor des Lehrstuhls für Produktentstehung am Heinz-Nixdorf-Institut beschrieben. Zuerst wird auf die Ausgangssituation bezüglich der Ausstattung des Labors eingegangen. Für die Implementierung musste vorab das Produktionssystem zu einem CPPS umgebaut werden. Dadurch wird das Fundament für die Implementierung des in dieser Arbeit entwickelten dezentralen Steuerungssystems geschaffen. Der Umbau des Labors wird im Anhang du den beiden Veröffentlichungen [GTY16, GPP16] ausführlich beschrieben (siehe Anhang A1). Anschließend wurden die automatisierte dezentrale Produktionssteuerung und die digitale Repräsentation im Labor implementiert.

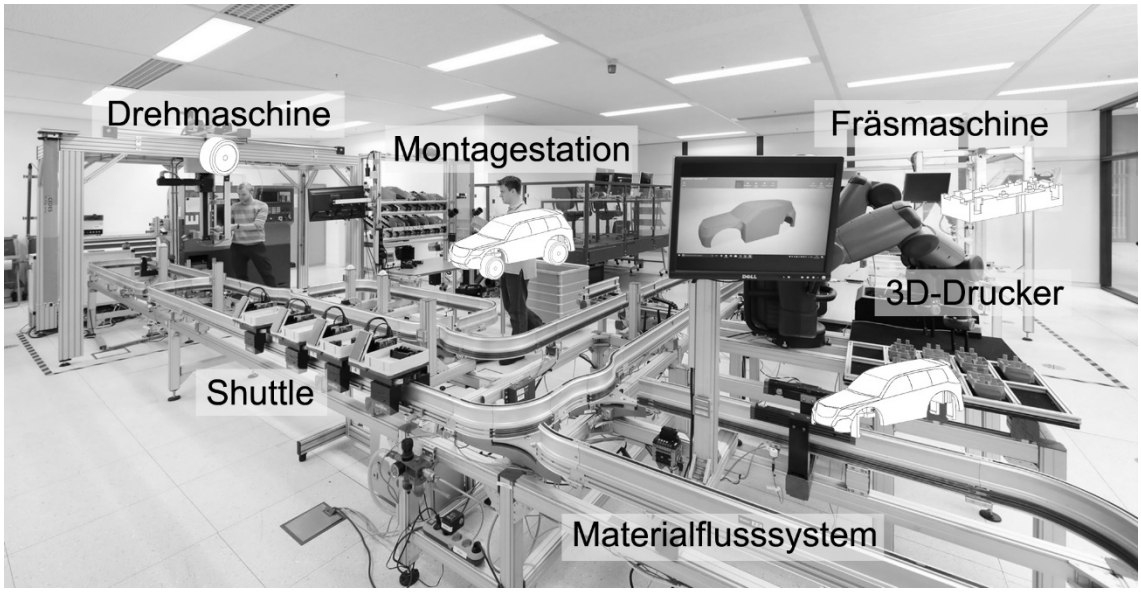

Abbildung 35: Zu CPPS umgebautes Smart Automation Labor

Das Labor dient der Forschung in den Bereichen Produktions- und Automatisierungstechnik sowie deren Anbindung an die Produktentwicklung und indirekt der anwendungsnahen Untersuchung von Geschäftsprozessen. In der Abbildung 35 wird das Labor nach dem Umbau veranschaulicht. Das Labor besteht aus drei Fertigungsstationen (einer Drehstation, einem 3D-Drucker-Station und einer Frässtation) und einer Montagestation. Die einzelnen Arbeitsstationen sind über ein schienengebundenes Materialflusssystem miteinander verbunden. Auf dem Materialflusssystem sind Shuttle im Einsatz, die über Weichen gesteuert zu den einzelnen Arbeitsstationen fahren können. Die Shuttles versorgen die Arbeitsstationen mit Material und transportieren fertig bearbeitete Komponenten und Produkte von den Arbeitsstationen ab. Die Arbeitsstationen sind für die Be- und Entladung mit Robotersystemen ausgestattet. Die Montagetätigkeiten werden von Beschäftigten

ausgeführt. In der Montagestation sind weitere Zukaufteile für den Montageprozess gelagert. Dabei werden Fernsteuerungsautos als Anwendungsszenario bzw. Demonstratoren im Labor produziert.

Vor dem Umbau wurde das Labor über eine zentrale speicherprogrammierbare Steuerung (SPS) des Materialflusssystems gesteuert, die an einen zentralen Computer angeschlossen war (siehe Anhang A1). Die Shuttles und Weichen des Materialflusssystems waren dabei über Profibus untereinander und mit den einzelnen Arbeitsstationen des Labors verbunden. So konnten die Arbeitsstationen gestartet werden und nach der Fertigstellung der Bearbeitungsoperation konnte die gefertigte Komponente vom Materialflusssystem abgeholt werden. Mittels RFID konnte zusätzlich die Position der Shuttle nachvollzogen und so der Status des Systems ermittelt werden. Das Rohmaterial wurde dabei an den Arbeitsstationen gelagert und die fertig bearbeiteten Komponenten wurden von den Robotern entnommen und auf die Shuttles gelegt. Die Shuttles haben anschließend die fertigen Komponenten zur Montagestation transportiert. Dabei wurde die Planung stark vereinfacht und für die Produktion nur kurzzeitige Zeiträume betrachtet, um das gesamte Produktionssystem nur durch einen zentralen Computer steuern zu können. Die Vereinfachung geschah unter der Annahme, dass die Material- und Kapazitätsplanung bereits abgeschlossen sind, also nur Aufträge in der Produktion vorhanden sind, für die auch genug Kapazitäten und Materialien zur Verfügung stehen. Wenn ein neuer Auftrag in das Produktionssystem ankam, wurde der Auftrag einfach an das Ende der Auftragswarteschlange gesetzt und dann entsprechend ausgeführt. Eine solche Art der Produktionssteuerung wird heutzutage noch in vielen Unternehmen eingesetzt [ScSt11].

Darauf basierend wurde das Labor im Rahmen der vorliegenden Arbeit auf ein dezentral gesteuertes CPPS umgebaut. Die Grundidee besteht darin, jede Arbeitsstation des Labors mit einem Computersystem auszustatten. Das Computersystem beschreibt hierbei das System, in dem das in dieser Arbeit entwickelte Steuerungssystem installiert wurde, sodass die Arbeitsstationen untereinander kommunizieren und sich dezentral steuern können. Im Labor wurden hierfür Einplatinencomputer (Raspberry Pis und Beaglebone) und PCs mit IO-Karten eingesetzt. Anschließend wurde durch Verhandlungen zwischen den Arbeitsstationen die Verteilung der Aufträge dezentral realisiert. Dies soll automatisiert geschehen, also ohne, dass Beschäftigte eingreifen müssen oder die Verhandlung von einem externen Ereignis ausgelöst wird. Dies bedeutet auch, dass Störungen zum Beispiel vom Steuerungssystem erkannt und eigenständig beseitigt werden müssen. Dieser Wandel von einem zentral gesteuerten Produktionssystem hin zu einem automatisierten dezentralen CPPS ist in Abbildung 36 konzeptionell visualisiert.

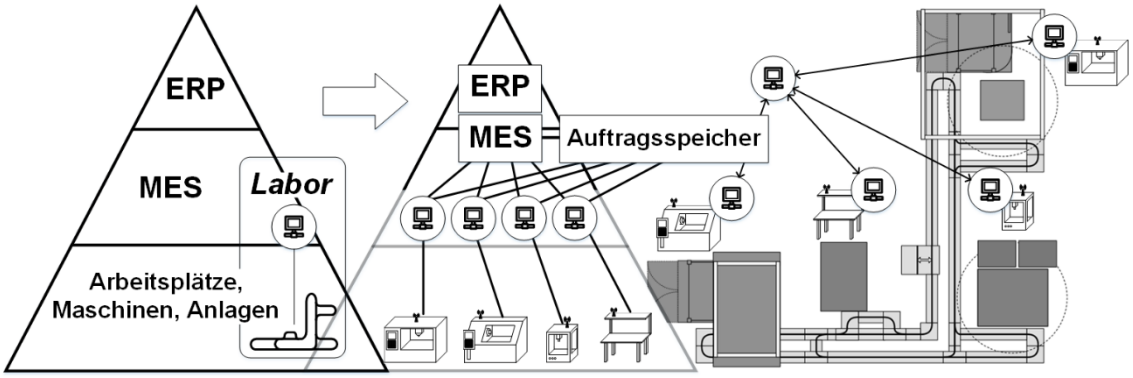

Abbildung 36: Konzept zum Umbau eines zentralen Produktionssystems zu einem CPPS im Smart Automation Labor

Produktionssysteme mit einem zentralen PPS-System werden heutzutage von den meisten Unternehmen eingesetzt. In dem Labor war ein solches PPS nicht notwendig. Allein die Auftragseinlastung, die ein Teil der Produktionssteuerung ist, wurde in dem Labor durch die zentrale SPS vorher umgesetzt (siehe linke Automatisierungspyramide in Abbildung 36). Für den Umbau zu einer dezentralen Produktionsstruktur wird die zentrale Produktionsstruktur bzw. die Automatisierungspyramide aufgelöst. Das heißt, dass die bisherige zentrale Produktionsplanung auf die Arbeitsstationen verteilt werden muss. Gleichzeitig müssen die Arbeitsstationen befähigt werden, selbst Entscheidungen für die Planung und Steuerung der Produktion zu treffen. Dafür wurde die in dieser Arbeit entwickelte automatisierte dezentrale Produktionssteuerung mitsamt digitaler Repräsentation der Beschäftigten eingesetzt. Das Steuerungssystem wurde auf den Computersystemen des Labors installiert. Die Computersysteme wurden über Schnittstellen zu den Arbeitsstationen mit dem realen Produktionsprozess verbunden.

CPPS sind, wie bereits in Kapitel 3 beschrieben, besonders für die Produktion kundenindividueller Aufträge geeignet. Daher wurde für die Implementierung im Labor eine Konfigurationsumgebung entwickelt, mit der die individuellen Aufträge für den Demonstrator des Fernsteuerungsautos erstellt werden können. Hierbei können unterschiedliche Komponenten des Fernsteuerungsautos ausgewählt werden, um das eigene Wunsch-Fernsteuerungsauto zusammenzustellen. Nach der Erstellung des Auftrags, bestünde der nächste Schritt nun darin, diese Aufträge in Kapazitätszeiträume einzuplanen und dementsprechend die Materialplanung vorzunehmen. Dieser Schritt wird im Labor vom dezentralen Steuerungssystem übernommen. Deshalb können Kunden während der Auftragserstellung anhand der vorhandenen Kapazitäten festlegen, in welchem Fertigstellungszeitraum der Auftrag erfolgen soll. Dafür werden die freien Kapazitätszeiträume angezeigt. Für die benötigten Materialien wird stets ein genügend großer Vorrat vorgehalten, sodass keine gesonderte Materialplanung erfolgen muss. Neben der Menge, Spezifikation und dem Produktionszeitraum gibt der Kunde noch seine Daten und weitere Anforderungen an, woraufhin das System prüft, ob dieser Auftrag gefertigt werden kann. Neben den Kapazitäten und Beständen wird die Plausibilität des Auftrags geprüft. Anschließend werden die Produktionsaufträge erzeugt. Diese

Produktionsaufträge werden in einem Auftragsspeicher abgelegt, auf den die einzelnen Arbeitsstationen zugreifen können.

Dieser Prozess zur Erzeugung der Aufträge wird in Abbildung 37 verdeutlicht. Der Kunde bestellt bzw. konfiguriert dabei ein individuelles Fernsteuerungsauto. Dann wird geprüft, ob die Anforderungen des Kunden erfüllt werden können. Wenn ja, wird anschließend der entsprechende Kundenauftrag erzeugt, der die folgenden Informationen enthält: Bezeichnung, Kundendaten, Spezifikation, Termin, Versand und Menge. Dieser Auftrag wird in den Auftragsspeicher gelegt, worauf die einzelnen Arbeitsstationen zugreifen können. Anschließend wird der Kundenauftrag von der dezentralen Produktionssteuerung in Produktionsaufträge aufgelöst und unter den Bearbeitungsstationen verhandelt. Das Chassis wird von der Frässtation bearbeitet. Die Karosserie wird vom 3D-Drucker erstellt und die Räder von der Drehstation. An die Montagestation werden alle fertigen Komponenten geliefert und diese werden mit Zukaufteilen an der Montagestation zusammengebaut.

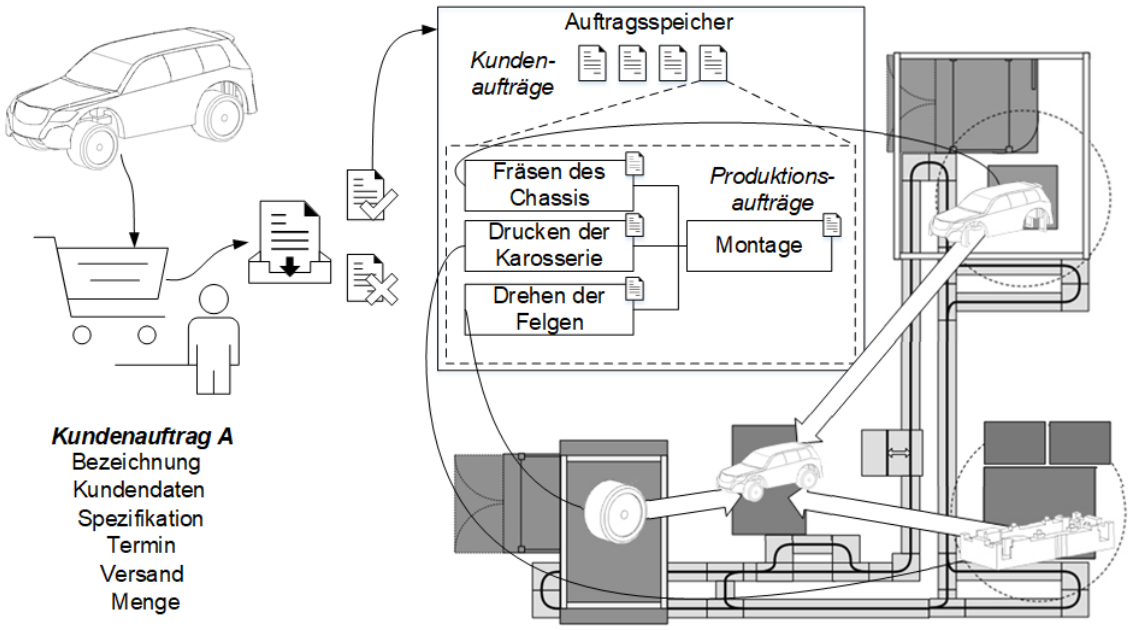

Abbildung 37: Prozess zur Erzeugung der Produktionsaufträge

6.1 Umbau des Smart Automation Labors zu einem CPPS

In diesem Abschnitt wird der Umbau des Labors zu einem CPPS vorgestellt. Dabei werden unter anderem die Auswahlentscheidungen der eingesetzten Soft- und Hardware beschrieben. Die detaillierte Beschreibung des Umbaus ist in Anhang A1 beschrieben. Als Programmiersprache wurde auf Grund der Plattformunabhängigkeit (im Labor werden sowohl linux- als auch windowsbasierte Computersysteme eingesetzt) Python ausgewählt. Mit Python ist es zudem möglich, auch große Projekte schnell und strukturiert zu programmieren. Die Einbindung von C-Code ermöglicht darüber hinaus leistungsfähige Programme, ohne dass das gesamte Programm umständlich in C

geschrieben werden muss. Aufgrund der weiten Verbreitung von Python existieren außerdem bereits zahlreiche Bibliotheken, auch bezogen auf mathematische Operationen und künstliche Intelligenz, die eine Auswertung und Einbindung deutlich vereinfachen. Als Datenbanksystem wurde angesichts der einfachen Verfügbarkeit und der weitverbreiteten freien Verwendung MySQL gewählt. Wegen den unterschiedlichen Anforderungen der Produktionselemente wurden verschiedene Computersysteme eingesetzt. Raspberry Pis wurden im Labor am häufigsten angewendet. Teilweise (z. B. bei den Shuttles) waren Einplatinencomputer aufgrund des Bauraums und des Stromverbrauchs notwendig. In einigen Arbeitsstationen haben die Anforderungen an die Computersysteme zur Verwendung der Einplatinencomputer ausgereicht. Auf Grund der höheren Kosten und der geringen Anforderungen an die Prozesssicherheit und industrielle Einsatzfähigkeit wurden keine industriellen Computerlösungen eingesetzt. IoT-Systeme, wie zum Beispiel „Siemens Mindsphere“ oder „Fanuc Field System“, waren zum Zeitpunkt des Umbaus des Labors noch nicht verfügbar. Für zwei Arbeitsstationen wurde aufgrund der höheren Anzahl an IO-Eingängen ein Beaglebone ausgewählt. Darüber hinaus wurden mehrere Windows-PCs als Computersysteme eingesetzt, auf denen unter anderem Dienste und auch die Datenbankapplikationen ausgeführt wurden. Aufgrund der Vereinfachungen bezogen auf die Produktionsplanung im Labor, wurde kein zusätzliches ERP-System oder MES angeschafft.

Die Softwarelösung für die Produktionssteuerung ist in drei Applikationen aufgeteilt. Die Applikationen wurden für die folgenden Produktionselement geschrieben [GP18]:

- Arbeitsstation: Bildet das Hauptmodul der Produktion und übernimmt die Produktionssteuerung und die Ausführung der Bearbeitungsoperationen.
- Logistiksystem: Transportiert Material, Komponenten, Baugruppen oder Produkte von A nach B und lagert diese. Das unmittelbare Handling von Werkstücken an der Arbeitsstation wird jedoch von der Arbeitsstation selbst ausgeführt. Für diese wird in der Regel kein eigenes Logistiksystem definiert.
- Dienste: Nicht direkt in das Produktionssystem eingebundene Elemente werden als Dienste definiert. Diese können zum Beispiel die Arbeitsstationen und Logistiksysteme beeinflussen und Informationen liefern oder diese steuern.

Jedes Computersystem wurde dabei einem Produktionselement (Arbeitsstationen, Logistiksystem, Lagerstation) des Produktionssystems zugeordnet. Aus programmiertechnischen Gründen und zur Verringerung des Aufwands wurde die Möglichkeit ausgeschlossen, dass ein Computersystem mehrere Anlagen steuert. Darüber hinaus arbeiten Beschäftigte in dem Produktionssystem. Die Beschäftigten werden im Rahmen der Steuerung durch die digitale Repräsentation in das Produktionssystem eingebunden. Insgesamt ergeben sich hieraus 17 Computersysteme, die als aktive Produktionselemente im Labor umgesetzt wurden. Diese Computersysteme sind in Abbildung 38 verdeutlicht. Im Folgenden wird die Implementierung der dezentralen Produktionssteuerung an den Arbeitsstationen detailliert vorgestellt.

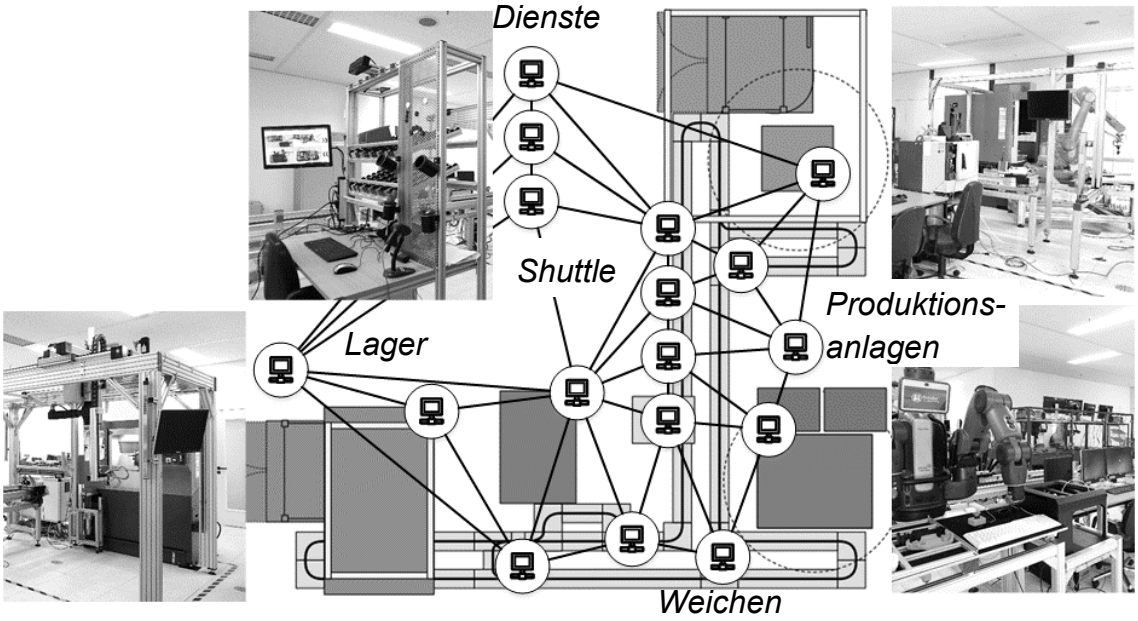

Abbildung 38: Implementierung der automatisierten dezentralen Steuerung im Labor

Implementierung an Arbeitsstationen

Die Arbeitsstationen sollen als eigenständige Systeme im CPPS untereinander interagieren. Dafür wurde die in dieser Arbeit entwickelte dezentrale Produktionssteuerung in entsprechenden Computersystemen, die an die jeweiligen Arbeitsstationen angebaut wurden, installiert. Über Schnittstellen wird die Produktion an der jeweiligen Arbeitsstation von dem Computersystem gesteuert. Die Erweiterung der Arbeitsstationen auf Schnittstellenebene ist in Abbildung 39 dargestellt. Die einzelnen Arbeitsstationen sind über ein lokales Netzwerk miteinander verbunden, wobei die Verbindung zu den Anlagen der Arbeitsstationen über unterschiedliche Schnittstellen geschieht, beispielsweise über analoge und digitale Ein- und Ausgänge, Feldbussysteme oder eigene Standards. Zusätzlich wurden weitere Sensoren und Aktoren angeschlossen. Bei der Frässtation sind zum Beispiel direkt der Industrieroboter über ein Feldbussystem, die Fräsmaschine direkt über Ethernet und die Materialflussstation über Ein- und Ausgangssignale eingebunden. Zusätzlich wurden weitere Sensoren und Aktoren eingesetzt, die direkt vom Computersystem gesteuert werden. Sensoren und Aktoren bilden in der Frässtation unter anderem die Überwachung und Steuerung des Materialvorrats dieser Arbeitsstation und des Materialflusssystems, die über Stellglieder und Näherungssensoren gesteuert werden. Dieses Computersystem und damit die Erweiterung der Arbeitsstation werden in dieser Arbeit als Aufwertung zu einem cyber-physischen Gerät (englisch cyber-physical device – CPD) bezeichnet. Alle vier Arbeitsstationen im Smart Automation Labor (3D-Drucker-Station, Frässtation, Drehstation und Montagestation) wurden entsprechend mit einer CPD-Erweiterung ausgestattet.

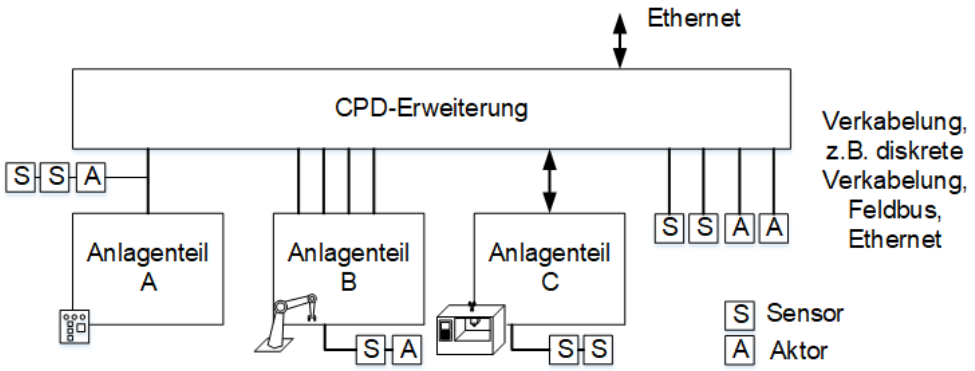

Abbildung 39: CPD-Erweiterungen der Arbeitsstationen

Konkret für eine Maschine umgesetzt, verdeutlicht dies das folgende Beispiel der Fräsmaschine. Die Fräsmaschine von EMCO ist über eine direkte Schnittstelle mit dem Roboter und auch direkt über eine Ethernet-Schnittstelle mit der CPD-Erweiterung verbunden. Die Steuerung der Fräsmaschine, also zum Beispiel die Befehle zum Starten und Beenden eines Programms, werden von der Robotersteuerung ausgeführt. Ausgewählt wird das Fräsprogramm durch die CPD-Erweiterung. Die Steuerung des Produktionsprozesses über die CPD-Erweiterung erfolgt über Befehle an den Roboter, der über Profibus an die CPD-Erweiterung angebunden ist. Direkt von der CPD-Erweiterung werden weitere Teile des Prozesses, wie das eigene Materiallager und das Materialflusssystem, gesteuert. Über einen Monitor wird zudem auf einer grafischen Oberfläche der Status der Arbeitsstation für die Beschäftigten dargestellt.

Implementierung an der Montagestation

Für die Implementierung der Produktionssteuerung an der Montagestation müssen zusätzliche Funktionen zur Steuerung des Montageprozesses umgesetzt werden. Die Produktionssteuerung unterscheidet sich nicht zwischen Montagestationen und anderen Arbeitsstationen. Der Montageprozess wird jedoch von einer Mitarbeiterin oder einem Mitarbeiter manuell ausgeführt. Dies führt dazu, dass die Prozesszeiten in der Montage schwanken und außerdem von dem eingesetzten Personal abhängig sind. Die Fertigungsstationen werden über Befehle und Signale von dem Computersystem gesteuert und überwacht. Dies ist bei der Montagestation nicht ohne weiteres möglich. Für die Steuerung der Montagestation wurde eine Montageschrittführung installiert. Die Montageschrittführung erfasst dabei zusätzlich den aktuellen Status des Montagesystems und den aktuellen Stand der Prozessausführung. Für die Prozesskontrolle wurde zudem eine Betriebsdatenerfassung realisiert, in deren Rahmen ein Mensch die Aufträge anmeldet und entsprechend der Montageschrittführung ausführt. Für die Einbindung der Beschäftigten wurde die digitale Repräsentation auf einem weiteren Computersystem installiert. Die Montageschrittführung und der Aufbau des Montagesystems sind in Abbildung 40 dargestellt.

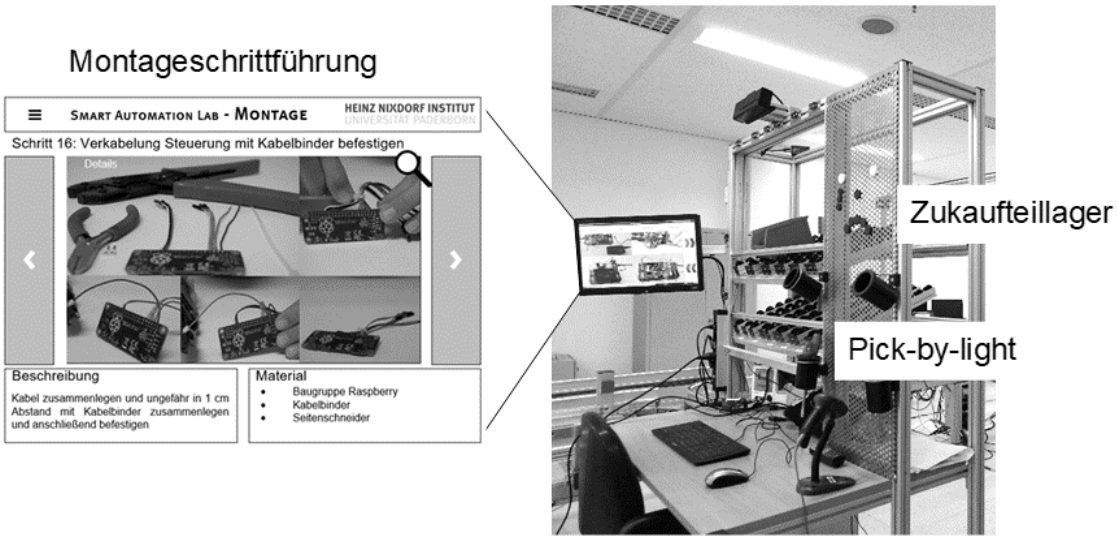

Abbildung 40: Elemente der Prozesskontrolle der Montagestation

Die Benutzeroberfläche der Montageschrittführung besteht aus einer Liste der relevanten Montageteile, einer Darstellung der aktuellen Aufgabe als Bild und einer stichpunktartigen Beschreibung des Montageschrittes. Zudem können allgemeine Darstellungen wie Übersichtszeichnungen und Stücklisten der Baugruppe angezeigt werden. Die Modellierung der Montagepläne findet hierbei über definierte Zuweisungen und ein eigens entwickeltes, auf BPMN basierendes Prozessmodellierungstool statt, in das Montageprozesse importiert werden können. Das Prozessmodellierungstool bietet zudem eine vereinfachte Möglichkeit zur Erstellung der Montageprozesse an. Dabei werden die Bilder der einzelnen Montageschritte über das Kamerasystem aufgenommen. Zudem übernimmt das Kamerasystem die Prozess- bzw. Qualitätskontrolle. Über definierte Merkmale werden Abgleiche zwischen den Bildern vorgenommen, um Fehler bei der Montage zu erkennen.

Bei dem Umbau wurde die Möglichkeit zur Erweiterung des Produktionssystems bzw. der Arbeitsstationen in der Simulationssoftware geschaffen. Die Simulation weiterer Arbeitsstationen kann über die virtuelle Abbildung von entsprechenden Zeiten in der Simulationssoftware simuliert werden. Für die von Menschen bedienten Arbeitsstationen wurde ein Simulator entwickelt, der ähnlich wie ein Terminal zur Betriebsdatenerfassung funktioniert. Zum Start und zum Abschluss eines Auftrags müssen jeweils zwei unterschiedliche Knöpfe gedrückt werden. Über eine LCD-Anzeige werden die zu bearbeitenden Aufträge angezeigt. Ansonsten sind die simulierten Arbeitsstationen genauso wie die vier bereits im Labor stehenden Arbeitsstationen mit dem Produktionssteuerungsprogramm ausgestattet und damit im Produktionssystem eingebunden.

Implementierung im Logistiksystem

Allgemein wird die Logistik unabhängig von der Produktion geplant. Logistikelemente sind alle Elemente, die Material lagern oder transportieren, dieses aber nicht verändern. Eine Arbeitsstation nimmt eine Bearbeitung von Ausgangsmaterialien hin zu einer Komponente, einer Baugruppe oder einem Produkt vor. Logistikelemente speichern und bewegen dieses Material nur und nehmen keine eigenständige Veränderung vor. Darüber hinaus besitzen Logistikelemente eigene Routinen und Verhandlungen. Im Smart Automation Labor werden die Logistikelemente zum Abtransport der fertiggestellten Komponenten von den einzelnen Arbeitsstationen zur Montagestation angefragt. Die Arbeitsstationen haben zudem jeweils eigene Lager, die befüllt werden müssen. Die Logistikelemente verhandeln dies untereinander und das nächste freie Logistikelement führt diesen Auftrag aus. Im derzeitigen Logistiksystem im Smart Automation Labor ist dies leicht umzusetzen, da jeweils nur eine Einheit transportiert und dann zur Montagestation befördert wird.

Im Labor existieren neben den Shuttles noch Weichen als Logistikelemente. Die Weichen steuern die eigentliche Fahrt der Shuttles und dienen somit als Ankerpunkte bzw. Kontrollpunkte. Über Barcodes haben die Shuttles Kenntnis darüber, wo sie sich befinden und welche Weiche sie als Nächste ansprechen können. Die Weiche prüft lediglich, ob die Arbeitsstation frei ist und lässt dann das Shuttle entsprechend in die Arbeitsstation einfahren. In den Logistikelementen wurden bereits Lieferstationen und Übergabepunkte eingegeben, wodurch sie sich eigenständig koordinieren können und einen Fahrtweg zu ihrem Ziel finden. Eine Wege- oder Routenplanung ist für die Umsetzung im Smart Automation Labor nicht erforderlich.

Neben den Shuttles und Weichen wurde eine Lagerstation im Labor für den Speicher des Rohmaterials und der Zukaufteile gebaut. Für die Lagerstation wurde ebenso die in dieser Arbeit entwickelte dezentrale Produktionssteuerung installiert. Dadurch ist die Lagerstation in der Lage, mit anderen Arbeitsstationen zu kommunizieren. Die Lagerstation funktioniert grundsätzlich auf die gleiche Weise wie andere Arbeitsstationen. Der einzige Unterschied besteht darin, dass in der Lagerstation nur der Warenbestand gesteuert wird und keine Produktionsprozesse ausgeführt werden.

6.2 Implementierung der automatisierten dezentralen Produktionssteuerung

Bisher wurde der Umbau des ursprünglich zentralen Produktionssystems im Labor auf ein CPPS vorgestellt. Die Implementierung der automatisierten dezentralen Produktionssteuerung mit digitaler Repräsentation der Beschäftigten erfolgt über die Ausstattung aller Arbeitsstationen mit Computersystemen, auf denen die entsprechende Software bzw. das automatisierte dezentrale Steuerungssystem installiert ist. Die Software der dezentralen Produktionssteuerung besteht dabei aus unterschiedlichen

Modulen und Klassen zur Prozesskontrolle, zur Auftragssteuerung und zur Kommunikation und Koordination und zu weiteren Diensten (siehe Abbildung 41).

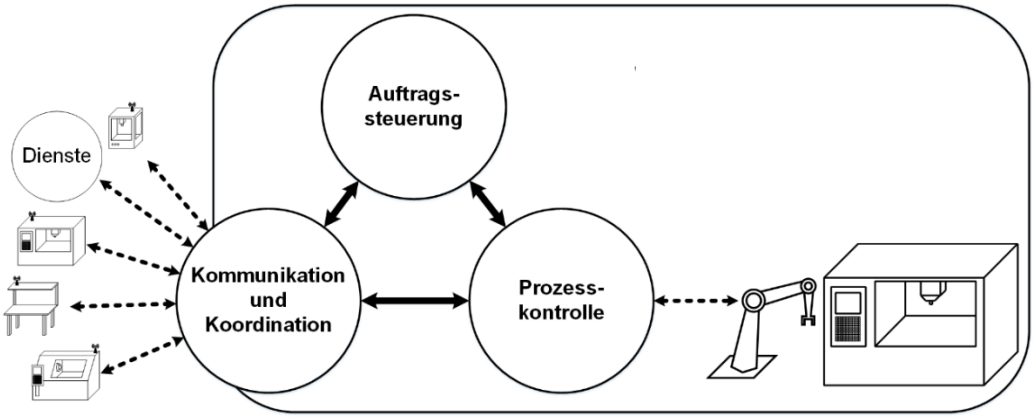

Abbildung 41: Module der automatisierten dezentralen Produktionssteuerung

Jedes Modul weist dabei seine eigene Funktion auf. Das Modul für die **Auftragssteuerung** bildet im Wesentlichen die Produktionssteuerung ab, in der die Prozesse zur Einsteuerung der Aufträge, zur Freigabe der Aufträge und zur Reihenfolgebildung abgelegt sind. Das Modul ist für jede Arbeitsstation gleich, jedoch kann die Parametrisierung unterschiedlich ausgestaltet sein. Die Auftragssteuerung ist für die Zuordnung und Planung der Aufträge verantwortlich. Zudem werden die Entscheidungen der Arbeitsstationen über das Annehmen oder Ablehnen von Aufträgen von diesem Modul getroffen.

Die **Prozesskontrolle** besitzt eine Schnittstelle zur realen Produktionsumgebung, um die Steuerung der Produktionsprozesse zu ermöglichen. Die Schnittstelle zum Produktionsprozess kann über eine Hardwareschnittstelle oder auch vollständig durch eine Softwarelösung realisiert werden. Dabei findet die Steuerung des Produktionsprozesses über Signale etc. statt. Im Rahmen der Implementierung der Produktionssteuerung im Smart Automation Labor wurde dies mit Hilfe von OPC UA und IO-Schnittstellen realisiert. Das Modul musste für die einzelnen Prozesse angepasst und für jede Arbeitsstation programmiert und parametrisiert werden. Werden in einer Arbeitsstation mehrere unterschiedliche Produktionsprozesse ausgeführt, oder besteht die Arbeitsstation aus unterschiedlichen Bestandteilen, wie z.B. einem Roboter, einem eigenen Lager und unterschiedlichen Werkzeugen, müssen diese ebenso in der Prozesskontrolle berücksichtigt werden. Zudem wurden die Daten von den Arbeitsstationen erfasst. Im Folgenden werden die wesentlichen Daten für die Prozesskontrolle aufgelistet:

- Auslastung
- Produktionsmenge
- Energieverbrauch

- Laufzeit
- Verfügbarkeit und Zuverlässigkeit
- Maschinenzustand

Die Bearbeitungszeiten werden im Rahmen der Prozesskontrolle ausgewertet und entsprechend angepasst. Die Auswertung der Bearbeitungszeiten und die Rückrechnung auf Soll-Bearbeitungszeiten erfolgt dabei produktspezifisch, wobei standardmäßig der Median der letzten Prozessausführungen verwendet wird. Hieraus können genaue Vorhersagen über die Bearbeitungszeiten zukünftiger Prozesse abgeleitet werden. Prozesse, die zusätzlich von Menschen gesteuert werden, müssen darüber hinaus eine entsprechende Interaktion mit der Benutzerin oder dem Benutzer implementiert haben. Alle notwendigen Informationen von der Steuerung werden auf einer Mensch-Maschine-Schnittstelle dargestellt und gleichzeitig werden die folgenden Prozessdaten zurückgemeldet:

- Anwesenheit
- Auftragsstatus
- Fertigmeldung, Fortschrittsmeldung, Gutstückmeldung
- Störung, Störungsbestätigung, Unterbrechung
- Material

Das **Kommunikations- und Koordinationsmodul** dient zur Kommunikation zwischen den Arbeitsstationen und übergeordneter Software. In dem Smart Automation Labor ist übergeordnete Software im Wesentlichen der Produktionsplan und der Auftragsspeicher. Der Auftragsspeicher ist eine Datenbank, in der die zu bearbeitenden Aufträge abgelegt sind. Der Produktionsplan beinhaltet alle Informationen der zu produzierenden Aufträge. Die wesentlichen Daten von den Aufträgen werden im Folgenden dargestellt:

- Produkt (Artikelnummer, Produktstruktur, Stückliste)
- Prozessdaten (unter anderem Freigabezeitpunkt, Plan-Starttermin, spätester Starttermin, Plan-Endtermin, spätester Fertigstellungszeitpunkt)
- Vorhergehender Prozess (Arbeitsstation, Bearbeitungszeiten)
- Nachfolgender Prozess (Arbeitsstation, Bearbeitungszeiten)
- Auftragsbezogene Informationen (Kunde, Menge, Auftragsvolumen, Priorität)

Dabei werden alle Aufträge von der Produktionsplanung in eine Auftragsliste eingetragen und auf die entsprechende Arbeitsstation übertragen. Zudem gibt es weitere Aufträge, die nicht einer Arbeitsstation zugewiesen sind. Solche Aufträge bleiben zuerst im Auftragsspeicher, bis die Arbeitsstationen sich die Aufträge eigenständig zuweisen.

Die Kommunikation zwischen den Arbeitsstationen wurde als **P2P-Netzwerk** gestaltet. Jede Arbeitsstation meldet sich mit ihren Informationen über einen Rundruf bei allen anderen Arbeitsstationen an und wird von den anderen Arbeitsstationen in einem Adressbuch gespeichert und mit entsprechenden Eigenschaften versehen. Durch die Anmeldung und das Adressbuch wissen alle Arbeitsstationen jederzeit, welche Arbeitsstationen gerade verfügbar sind. Die Informationen über die Arbeitsstation enthalten unter anderem den Typ und die Eigenschaften der Arbeitsstation sowie die Komponenten bzw. die Prozesse, die sie bearbeiten kann. Dadurch kennt jede Arbeitsstation alle angemeldeten Arbeitsstationen und kann jederzeit deren Status nachvollziehen. Falls eine Arbeitsstation ausfällt, können die Aufträge automatisch auf die anderen Arbeitsstationen verteilt werden. Generelle Änderungen am Produktionsplan, wie zum Beispiel die Übernahme eines Auftrags von einer anderen Arbeitsstation, werden allen Produktionselementen per Rundruf (UDP) mitgeteilt. Alle anderen Nachrichten bzw. Informationen sind unmittelbar an die Arbeitsstationen gerichtet (über TCP) und finden nur zwischen den Arbeitsstationen statt. Dieser Aufbau der Kommunikation im Labor ist in der nachfolgenden Abbildung 42 ersichtlich.

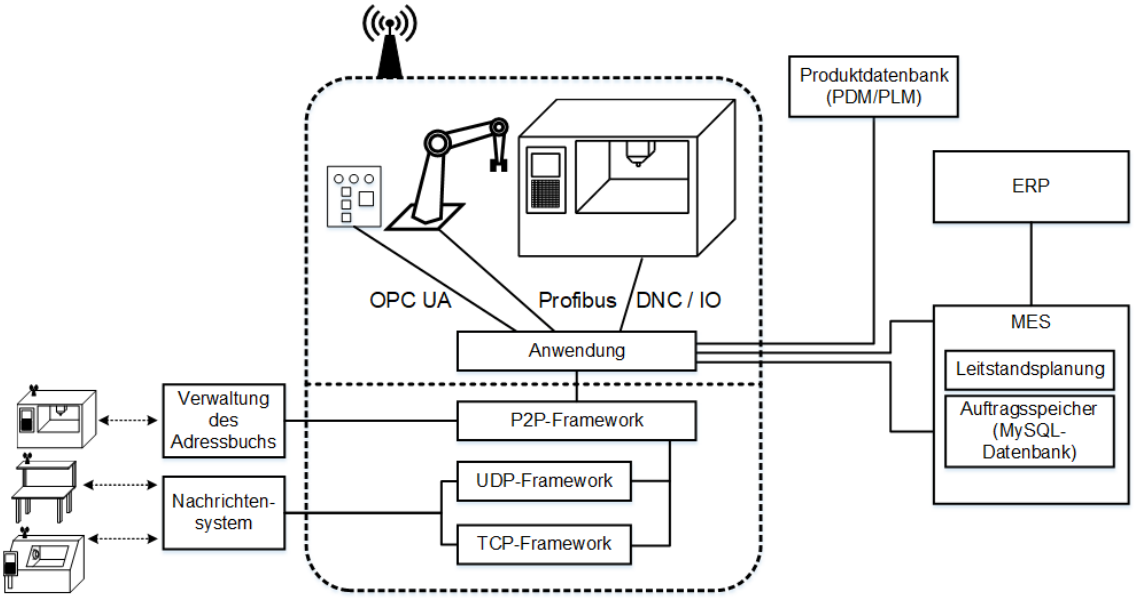

Abbildung 42: Kommunikationsschnittstellen zwischen Arbeitsstationen und übergeordneten Diensten

Zusätzlich zu der Implementierung des dezentralen Steuerungssystems im Labor existieren mehrere **Dienste**, die durch zusätzliche Computersysteme realisiert wurden. Ein Dienst dient hierbei z.B. zur Erzeugung der Produktionsaufträge aus den Kundenaufträgen. Die Datenbank des Auftragsspeichers befindet sich ebenso in diesem Computersystem. Der Auftragsspeicher wurde im Labor als MySQL-Datenbank umgesetzt. Die Kommunikation zwischen den Produktionselementen erfolgt über Abfragen dieser Datenbank.

6.3 Implementierung der digitalen Repräsentation

Dieser Abschnitt beschreibt die prototypische Umsetzung der digitalen Repräsentation in dem Smart Automation Labor. Die Intention besteht darin, den Ansatz der digitalen Repräsentation zu einem funktionsfähigen Werkzeug umzusetzen. Um die Funktionalitäten der dezentralen Produktionssteuerung und der digitalen Repräsentation der Beschäftigten zu überprüfen, müssen sie separat validiert werden. Da die Validierung der digitalen Repräsentation losgelöst von der dezentralen Produktionssteuerung durchgeführt werden kann, wird sie in diesem Abschnitt beschrieben.

Für die Implementierung der digitalen Repräsentation wurde sich auf Raspberry Pis konzentriert, die auch im Labor zur Steuerung einzelner Arbeitsstationen verwendet wurden. Die Raspberry Pis dienen dabei der Interaktion mit der Benutzerin oder dem Benutzer über eine GUI. Die Raspberry Pis sind Computersysteme und übernehmen die Kommunikation mit anderen Produktionselementen und die Durchführung der Prozesse der digitalen Repräsentation. Hierzu wurden die Raspberry Pis mit einem 7-Zoll-Touchscreen ausgestattet, sodass eine Interaktion mit Benutzern ermöglicht wird. Als Programmiersprache wurde, wie bereits bei der Produktionssteuerung, Python ausgewählt. Zum einen können zahlreiche Module und Klassen übernommen werden, weil Python für die Programmierung der entwickelten dezentralen Produktionssteuerung verwendet wurde. Zum anderen ermöglicht die Python-Programmiersprache eine leichte Übertragbarkeit auf mobile Endgeräte.

Die Grundstruktur des Informationsaustausches ist dabei überschaubar: die digitale Repräsentation kommuniziert mit anderen Produktionselementen über das Netzwerk und der Benutzerin bzw. dem Benutzer über die GUI. Diese Grundstruktur wurde als Grundlage der Implementierung der digitalen Repräsentation verwendet. Die Daten der digitalen Repräsentation werden in einer MySQL-Datenbank gespeichert. Dies sind alle relevanten historischen Daten, die Eigenschaften der Beschäftigten, also welche Fähigkeiten und Qualifikationen vorhanden sind, die Präferenzen und zudem die Aufgaben und die Anwesenheitsplanung. Alle weiteren Daten werden im Log gespeichert. Im Folgenden wird der Ablauf des Informationsaustausches in der MySQL-Datenbank anhand des Zuweisungsprozesses der digitalen Repräsentation vorgestellt (siehe Abbildung 43).

In der Datenbank befinden sich die historischen Daten von Beschäftigten, die im Wesentlichen aus Prozessausführungen mit entsprechenden Ausführungszeiten und deren zugeordneten Aufträgen und Arbeitsstationen bestehen. Darauf basierend können die Prozessausführungszeiten der Beschäftigten prognostiziert werden. Darüber hinaus enthält die Datenbank die Anwesenheitsplanung bzw. die geplante Anwesenheitszeiten der Beschäftigten. Die Fähigkeiten und Qualifikationen von Beschäftigten, die als Fähigkeitsprofil tabellarisch gespeichert sind (siehe Abschnitt 5.2), werden aus einem Zusatzprogramm aus historischen Daten und den ursprünglichen Einträgen von Beschäftigten gespeist. Durch die Favorisierungen werden die Wünsche und Bedürfnisse

der Beschäftigten berücksichtigt. Dabei werden die von der Mitarbeiterin, bzw. dem Mitarbeiter unbeliebten Tätigkeiten und Arbeitsstationen in der Datenbank erfasst. Zudem befinden sich in der Datenbank die arbeitsstationsbezogenen Informationen, wie z.B. die Daten von den zu bearbeitenden Aufgaben und Anlagen.

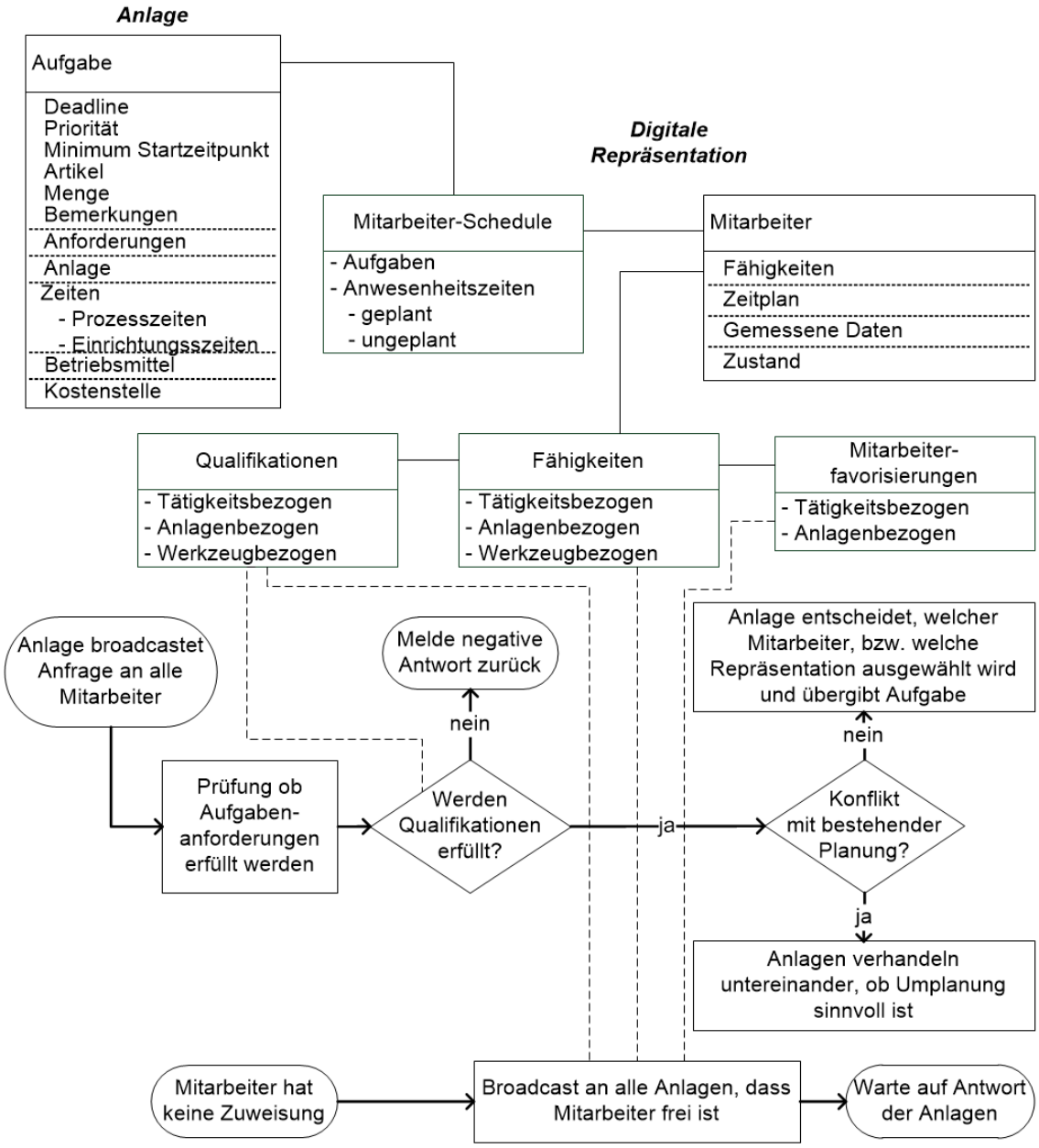

Abbildung 43: Informationsaustausch des Zuweisungsprozesses der digitalen Repräsentation in der MySQL-Datenbank

Die Implementierung des Zuweisungsprozesses der digitalen Repräsentation beginnt mit einer Anfrage an alle Beschäftigten: welche Mitarbeiterin, bzw. welcher Mitarbeiter kann an dieser Arbeitsstation den Auftrag ausführen. Nach der Anfrage wird geprüft, ob die Aufgabe von den Beschäftigten ausgeführt werden kann. Dafür müssen die Qualifikationen, die Fähigkeiten und die Favorisierungen sowie die Verfügbarkeit der

Mitarbeiterin bzw. des Mitarbeiters gleichzeitig berücksichtigt werden. Falls Konflikte mit der Verfügbarkeit bestehen, findet eine Verhandlung bezüglich einer Umplanung statt. Eine ausführliche Beschreibung des Zuweisungsprozesses ist in Abschnitt 5.3 dargestellt. Daraus ergibt sich das Konzept zur Implementierung der Ablaufsteuerung der digitalen Repräsentation. Dabei wird auf die drei wesentlichen Funktionsmodule der digitalen Repräsentation der Beschäftigten eingegangen: die interne Ablaufsteuerung, die GUI und die Kommunikation mit anderen Produktionselementen. Zudem wird hierbei die Implementierung der Datensicherheit erläutert.

Die Aufgabe der internen **Ablaufsteuerung** ist es, alle Prozesse der digitalen Repräsentation auszuführen. Wie bereits vorgestellt, ist der wesentliche Prozess der digitalen Repräsentation die Zuweisung der Beschäftigten zu Arbeitsstationen. Die interne Ablaufsteuerung beginnt mit einer Anmeldung der Mitarbeiterin bzw. des Mitarbeiters über die GUI, indem der Benutzername und das Passwort eingegeben werden. Anschließend werden die Daten von der persönlichen Datenbank abgerufen. Nach der Anmeldung erfolgt die Abfrage auf der GUI nach dem aktuellen Status: „Bereit“, „Pause“ oder „Stopp“. Die GUI ist kontextbasiert aufgebaut. Mit der Einstellung „Bereit“ wird der Prozess zur Arbeitsaufnahme gestartet und anschließend wird eine Arbeitsstation zugewiesen. Auf der GUI wird der aktuelle Status angezeigt, also die Zuweisung zur Arbeitsstation, die Zuweisung zum Auftrag und die Anweisung der Arbeitsschritte. Gleichzeitig wird auf der GUI die nächste Aufgabe angezeigt. Die Abwesenheit wird so gemessen, dass wenn auf der GUI 3 Minuten lang nichts gedrückt oder quittiert wurde und gleichzeitig keine Aktion z.B. an der Montagestation ausgeführt wird, wird dies als Pause wahrgenommen und als Abwesenheit erfasst. Sobald ein Ereignis erreicht wird, wie z.B. die Mittagspause, wird dies auf der GUI angezeigt. Dabei wird ein Kontextmenü aufgeblendet. Soll die Pause verschoben werden, existiert hierfür ein entsprechendes Bedienfeld.

Während des Zuweisungsprozesses werden die Daten über die Beschäftigten an die Arbeitsstation übertragen. Die Arbeitsstation übermittelt die Daten der Aufgabe bzw. der Aufträge zur digitalen Repräsentation. Dadurch findet die **Kommunikation** zwischen Arbeitsstationen und digitalen Repräsentationen statt. Diese Kommunikation findet über das P2P-Netzwerk mithilfe des Austausches von JSON-Daten statt. Die erwarteten Prozesszeiten werden einfach über historische Ausführungen des gleichen Prozesses an der gleichen Arbeitsstation gebildet, indem der Median der letzten 50 Ausführungen berechnet wird. Existieren weniger Daten, wird entsprechend nur der Median der vorhandenen Daten gewählt. Liegen keine Daten vor, wird die Ausführung des gleichen Prozesses an einer anderen Arbeitsstation herangezogen. Existieren auch hierfür keine Daten, wird der Median des Arbeitsschrittes, der nach den Daten von anderen Beschäftigten berechnet wird, verwendet. Falls diese Daten auch nicht vorhanden sind, werden die Vorgabezeiten herangezogen. Dieser Fall ist selten, da diese Aufgabe einer komplett neuen Ausführung für die Beschäftigten entspricht. Anschließend wird der Prozess ausgeführt. Während der Prozessausführung werden alle Ereignisse über einen

„Eventhandler“ dokumentiert. Insbesondere wird an der Montagestation jeder Schritt, der ausgeführt wurde, dokumentiert und per JSON-Austausch der digitalen Repräsentation mitgeteilt.

Um **Datenschutz** zu gewährleisten, werden die persönlichen Daten der Beschäftigten an Arbeitsstationen nach der Abmeldung gelöscht. Das heißt, dass die Zuweisungsdaten von Beschäftigten aus der Datenbank der Arbeitsstationen entfernt werden. Dadurch werden die personenbezogenen Daten ausschließlich auf der digitalen Repräsentation gespeichert.

Validierung der digitalen Repräsentation der Beschäftigten

Bisher wurde die Implementierung der digitalen Repräsentation vorgestellt. Im Folgenden wird auf ihre Validierung eingegangen. Bei der Produktion des Fernsteuerungsautos im Smart Automation Labor gibt es nur eine Tätigkeit, die von Beschäftigten ausgeführt werden muss: Die Bedienung der Montagestation. Für ein aussagekräftiges Validierungsergebnis wurden zu diesem Zweck zwei weitere Montagestationen virtuell hinzugefügt. An diesen Montagestationen werden Fernsteuerungsautos mit bereits vorproduzierten Teilen zusammengefügt. Im Rahmen der Testdurchläufe wurde die digitale Repräsentation der Beschäftigten anhand von drei Szenarien validiert: dem normalen Betrieb, dem Ausfall von Beschäftigten und dem Ausfall eines Logistikelements.

Der **normale Betrieb** beinhaltet die Montage des Fernsteuerungsautos. Im normalen Betrieb sind die Zuweisungen zu den Arbeitsstationen bereits erfolgt. Das heißt, dass mit der Anmeldung schon eine Zuweisung zu einer Arbeitsstation für die Mitarbeiterin oder den Mitarbeiter vorgegeben wird. Anschließend findet die Montage an den Arbeitsstationen statt. Im Normalbetrieb werden keine Änderungen vorgenommen. Dabei wurden die Grundfunktionalitäten der digitalen Repräsentation erfolgreich validiert. In diesem Fall wurden keine Bearbeitungszeiten von der digitalen Repräsentation der Beschäftigten prognostiziert. Da der Auftrag zum ersten Mal von dem eingesetzten Personal durchgeführt wurde, lagen nicht ausreichend historische Daten für eine Prognose vor. Der Hauptgrund hierfür ist, dass die Validierungsdurchläufe maximal 2 Stunden entsprachen und unterschiedliche Personen für den Test eingesetzt wurden. Im normalen Betrieb konnte keine Veränderungen in der Produktionssteuerung zwischen der Steuerung mit digitaler Repräsentation und der Steuerung ohne digitale Repräsentation festgestellt werden. Tritt kein Ereignis auf oder werden keine Randbedingungen verändert, findet die Planung und Steuerung somit unverändert statt. Dadurch entsprachen die Auftragsreihenfolgeplänen auf den Arbeitsstationen mit digitaler Repräsentation denen ohne digitale Repräsentation. Erst wenn ein Ereignis auftritt oder die Planung nicht mehr mit der Realität übereinstimmt, wird eine Veränderung eingeleitet.

Um dies auszutesten, wurde das zweite Szenario „**Ausfall eines Beschäftigten**“ entwickelt. In diesem Szenario melden sich eine Mitarbeiterin oder ein Mitarbeiter im Laufe des Betriebs ab. Ohne die digitale Repräsentation würde die Arbeitsstation

ausgeschaltet werden und wäre so lange ausgefallen, bis ihr manuell eine neue Mitarbeiterin bzw. ein neuer Mitarbeiter zugewiesen würde. Durch die digitale Repräsentation fragt nun die Montagestation andere digitalen Repräsentationen an, also in diesem Fall die anderen beiden virtuellen Montagestationen. Wenn dabei keine Aufträge kritisch werden, wird eine Umplanung der Produktion bzw. der Montage eingeleitet und ein Mitarbeiter aus einer virtuellen Montagestation zu dieser Montagestation zugewiesen. Dieses Szenario wurde in insgesamt 7 Validierungsdurchläufen durchgeführt. In 4 Validierungsdurchläufen konnte erfolgreich eine Umplanung durchgeführt werden. In 3 Validierungsdurchläufen hiervon wurde eine Neuzuweisung von einer virtuellen Montagestation vorgenommen. Einmal wurden die Aufträge auf die virtuelle Montagestation verschoben. In einem Validierungsdurchlauf mussten keine Veränderungen eingeleitet werden, da die Aufträge während des Validierungsdurchlaufs nicht kritisch wurden. Bei den zwei verbleibenden Validierungsdurchläufen konnte keine Umplanung durchgeführt werden, da gleichzeitig bei mehreren Aufträgen die Liefertreue gefährdet war und deswegen keine neue Zuweisung von Beschäftigten oder eine Verschiebung der Aufträge möglich war. Die vier erfolgreichen Validierungsdurchläufe haben jedoch gezeigt, dass durch den Einsatz der digitalen Repräsentation der Beschäftigten eine automatisierte Umplanung mit einer Neuzuweisung der Beschäftigten realisiert wird. Durch dieses Validierungsszenario konnte gezeigt werden, dass die digitale Repräsentation Störungen während des Betriebs automatisiert beseitigen kann.

Das dritte Szenario bildet den „**Ausfall eines Logistikelements**“. Dabei wurde zusätzlich ein Hilferuf in den Programmcode des Logistikelements implementiert. Der Hilferuf schickt eine Meldung an alle digitalen Repräsentationen, ob eine geeignete Person zu Hilfe kommen und dadurch eine Störung beseitigen kann. Dies geschieht durch eine Mitteilung an alle digitalen Repräsentationen. Der Ausfall eines Logistikelements wird in dem Smart Automation Labor durch eine Routine der Shuttles ausgelöst. Die Routine wird im Folgenden beschrieben. Die Shuttles fahren ständig im Kreis und warten dabei auf Aufträge von den Produktionsanlagen. Diese Aufträge sind Anfragen, dass Teile abgeholt oder abgeliefert werden sollen. Die Shuttles fahren dabei von Weiche zu Weiche. Das Shuttle weiß dabei, welche Weiche als nächstes angefahren wird tauscht sich über seine Route vor dem Erreichen des Haltepunkts mit der Weiche aus. Für das Shuttle wird ein Ausfall dadurch erkannt, dass das Shuttle sich vor Erreichen einer Weiche an dieser meldet. Sobald nach dieser Meldung über zwei Minuten ohne weitere Meldung vergangen sind, wird darauf geschlossen, dass es zu einer Störung gekommen ist. Der Kontrollpunkt erzeugt daraufhin eine Meldung an alle digitalen Repräsentationen, um diese Störung zu beseitigen. Daran schließt sich eine Verhandlung zwischen den digitalen Repräsentationen an, in deren Rahmen bei den Arbeitsstationen angefragt wird, ob eine kurze Unterbrechung ihrer Tätigkeit möglich ist. In der Folge wird dann eine Mitarbeiterin oder ein Mitarbeiter ausgewählt, bei deren Auswahl alle Fertigstellungszeitpunkte der Aufträge weiterhin eingehalten werden können. Sollte hierbei niemand gefunden werden, wird diejenige oder derjenige mit den geringsten

Auswirkungen auf den Produktionsplan ausgewählt. Die Mitarbeiterin oder der Mitarbeiter erhält anschließend eine entsprechende Meldung auf der GUI, welche angenommen oder abgelehnt werden kann. Dabei wurde der Ausfall eines Shuttles mehrfach getestet und jedes Mal konnte eine Mitarbeiterin, bzw. ein Mitarbeiter erreicht und eingeplant werden. Dadurch wurde gezeigt, dass durch die digitale Repräsentation unnötige Umplanung vermieden werden und Störungen automatisch beseitigt werden können.

In diesem Kapitel wurde die in dieser Arbeit entwickelte automatisierte dezentrale Produktionssteuerung und digitale Repräsentation implementiert. Dafür wurde zuerst das Smart Automation Labor zu einem CPPS mitsamt dezentraler Produktionsstruktur umgebaut. Dabei wurden an den einzelnen Arbeitsstationen, dem Logistiksystem und der Lagerstation Computersysteme eingebaut, auf denen sich die Software des automatisierten dezentralen Steuerungssystems befindet. Darauf basierend wurde die dezentrale Produktionssteuerung implementiert. Dabei wurde auf die Implementierung der drei wesentlichen Funktionsmodule eingegangen: die Auftragssteuerung, die Prozesskontrolle sowie die Kommunikation und Koordination. Anschließend wurde die digitale Repräsentation der Beschäftigten implementiert. Zum Schluss des Kapitels wurde die digitale Repräsentation mit Hilfe von drei Testszenarien validiert.

7 Validierung

In diesem Kapitel wird die Validierung der automatisierten dezentralen Produktionssteuerung vorgestellt. Angelehnt an die zugrundeliegende Methodik der DSRM (siehe Abschnitt 1.4) findet die Validierung durch die Feststellung der Effizienz einer Anwendung statt. Die Anwendung wird nachfolgend in Abschnitt 7.1 beschrieben. Die Effizienz wird dabei durch die logistischen Zielgrößen, also eine hohe Liefertreue, kurze Durchlaufzeiten, geringe Bestände und eine hohe Auslastung, widergespiegelt (siehe Abschnitt 2.2). Daraus ergeben sich die Zielgrößen der Validierung des dezentralen Steuerungssystems. Die Validierung der dezentralen Produktionssteuerungen wird in zwei Stufen durchgeführt:

- Die erste Stufe der Validierung findet im Smart Automation Labor des Lehrstuhls für Produktentstehung am Heinz Nixdorf Institut statt (siehe Abschnitt 7.2). Die Anwendung wird im Labor jeweils von der zuvor vorhandenen Steuerung (siehe Abschnitt 6.1) und der dezentralen Steuerung durchgeführt und die Ergebnisse untereinander verglichen. Da die zentrale Steuerungslösung heutzutage in den meisten Unternehmen eingesetzt wird, werden diese Untersuchungsergebnisse als Referenz für die in dieser Arbeit entwickelten Lösung angesehen.
- Die zweite Stufe der Validierung wird mit Hilfe einer Simulation in dem Programm Plant Simulation durchgeführt (siehe Abschnitt 7.3). Hierbei wird dieselbe Bearbeitungsaufgabe wie bei der ersten Stufen der Validierung zugrunde gelegt. Für diese Validierung wird zuerst das gesamte Produktionssystem in dem Programm Plant Simulation modelliert. Anschließend wird die Bearbeitungsaufgabe in dem Programm abgebildet und dann mit verschiedenen Steuerungsverfahren ausgeführt, die mit der automatisierten dezentralen Produktionssteuerung verglichen werden.

Wesentliches Ziel der Validierung ist die Funktionsfähigkeit des in dieser Arbeit entwickelten Steuerungssystems nachzuweisen. Hierzu muss insbesondere die Fähigkeit zur Selbstorganisation validiert werden, die den wesentlichen Neuheitsgrad der automatisierten dezentralen Produktionssteuerung darstellt (siehe Abschnitt 3.3). Die Selbsorganisation bedeutet, dass Aufträge eigenständig durch die Software auf die Arbeitsstationen verteilt und selbst bei Störungen oder geänderten Randbedingungen optimal ausgeführt werden.

7.1 Beschreibung des Anwendungsszenarios

Als Demonstrator für Produktionsanwendungen im Smart Automation Labor wurde ein Fernsteuerungsauto entwickelt (siehe Abbildung 44). Für die Auswahl des Demonstrators und des Anwendungsszenarios wurden die folgenden Anforderungen berücksichtigt:

- Der Demonstrator muss in dem Smart Automation Labor produziert werden können.

- Der Produktionsprozess des Demonstrators soll alle Produktionsverfahren, die im Smart Automation Labor angewendet werden können, nutzen.
- Der Demonstrator soll verschiedene Varianten haben bzw. individualisierbar sein.
- Die Produktionskosten des Demonstrators dürfen nicht mehr als 40 € betragen.

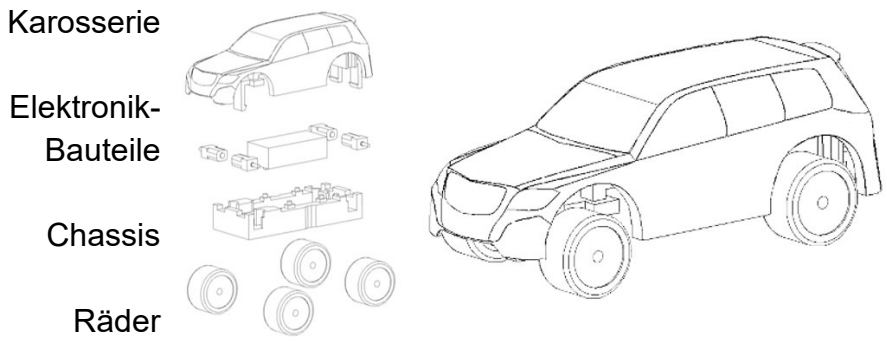

Abbildung 44: Demonstrator Fernsteuerungsauto

Das Fernsteuerungsauto besteht aus drei Eigenfertigungskomponenten und mehreren Zukaufteilen, die zusammen montiert werden. Dafür gibt es vier Arbeitsstationen, die das Fernsteuerungsauto durchlaufen muss. Der Produktionsprozess ist dabei zweistufig aufgebaut. Zuerst findet die Fertigung statt und sobald alle Teile gefertigt sind, erfolgt in der Endmontage gemeinsam mit den Zukaufteilen die Erstellung des Fernsteuerungsautos. Die Karosserie wird an einer 3D-Drucker-Station gedruckt, das Chassis an einer Frässtation gefräst und die Räder werden an einer Drehstation gedreht. Für alle Fertigungsprozesse werden entsprechende Rohmaterialien benötigt. Die Arbeitsstationen sind hierbei über das Logistiksystem mit der Lagerstation und der Montagestation verbunden. In Abbildung 45 werden der Aufbau des Fernsteuerungsautos und der Produktionsprozess dargestellt.

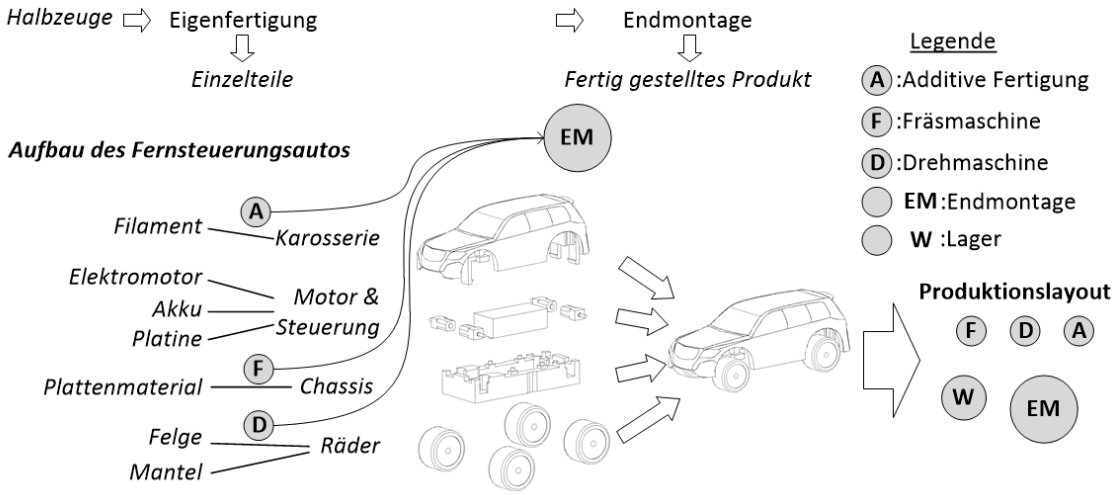

Abbildung 45: Aufbau und Produktionsprozess des Fernsteuerungsautos

Anwendungsszenario

Das Fernsteuerungsauto ist vom Kunden individuell konfigurierbar, der zudem einen Wunschtermin für die Fertigstellung festlegen kann. Auf dieser Grundlage wird der Kundenauftrag erstellt. Die benötigten Materialien werden auf Basis der Stückliste für den Auftrag eingeplant und bei entsprechenden Unterdeckungen neu beschafft. Anschließend werden die einzelnen Produktionsschritte (siehe Abbildung 46 oben rechts) eingeplant. Hierfür werden im ersten Schritt die Produktionsaufträge (A_F1, A_D1, A_A1, A_M1) für die vier Produktionsprozesse angelegt. Die Anlieferungstermine für Materialien und Zukaufteile müssen dabei berücksichtigt werden. Anschließend wird der letzte Produktionsschritt (die Endmontage) vor dem Fertigstellungstermin eingeplant. Der Starttermin der Endmontage bildet den spätesten Fertigstellungszeitpunkt für die Fertigungsaufträge. Darauf basierend werden anschließend die Fertigungsaufträge der drei Eigenfertigungskomponenten auf freien Kapazitäten der dazugehörigen Arbeitsstationen eingeplant. Diese rückwärtsterminierte Auftragseinplanung wird in der Abbildung 46 veranschaulicht. Der Kundenauftrag für das Fernsteuerungsauto besteht aus den folgenden Produktionsaufträgen:

- Der Auftrag an der Frässtation: Fräsen des Chassis (A_F1)
- Der Auftrag an der 3D-Drucker-Station: Drucken der Karosserie (A_A1)
- Der Auftrag an der Drehstation: Drehen der Räder (A_D1)
- Der Auftrag an der Montagestation: Montage (A_M1)

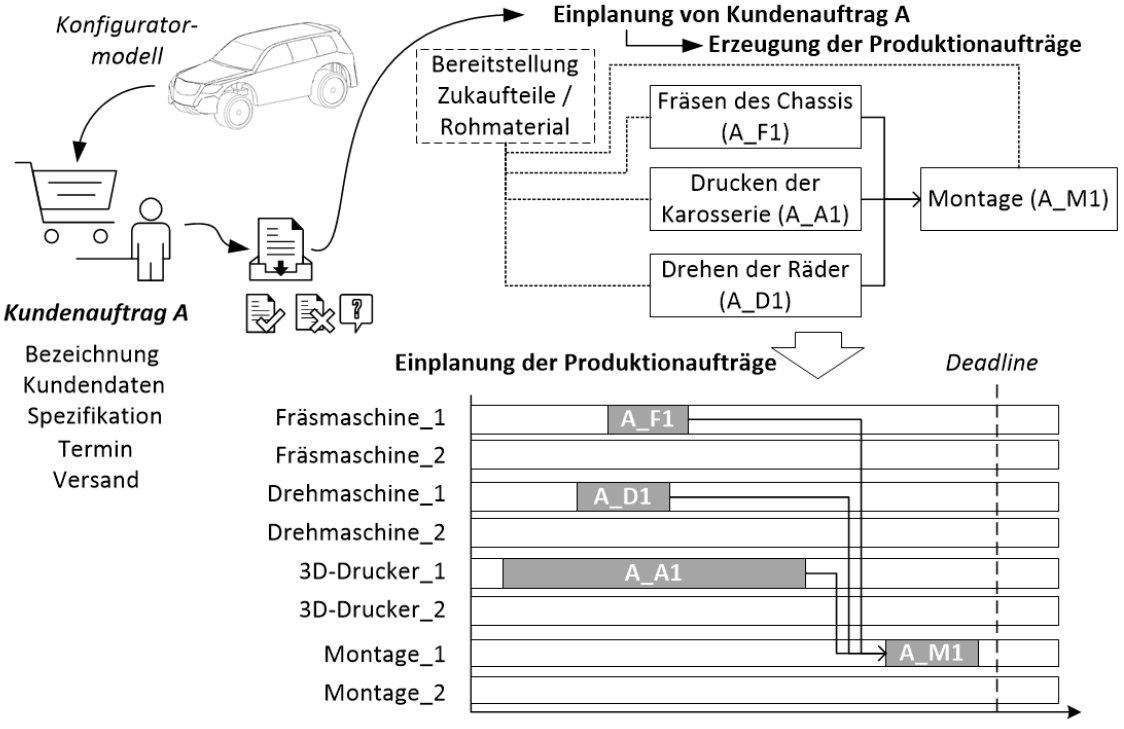

Abbildung 46: Einplanung eines Kundenauftrags für das Fernsteuerungsauto

Nach der Erzeugung der Produktionsaufträge wird der Produktionsprozess durchgeführt. Zuerst finden die Fertigungsprozesse statt. Anschließend werden die Komponenten von den Shuttles eingesammelt und zur Montagestation transportiert. Für den Fräs- und Drehprozess schwanken die Ausführungszeiten geringfügig. Der Fertigungsprozess der Frässtation dauert 6:07 Minuten und der Fertigungsprozess der Drehstation 7:02 Minuten. Der 3D-Druck-Prozess wird dabei nicht ausgeführt, da dieser Prozess über 16 Stunden in Anspruch nimmt. Für die Validierung wird der Druckprozess nur simuliert und eine virtuelle Fertigungsdauer von 5:30 Minuten definiert. Dadurch kann dieser Druckprozess sinnvoll in der Simulation genutzt werden. In realen Produktionssystemen werden zudem selten 3D-Druck-Prozesse eingesetzt und in der Regel sind die Prozesszeiten untereinander angeglichen. Deswegen bildet diese Simulation auch ein realistischeres Produktionsszenario ab. Die Karosserien für die Endmontage werden direkt aus einem Lager an der 3D-Druckstation entnommen. Die Montagezeiten sind stark von den eingesetzten Beschäftigten abhängig und schwanken zudem sehr stark. Im Durchschnitt dauert die Montage eines Fernsteuerungsautos bei erfahrenden Beschäftigten 5:36 Minuten. Der Transport zwischen den Arbeitsstationen ist abhängig von der Entfernung der Arbeitsstationen zueinander. Durchschnittlich beträgt eine Shuttlefahrt von Arbeitsstation zu Arbeitsstation 27 Sekunden. Die Shuttles sind jedoch deutlich länger blockiert, da sie erst zur Arbeitsstation fahren und dort auf die Beladung warten müssen.

Darüber hinaus muss in diesem Anwendungsszenario die Umplanung möglich sein, um die Selbstorganisation der dezentralen Steuerung zu validieren. Das heißt, dass bei Störungen automatisiert Umplanungen von der dezentralen Produktionssteuerung eingeleitet werden müssen. Da im Labor jeweils nur eine Arbeitsstation für einen Produktionsprozess zur Verfügung steht, wurden deswegen baugleiche Arbeitsstationen simuliert. Die simulierten Arbeitsstationen sind in Abbildung 46 dargestellt: Frässtation_2, Drehstation_2, 3D-Drucker-Station_2 und Montagestation_2. In den simulierten Arbeitsstationen wurden die Steuerungen der realen Maschinen eingesetzt. Durch die simulierten Arbeitsstationen existieren parallele Arbeitsstationen für jeden Produktionsschritt, die eine Auftragsumplanung systemseitig ermöglichen.

7.2 Validierung durch reale Umsetzung im Smart Automation Labor

Nach der Vorstellung des Demonstrators und des Anwendungsszenarios folgt die Umsetzung der Validierung. Dafür müssen die folgenden Zielgrößen, Annahmen und Randbedingungen berücksichtigt werden:

- Für die Bewertung der Ergebnisse von den verschiedenen Steuerungsverfahren werden die folgenden Zielgrößen ausgehend von den logistischen Zielgrößen (siehe Abschnitt 2.2) abgeleitet: die Termintreue und der Umlaufbestand. Die Termintreue soll möglichst hoch sein. Der Umlaufbestand sollen auf einem geringen Niveau geregelt werden. Ein geringer Umlaufbestand dient dazu, dass das Produktionssystem einerseits flexibel auf kurzfristige Kundenwünsche reagieren kann und andererseits

das in der Produktion gebundene Kapital minimiert werden kann. Die Durchlaufzeiten und die Auslastung werden direkt durch die Umlaufbestände beeinflusst, weswegen diese nicht explizit betrachtet werden müssen.

- Bei der Durchführung des Anwendungsszenarios wird eine Auftragsliste von der Produktionsplanung bereitgestellt. Während der Auftragsausführung können neue Aufträge für das Produktionssystem generiert werden.
- Es werden verschiedene Auftragstypen bearbeitet: Aufträge mit oder ohne Wunschtermine, Eilaufträge oder Aufträge mit festgelegten Terminfristen.
- Der Materialfluss wird über das Logistiksystem (siehe Abschnitt 6.1) sichergestellt. In einem Shuttle wird jedes Mal nur ein Teil transportiert. Im Rahmen der Validierung wird hierfür eine durchschnittliche Fahrtzeit von Arbeitsstation zu Arbeitsstation von 30 Sekunden angenommen.
- Es wird eine Verfügbarkeit der Arbeitsstationen von 95% angenommen. Bei der Modellierung von Störungen wird die Verfügbarkeit der Arbeitsstationen auf 80% reduziert.
- Die Produktion wird ohne Auftragsvorrat gestartet.
- Für jede Arbeitsstation wird ein Pufferlager mit 4 Einheiten eingerichtet.
- Der Simulationszeitraum für jeden Durchlauf beträgt 4 Stunden. Somit kann der Simulationszeitraum als halbe Schicht eines realen Produktionssystems angesehen werden.
- Rüst- und Einrichtungszeiten werden nicht betrachtet.
- Alle Aufträge haben die gleichen Prozesszeiten.
- Abhängigkeiten zwischen Aufträgen, zum Beispiel durch Rüstmatrizen werden nicht berücksichtigt.
- Es wird angenommen, dass die benötigten Rohmaterialien und Zukaufteile stets in ausreichender Menge vorhanden sind.
- Begonnene Aufträge werden nicht unterbrochen.
- Alle Aufträge haben die Losgröße 1. Daher wird das Teilen von Aufträgen oder überlappendes Arbeiten nicht betrachtet.

Aufbau des Untersuchungsmodells

Die Validierung findet zwar in der realen Umgebung des Labors statt, bei der Ausführung wurden dabei aber keine Komponenten von den Arbeitsstationen produziert, sondern mithilfe von Timern die einzelnen Produktionsschritte an den Arbeitsstationen simuliert. Dadurch kann gewährleistet werden, dass Prozessausführungen ohne Schwankungen ausgeführt werden. In der Realität haben die Produktionsmaschinen Schwankungen von

bis zu 30 Sekunden und in der Montage sind Schwankungen von bis zu einer Minute und gleichzeitig mehrfach Unterbrechungen zu beobachten. Zudem wird für alle Untersuchungen in der Validierung das gleiche initiale Auftragsset angenommen. Hierbei handelt es sich um 50 Aufträge ohne Wunschtermin.

Zuerst wird auf den Aufbau des Untersuchungsmodells des zentralen Steuerungssystems mittels SPS eingegangen. Ein solches zentrales Steuerungssystem wird heutzutage sehr häufig in der Industrie eingesetzt, weswegen es für die Validierung als Referenz genommen wird. Die zentrale SPS wurde ursprünglich als Steuerungslösung im Labor des Lehrstuhls für Produktentstehung am Heinz Nixdorf Institut eingesetzt. Bei dem Umbau des Labors auf ein dezentrales CPPS wurde die komplette Steuerung jedoch abgebaut. Daher ist der direkte Einsatz der SPS für diese Untersuchung nicht mehr möglich. Aus diesem Grund wurde die Funktionalität der SPS nachprogrammiert und in die neu eingesetzten Computersysteme implementiert. Dadurch kann der Durchlauf des Anwendungsszenarios in dem vorherigen, zentralen Steuerungssystem simuliert werden.

In dem ersten Schritt findet hierbei die Auftragsverteilung statt, die bereits von der Produktionsplanung festgelegt wurde. Neue Aufträge werden in der ankommenden Reihenfolge nach dem FIFO-Prinzip (siehe Abschnitt 3.1.2) so lange im Auftragsspeicher gelassen, bis freie Kapazitäten verfügbar sind. Anschließend wird der nächste Auftrag einfach an das Ende der Auftragswarteschlange platziert und abgearbeitet. Sobald alle Aufträge abgearbeitet sind, wird anschließend der nächste Auftrag aus dem Auftragsspeicher bearbeitet (siehe Abschnitt 4.2.1). Es findet somit nur initial eine Reihenfolgeplanung statt. Für Eilaufträge wurden zusätzlich Regeln eingebaut: Eilaufträge können direkt auf die nächste Position in der Auftragswarteschlange platziert werden. Das heißt, wenn alle Teile für die Eilaufträge in der Montage vorliegen, werden sie bevorzugt bearbeitet.

Das Pufferlager der zentralen Lösung wurde hierbei nur für Eilaufträge vollständig ausgeschöpft. Ansonsten wird immer ein Puffer freigehalten, um einen Eilauftrag bearbeiten zu können und Blockierungen des Produktionssystems zu vermeiden. Bei der Ausnutzung aller vier Puffereinheiten mit Bauteilen könnte es nämlich sein, dass ausschließlich Teile, welche nicht von der Montage weiterbearbeitet werden können, produziert wurden.

Die Planung, für das in dieser Arbeit entwickelte dezentrale Steuerungssystem erfolgt wie bei der zentralen Planung. Das bedeutet, dass neue Aufträge an das Ende der Warteschlange eines Aufgabenspeichers gelegt werden. Die Verteilung und Einlastung der Aufträge werden anschließend von der dezentralen Produktionssteuerung übernommen. Die dezentrale Produktionsteuerung zielt darauf ab, dass die Fertigstellungstermine eingehalten werden. Anschließend werden nach den Ablaufdiagrammen aus Abschnitt 4.2 die Aufträge auf den Stationen eingelastet.

Von dem Pufferlager nutzt die dezentrale Produktionssteuerung nur zwei Puffereinheiten, damit Umplanungen durchgeführt werden können. In der folgenden Tabelle sind die

eingestellten Parameter für das Anwendungsszenario mit der dezentralen Steuerung dargestellt. In diesem Anwendungsszenario sind keine Kapazitätserhöhungen vorgesehen, weswegen keine Kapazitätsstufe definiert wurde. Es werden keine vordefinierten Prioritäten verwendet, so dass der Algorithmus eigenständig die beste Produktionsreihenfolge ermitteln kann. Die Arbeitsstationen sind baugleich, weswegen keine Berechnungen für die am besten geeignete Arbeitsstation durchgeführt werden. Als Bestandsgrenze wurde der gesamte Simulationszeitraum gewählt. Der Steuerungshorizont liegt ebenso bei 4 Stunden. Das bedeutet, dass alle Aufträge direkt auf den Stationen eingeplant und nicht vorgehalten werden. Die Reihenfolge wird nach dem frühesten Zieltermin gebildet.

Tabelle 5: Produktions- und Steuerungsparameter bei der Durchführung

Kapazitätsstufe	Keine Kapazitätsstufe, da keine Kapazitätserhöhungen vorgesehen
Prioritäten	Keine Prioritäten
Geeignetste Arbeitsstation	Keine Berechnung der am besten geeigneten Arbeitsstation, da alle Arbeitsstationen baugleich
Bestandsgrenze	4 Stunden (Ende der Untersuchung)
Steuerungshorizont	4 Stunden (Ende der Untersuchung)
Kritische Aufträge	Alle Eilaufträge und alle Aufträge mit festgelegten Terminfristen (Reihenfolge nach frühester Zieltermin)

7.2.1 Erste Untersuchung mit initialen Aufträgen

In der ersten Untersuchung zur Überprüfung der Funktionalität beider Steuerungssysteme werden 50 initiale Aufträge in das Produktionssystem gegeben. Das heißt, dass während der Untersuchung keine neuen Aufträge in das Produktionssystem hinzukommen. Alle Aufträge sollen innerhalb der vordefinierten Untersuchungsdauer abgearbeitet werden.

Das folgende Gantt Chart (siehe Abbildung 47) illustriert die Ausführung der Produktion mit zentraler Steuerung, die als Referenz genommen wird. Dabei sind die 8 Arbeitsstationen (2x Frässtation, 2x 3D-Drucker-Station, 2x Drehstation, 2x Montagestation) dargestellt. Alle Aufträge sind mit 1-50 nummeriert. Die gesamte Untersuchungsdauer beträgt 4 Stunden. Da ohne Auftragsvorrat simuliert wird, muss die Montagestation am Anfang auf die Fertigstellung der Produktionsaufträge von Frässtation, Drehstation und 3D-Drucker-Station warten.

Während der Untersuchung gab es keine Veränderungen an den äußeren Randbedingungen, daher können die Aufträge einfach nach der vordefinierten Reihenfolge abgearbeitet werden. Bei der Produktion an der Drehstation gibt es keine

Unterbrechung, da sich dieser Arbeitsstation der Engpass befindet. Die Montage muss hierbei auf die Teile aus der Drehstation warten. Die 3D-Drucker-Station und die Frässtation produzieren auf Grund der kürzeren Prozesszeiten vor, bis die Puffer an den Arbeitsstationen gefüllt sind. Sobald die Puffer gefüllt sind, wird auf die Abarbeitung durch die Montage gewartet. Die Abbildung 47 stellt die Untersuchungsergebnisse dar. Dabei ist zu sehen, dass an den beiden 3D-Drucker-Arbeitsstationen jeweils eine Unterbrechung nach dem Auftrag 27 und dem Auftrag 28 vorgekommen ist, da die Pufferlager an den beiden Arbeitsstationen gefüllt wurden. Alle Aufträge können problemlos im Untersuchungszeitraum produziert werden und es wird die vorher definierte Auftragsreihenfolge abgearbeitet.

F_1: Frässtation 1 F_2: Frässtation 2 A_1: 3D-Drucker-Station 1

A_2: 3D-Drucker-Station 2 D_1: Drehstation 1 D_2: Drehstation 2

M_1: Montagestation 1 M_2: Montagestation 2

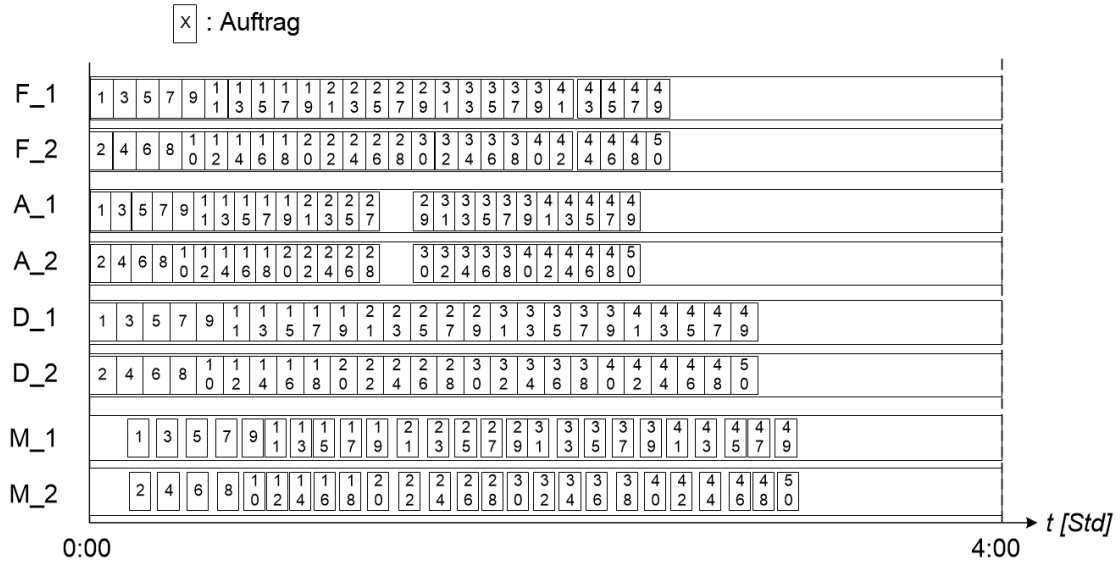

Abbildung 47: Gantt-Chart – Darstellung der ersten Untersuchungsergebnisse der zentralen Steuerung (Referenzergebnisse)

Die Ausführung der dezentralen Produktionssteuerung ist analog zu den Referenzergebnissen. Es wurden, wie bei der Referenz, nur die initialen Aufträge abgearbeitet. Hierbei waren keine Umplanungen erforderlich, weswegen auch das Ergebnis, dem der zentralen Ausführung gleicht. Der einzige Unterschied zu der Referenz ist, dass die Unterbrechungen bzw. Wartezeiten der Frässtationen und 3D-Drucker-Stationen eher erfolgen, da von dem Puffer nur 2 anstelle von 4 Einheiten verwendet wurden (siehe Abbildung 48). Ansonsten wird genau wie bei der Referenzlösung bei der Drehstation, dem Engpass des Produktionssystems, durchproduziert. Die Montage wartet anschließend auf die Bauteile der Drehstation.

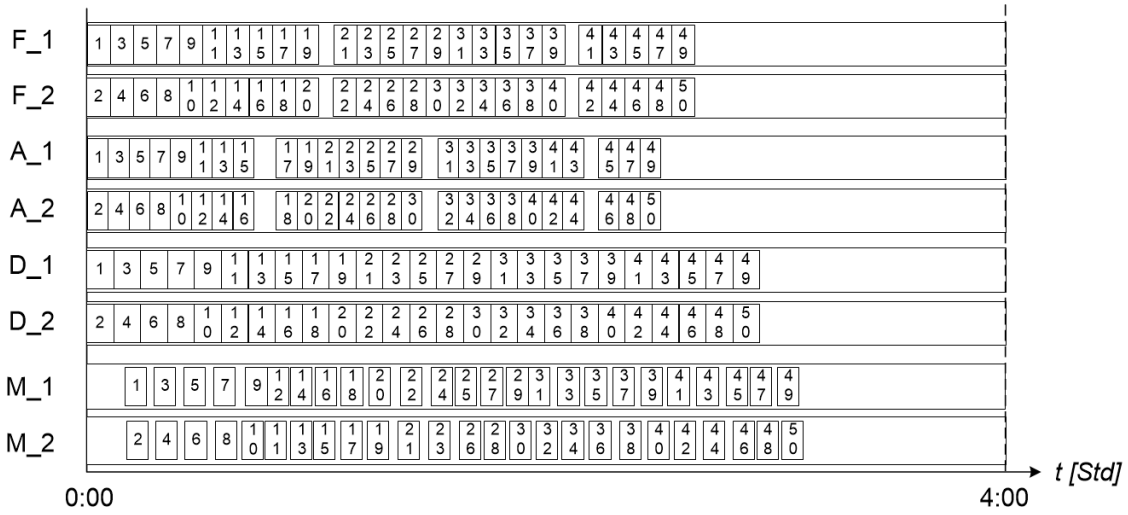

Abbildung 48: Gantt-Chart – Darstellung der ersten Untersuchungsergebnisse der dezentralen Steuerung

Die genauen Ergebnisse dieser Untersuchungen werden in der folgenden Tabelle 6 dargestellt. Entsprechend gibt es bei den Untersuchungsergebnissen der beiden Modelle fast keine Unterschiede. Einzig die Umlaufbestände sind bei der dezentralen Steuerung auf Grund der Nutzung eines geringeren Puffers etwas kleiner. Die Auslastung beider Lösungen ist gleich. Beide Lösungen stellen die 50 Aufträge zum gleichen Zeitpunkt fertig. Die Auslastung wird hierbei über die Gesamtproduktionsdauer berechnet. Die Durchlaufzeiten bei beiden Steuerungssystemen sind ebenso gleich. Dies ist hauptsächlich dem Umstand geschuldet, dass keinerlei Änderungen und Störungen in dem Produktionssystem vorkamen und somit auch keine Änderungen an dem Produktionsplan notwendig waren.

Tabelle 6: Ergebnisse erste Untersuchung

	DZ [min]	B [Stück]	A [%n]	F [min]	E-A [%]	DS [Anzahl Aufträge]
Referenz	16:52	2,23	80,1	187:15	93,9	50
Dezentrale Produktionssteuerung	16:54	1,57	80,1	186:12	93,9	50

DZ : Durchlaufzeit B: Umlaufbestand A: Auslastung

F: Fertigstellungszeitpunkt E-A: Engpass-Auslastung DS: Durchsatz

7.2.2 Zweite Untersuchung mit Einspielung von Eil- und Terminfrist-Aufträgen

In dieser Untersuchung sollen durch das Einspielen neuer Aufträge die Randbedingungen der Produktion geändert werden. Dadurch kann die automatisierte Umplanungsfunktion der dezentralen Produktionssteuerung veranschaulicht werden. Hierfür wurden zusätzlich zu dem Auftragsset der 50 Aufträge alle 20 Minuten die folgenden Aufträge in das System eingespielt:

- 1x Eilauftrag (Auftrag, welcher so schnell wie möglich produziert werden soll)
- 1x Auftrag ohne Wunschtermin
- 1x Terminfrist-Auftrag mit Terminfrist 40 Minuten nach der Einspielung

Die zentrale Steuerung hat auf die Eilaufträge wie folgt reagiert: die Eilaufträge wurden einfach auf einer geeigneten Arbeitsstation als Nächstes ausgeführt. Dabei wird die erste Arbeitsstation (in der Regel Arbeitsstation 1) ausgewählt. Die Zuweisung erfolgt dabei über den zentralen Auftragsspeicher.

Für die Terminfrist-Aufträge gibt es bei der Referenzlösung keine Routine zur Einplanung. Diese werden von der Steuerung als normale Aufträge behandelt und in dem Auftragsspeicher belassen, bis freie Kapazität an der Arbeitsstation verfügbar ist und anschließend an das Ende der Warteschlange positioniert.

In Abbildung 49 wird der Produktionsplan dieses Durchlaufs als Referenz dargestellt. Sie veranschaulicht, dass die Terminfrist-Aufträge nicht berücksichtigt wurden bzw. als normale Aufträge behandelt wurden. Dabei konnten bei allen Terminfrist-Aufträgen die Terminfristen nicht eingehalten werden. Zum Beispiel sollte der Auftrag D1 vor der Terminfrist 1 fertig bearbeitet werden. Die Eilaufträge wurden eingeplant und direkt an nächster Position bearbeitet. Aber ihre Zuweisung zu Arbeitsstationen wurde nur nach der fest definierten Regel bearbeitet. Das heißt in diesem Fall, dass die Eilaufträge immer nur den ersten Arbeitsstationen, nämlich F1, A1, D1 und M1 zugeordnet wurden. Das führt zu einem ineffizienten Produktionsdurchlauf, da die Warteschlange der ersten Station immer länger wird. Darüber hinaus resultieren aus den Wechselwirkungen zwischen den Arbeitsstationen durch die Eilaufträge mehrfach Verzögerungen.

Die dezentrale Produktionssteuerung hingegen plant die Eilaufträge und die Terminfrist-Aufträge korrekt ein. Die Reihenfolge der Aufträge wurde durch die dezentrale Steuerung unter Berücksichtigung aller Randbedingungen automatisiert berechnet. Eilaufträge wurden als Nächstes produziert. Terminfrist-Aufträge wurden von der Montage ausgehend auf die Arbeitsstationen rückwärtsterminiert eingeplant. Bei jeder neuen Einbringung von Aufträgen wird hierbei die gesamte Produktion automatisiert umgeplant. Zum Beispiel kamen an der Frässtation 1 nach dem Auftrag 5 ein normaler Auftrag ohne Wunschtermin N1, ein Eilauftrag E1 und ein Terminfrist-Auftrag D1 im Produktionssystem an. Der Eilauftrag E1 wurde sofort bearbeitet, obwohl nach dem

ursprünglichen Produktionsplan der Auftrag 7 in der Zeit hätte bearbeitet werden sollen. Die Aufträge 7, 8, 9 wurden dabei umgeplant, bevor die Aufträge 10 und 11 ausgeführt wurden. Gleichzeitig wurde der Terminfrist-Auftrag D1 rückwärtsterminiert eingeplant, so dass die Terminfrist eingehalten wurde. Da für den normalen Auftrag N1 kein Wunschtermin definiert wurde, wurde dieser einfach an das Ende der Warteschlange positioniert. In der Abbildung 50 sind alle Umplanungen durch die grau markierten Aufträge zu erkennen.

N: Normaler Auftrag E: Eilauftrag D: Terminfrist-Auftrag

F_1: Frässtation 1 F_2: Frässtation 2 A_1: 3D-Drucker-Station 1

A_2: 3D-Drucker-Station 2 D_1: Drehstation 1 D_2: Drehstation 2

M_1: Montagestation 1 M_2: Montagestation 2

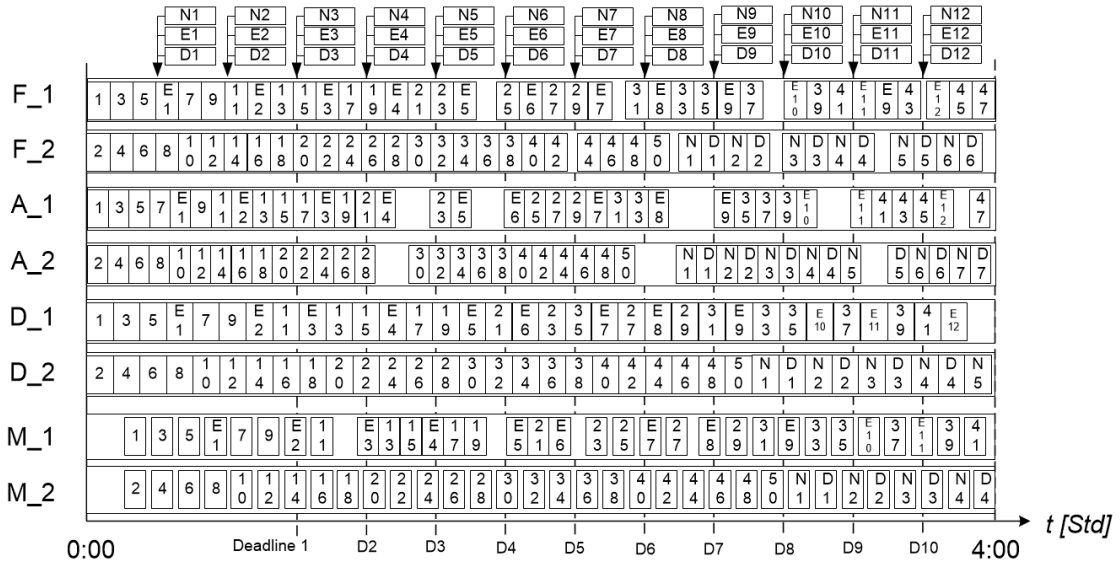

Abbildung 49: Gantt-Chart – Darstellung der zweiten Untersuchungsergebnisse der zentralen Steuerung (Referenzergebnisse)

Die automatisierte Umplanung von Terminfrist-Aufträgen zeigt bereits, dass das Produktionssystem durch die Einführung der dezentralen Produktionssteuerung sich automatisiert steuern kann. Alle Eilaufträge wurden nach der Einbringung direkt bearbeitet. Alle Terminfrist-Aufträge konnten im Rahmen des Zeitrahmens produziert werden und haben ihre Terminfristen eingehalten.

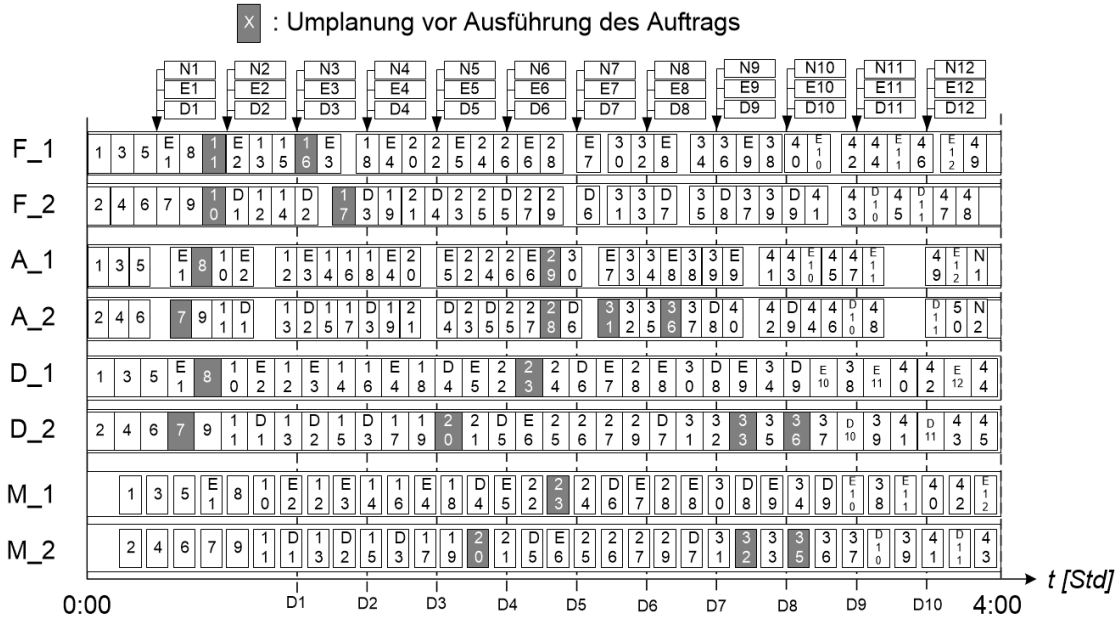

Abbildung 50: Gantt-Chart – Darstellung der zweiten Untersuchungsergebnisse der dezentralen Steuerung

In dieser Untersuchung zeigen sich die Vorteile der dezentralen Steuerung im Vergleich zu der Referenzlösung (siehe Tabelle 7). Die Termintreue von Terminfrist-Aufträgen konnte von der zentralen Steuerung nicht eingehalten werden. Im Gegensatz dazu konnte die dezentrale Steuerung alle Terminfrist-Aufträge rechtzeitig fertigstellen. Die Durchlaufzeit der Eilaufträge ist bei der Referenzlösung höher. Dies liegt im Wesentlichen daran, dass die Eilaufträge immer der gleichen Station zugewiesen wurden. Die geringeren Umlaufbestände der dezentralen Steuerung ergeben sich durch die geringeren Puffer und die häufigeren Umplanungen. Bei der Auslastung und dem Durchsatz gibt es bei beiden Lösungen nur geringe Unterschiede, da die Simulation zeitlich begrenzt ist und Überproduktion deswegen vernachlässigt wird.

Tabelle 7: Ergebnisse der zweiten Untersuchung

	TT Terminfrist [%]	DZ [min]	DZ Eil [min]	B [min]	A [%]	DS [Anzahl Aufträge]
Referenz	0	24:20	17:07	2,76	86,2	63
Dezentrale Produktions-steuerung	100	22:15	17:24	1,82	85,7	66

TT: Termintreue DZ: Durchlaufzeit DZ Eil: Durchlaufzeit Eilaufträge

B: Umlaufbestand A: Auslastung DS: Durchsatz

7.2.3 Dritte Untersuchung mit der Einspielung von Störungen

Die dritte Untersuchung soll die Selbstorganisation der dezentralen Produktionssteuerung zeigen. Die dritte Untersuchung baut auf der zweiten Untersuchung auf. Dabei wurden nicht nur zusätzlich zu dem Auftragsset der initialen 50 Aufträge alle 20 Minuten Eilaufträge, normale Aufträge ohne Wunschtermine und Terminfrist-Aufträge mit Terminfrist 40 Minuten in das Produktionssystem eingespielt, sondern es wurden zudem Störungen in die Produktion eingebaut. Die Störungen wurden als längere Pausenzeiten an den Arbeitsstationen simuliert. Diese Störungen können im realen Betrieb zum Beispiel die Folgenden sein:

- längere Bearbeitungszeiten
- Ausfall einer Maschine
- Unerwartete Rüstzeiten
- Ungeplante Wartung
- Nacharbeit, etc.

Es wurde hierbei eine durchschnittliche Verfügbarkeit der Produktionsanlagen von 80% angenommen und die Störungen zufällig als Szenario generiert. Die Störungen wurden wie in Abbildung 51 gezeigt in den Durchlauf eingebaut. Bei den Referenzergebnissen zeigen sich bereits bei der Abarbeitung von Aufträgen ohne Terminfristen erhebliche Probleme. Die Aufträge wurden nur nacheinander bearbeitet, was zu langen Stillstandszeiten und Wartezeiten geführt hat. Zum Beispiel wurden an der 3D-Drucker-Station 2 die Aufträge 42, 44 und 46 abgearbeitet und danach auf Grund von blockierten Aufträgen lange Wartezeiten erzeugt. Auch wenn im Engpass der Drehstationen nahezu durchproduziert wurde, wurden teilweise die falschen, nicht dringend benötigten Aufträge produziert, was wieder zu weiteren Wartezeiten für die Montage führt.

Durch die Einspielung der Eilaufträge resultierten bereits mehrere Wartezeiten bzw. Unterbrechungen (siehe Abbildung 50). Der Einbau von Störungen verstärkt diese Unterbrechungen erheblich. Diese Unterbrechungen führen zu einer niedrigen Auslastung, längeren Durchlaufzeiten und höheren Umlaufbeständen. Es wurde zwar versucht die Eilaufträge an der nächsten Position zu bearbeiten, aber da bei einer Störung keine Umplanung auf eine andere Station stattfand, wurden die Durchlaufzeit der Eilaufträge erheblich verlängert.

N: Normaler Auftrag	E: Eilauftrag	D: Terminfrist-Auftrag
F_1: Frässtation 1	F_2: Frässtation 2	A_1: 3D-Drucker-Station 1
A_2: 3D-Drucker-Station 2	D_1: Drehstation 1	D_2: Drehstation 2
M_1: Montagestation 1	M_2: Montagestation 2	

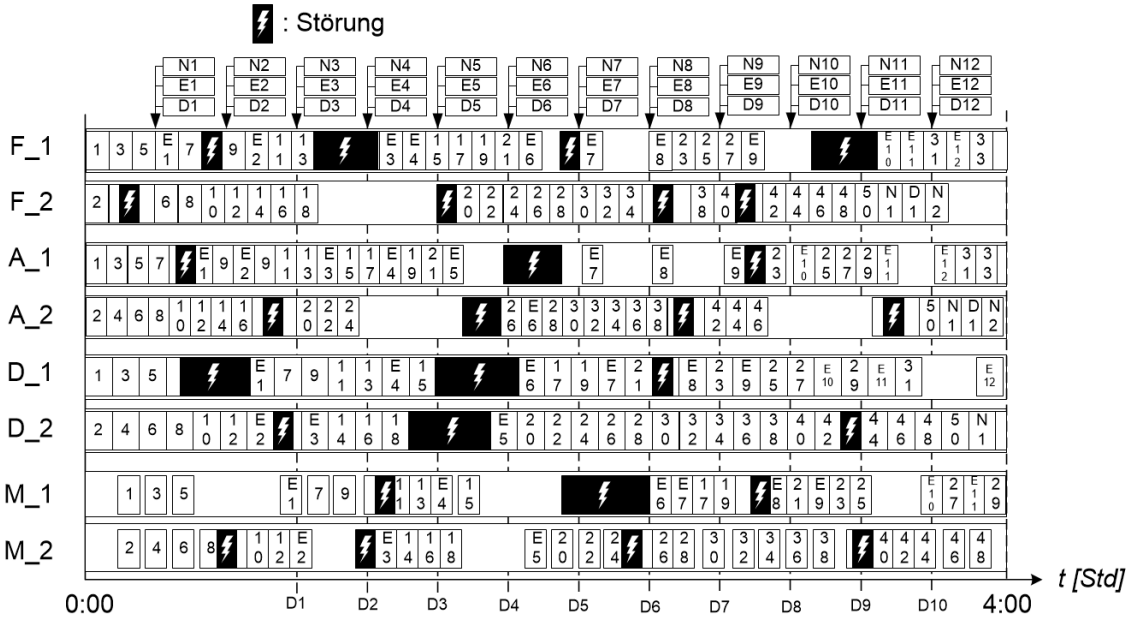

Abbildung 51: Darstellung der dritten Referenz-Untersuchungsergebnisse

Das gleiche Störungsszenario wurde auch auf das dezentrale Steuerungssystem angewendet. Das Verhalten der dezentralen Steuerung ist in der Abbildung 52 dargestellt. Alle Eilaufträge wurden mithilfe von Umplanungen als nächste Position in der Warteschlange produziert. Zum Beispiel wurde der Eilauftrag E3 an der 3D-Drucker-Station 2 bearbeitet, da die 3D-Druck-Station 1 nach der Einlastung eine Störung hatte. Die dezentrale Produktionssteuerung erreichte, dass alle Terminfrist-Aufträge plangemäß erfüllt wurden und jedes Mal eine Umplanung eingeleitet wurde, sobald eine Störung ein Fertigstellungsdatum gefährdete. Damit ist die Funktionsfähigkeit der Selbstorganisation der automatisierten dezentralen Steuerung nachgewiesen.

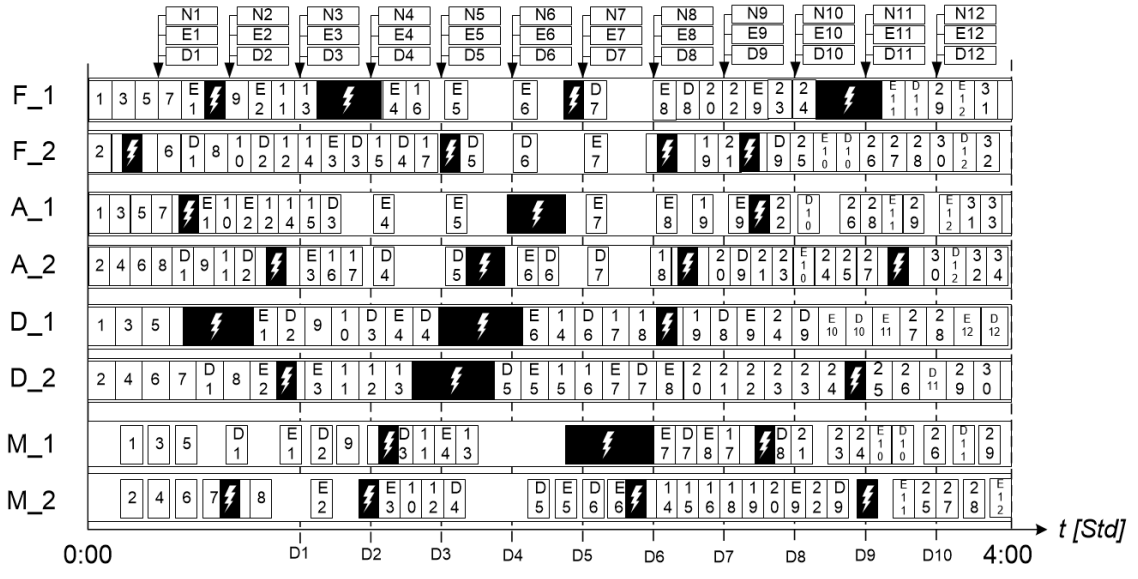

Abbildung 52:Darstellung der Untersuchungsergebnisse der dezentralen Steuerung

In der Tabelle 8 sind die genauen Untersuchungsergebnisse dargestellt. Die dezentrale Produktionssteuerung konnte die Fertigstellungstermine aller Terminfrist-Aufträge einhalten. Daraus ergibt sich eine Termintreue von 100%. Gleichzeitig hat sie eine kurze Durchlaufzeit im Vergleich zum Referenzmodell. Die Durchlaufzeit der Eilaufträge vom Referenzmodell ist zwar kürzer, aber dies geschieht auf Kosten mehrerer Unterbrechungen.

Tabelle 8: Ergebnisse Störungsverhalten

	TT Terminfrist [%]	DZ [min]	DZ Eil [min]	B [min]	A [%]	DS [Anzahl Aufträge]
Referenz	0	37:58	24:47	2,98	72,2	50
Dezentrale Produktionssteuerung	100	33:45	30:38	1,84	74,5	52

TT: Termintreue DZ: Durchlaufzeit DZ Eil: Durchlaufzeit Eilaufträge

B: Umlaufbestand A: Auslastung DS: Durchsatz

Die Ergebnisse in Tabelle 8 weisen nach, dass die dezentrale Produktionssteuerung die Produktion selbstorganisiert steuern kann. Damit werden die wesentlichen Funktionen der dezentralen Steuerung (die automatisierte Umplanung und die automatisierte Reihenfolgebildung) und die sich dadurch ergebenden Vorteile gezeigt. Die Validierung der dezentralen Produktionssteuerung im Smart Automation Labor bestätigt die folgenden Vorteile:

- Gewonnene Flexibilität durch die Möglichkeit der dynamischen Umplanung des dezentralen Produktionssteuerungssystems. Beim Auftreten von Fehlern (z. B. Ausfall einer Maschine) wird eine Umplanung automatisiert eingeleitet, die dazu führt, dass die Fertigstellungstermine der Aufträge eingehalten werden können. Diese Umplanung führt allerdings dazu, dass die Auslastung des Gesamtsystems gesenkt wird und die Umlaufbestände kurzfristig erhöht werden.
- Die initialen Einlastungsergebnisse zwischen der zentralen Steuerung und der dezentralen Steuerung unterscheiden sich nicht. Da die Datenbasis der Planung und der einzelnen Arbeitsstationen im Smart Automation Labor der umgesetzten Produktionsszenarien gleich war, ergaben sich keine Verbesserungen durch den implementierten Einlastungsprozess. Die Vorteile der dezentralen Steuerung ergeben sich somit erst durch die dynamische Anpassung des Produktionssystems an die Gegebenheiten im Betrieb.

7.3 Validierung durch Simulation zum Vergleich mit anderen Steuerungsverfahren im Programm Plant Simulation

Da in der Validierung in dem Labor nur das alte zentralen Steuerungssystem mit der dezentralen Produktionssteuerung verglichen werden konnte, wird eine Untersuchung mit weiteren Steuerungsverfahren durchgeführt. Dabei wurden neben der automatisierten dezentralen Produktionssteuerung, das Referenzmodell der zentralen SPS-Steuerung und das CONWIP-Verfahren simuliert. CONWIP hat eine weite Verbreitung in der Industrie und es lässt sich einfach auf unterschiedliche Produktionssysteme übertragen (siehe Abschnitt 3.1.3). Die Untersuchung findet in einer Simulationsumgebung des Programms Plant Simulation statt. Als Produktionsszenario wird hierbei weiterhin die Produktion des Fernsteuerungsautos in dem Smart Automation Labor verwendet. In dem ersten Schritt wurde das Labor modelliert (siehe Abbildung 53). Dabei sind die Arbeitsstationen über ein Einschienensystem miteinander verbunden. Gleichzeitig wird ein fahrerloses Transportsystem für den Transport von Rohmaterial, Zukaufteilen und zum Abtransport der Fertigteile verwendet. Nach der Simulation des Labors wurden die einzelnen Durchläufe, die in Abschnitt 7.2 beschrieben wurden, durchgeführt.

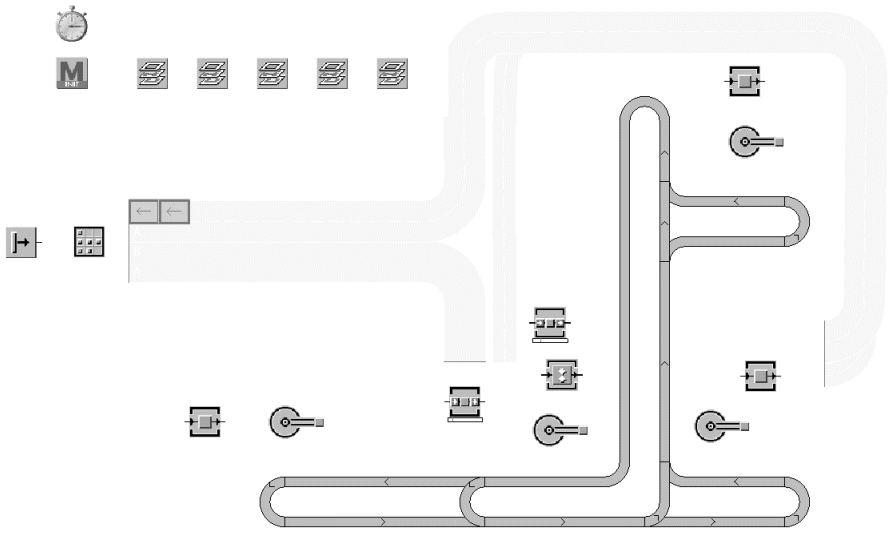

Abbildung 53: Plant Simulation Nachbildung des Smart Automation Labors

Bei den Durchläufen hat sich gezeigt, dass der Einsatz des Einschienensystems als Logistiksystem zu Problemen in der Simulation führen kann. Durch den Einsatz können Blockierungen von Aufträgen oder der Shuttles vorkommen. Diese verzerren die Simulationsergebnisse und zeigen dadurch Resultate, welche nicht auf die Effizienz des Produktionssteuerungsverfahrens zurückschließen lassen. Aus diesem Grund wurde für die Simulation das Logistiksystem weggelassen und stattdessen dieses als Zeit, wie bereits bei der Simulation aus Abschnitt 7.2, simuliert. Es ergibt sich für das Smart Automation Labor in Plant Simulation somit das in Abbildung 54 dargestellte Modell.

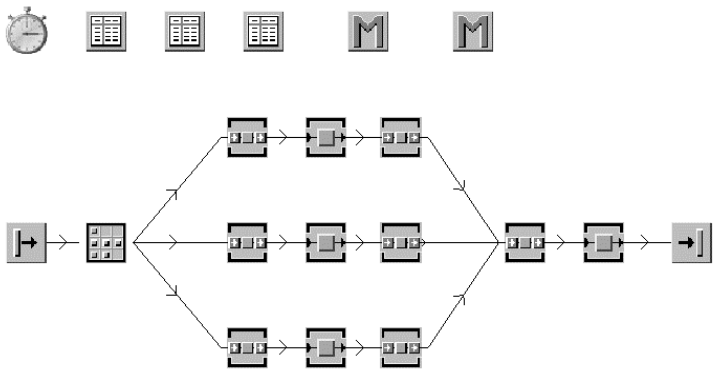

Abbildung 54: Plant Simulation-Modell der vereinfachten Simulation

Anpassung des Anwendungsszenarios

Die Einstellungen und Zeiten wurden von dem in Abschnitt 7.1 definierten Anwendungsszenario übernommen. Als Bearbeitungszeiten der einzelnen Arbeitsstationen werden die folgenden Werte angenommen:

- 3D-Druck des Chassis 5:30 Minuten
- Fräsen des Unterbodens 6:07 Minuten
- Drehen der Räder 7:02 Minuten
- Endmontage 5:36 Minuten
- Transportzeit zwischen den Stationen: 30 Sekunden

Zusätzlich werden bei der Simulation Schwankungen in den Bearbeitungszeiten eingebracht. Bei den Fertigungsprozessen wird hierbei eine Schwankung von 3% simuliert und bei der Montage eine Schwankung von 15%.

Die Parameter des Anwendungsszenarios werden wie folgt festgelegt:

- Eine Produktionswoche im Einschichtbetrieb mit 5 Tagen - 8 Stunden: Gesamtsimulationszeitraum von 40 Stunden.
- Maschinenverfügbarkeit von 95%.
- Verfügbarkeit der Beschäftigten, bzw. der Montageanlage von 85%.
- Herstellung des Fernsteuerungsautos mit jeweils zwei gleichartigen Arbeitsstationen, also insgesamt 2 Drehstationen, 2 Frässtationen, 2 3D-Drucker-Stationen, 2 Montagestationen.
- Puffergröße vor und nach den Arbeitsstationen: 50 Stück
- Kapazitätserhöhungen sind nicht vorgesehen, die Kapazitätsplanung wird somit nicht angewendet.

- Die Randbedingungen aus Abschnitt 7.2 bleiben hierbei erhalten.

Zusätzlich wird angenommen, dass alle Aufträge von einer Produktionsplanung bereits geplant wurden und diese auf die Anlange verteilt wurden. Im realen System könnte dies bereits im Rahmen einer Produktionsplanung mit Hilfe des MRP-II-Verfahren erfolgt sein.

Der Fertigstellungstermin wird hierbei von dem Auftragsset vorgegeben. Die 300 Aufträge werden gleichmäßig auf die 10 Fertigstellungstermine verteilt. Das heißt, dass alle 4 Stunden 30 Aufträge von dem Produktionssystem abgearbeitet werden müssen, um alle Termine einzuhalten.

Darüber hinaus wurden die folgenden Einstellungen bei der Simulation der Untersuchung angenommen. Die Produktionsparameter müssen realitätsnah eingestellt werden. Um die Steuerung für die individuelle Produktion nachzuweisen, wurde als Losgröße eines Auftrags „1“ gewählt.

Tabelle 9: Parameter und Eigenschaften der simulierten Aufträge

Bauteil	Fernsteuerungsauto
Losgröße	1
Starttermin	0:00
Fertigstellungstermin	4:00, 8:00, 12:00, 16:00, 20:00, 24:00, 28:00, 32:00, 36:00, 40:00
Anzahl an Aufträgen	300 mit Terminfrist

Durchführung der simulationsbasierten Validierung

Zuerst wird kurz auf den Ablauf der drei ausgewählten Verfahren eingegangen. Bei dem Referenzmodell der zentralen SPS-Steuerung wird eine direkte Auftragsfreigabe angewandt. Der Reihenfolgeplan wird einfach auf die Arbeitsstationen weitergeleitet (siehe Abschnitt 7.2). Das CONWIP -Verfahren ist auf Produktionsstraßen ausgelegt. Daher wurde das Verfahren in diesem Fall leicht angepasst (siehe Tabelle 10). Die CONWIP -Steuerung läuft dabei wie folgt ab: nach dem Abgang eines Auftrags wird der nächste Auftrag an die Fertigungsstationen, also hierbei an die Frässtation, 3D-Drucker-Station und Drehstation, übergeben. Anschließend wird dieser Auftrag an der Montagestation fertig montiert. In der folgenden Tabelle ist die Parametereinstellung der CONWIP -Steuerung dargestellt.

Tabelle 10: Parameter der eingesetzten CONWIP -Steuerung

Bestandsgrenze	10 Karten
Einlastregel	frühester Zieltermin
Abarbeitungsregel	Zuerst Terminfrist, anschließend Wartezeitregel
Anzahl an Aufträgen	300

Die dezentrale Produktionssteuerung übernimmt zuerst die von der Produktionsplanung berechnete Auftragsreihenfolge. Dann wird die Auftragsausführung mit Hilfe der Verfahren zur Kapazitätssteuerung, Reihenfolgebildung und Auftragsfreigabe automatisiert gesteuert. Wenn Störungen auftreten, werden eventuell Umplanungen eingeleitet (siehe Kapitel 3). Es wurde versucht, die dezentrale Produktionssteuerung in Plant Simulation nachzubilden. Hierbei wurde versucht die Abläufe zur Auftragsfreigabe, zur Reihenfolgebildung und zur Umplanung in Plant Simulation nachzubilden. Auf Grund nicht vorhandener Möglichkeiten zur Kapazitätserhöhung wurde auf eine Nachbildung der Kapazitätsplanung verzichtet. Folgende Parameter wurden für die Steuerung gewählt:

Tabelle 11: Parameter dezentrale Produktionssteuerung

Kapazitätsstufe	Keine Kapazitätsstufe, da keine Kapazitätserhöhungen vorgesehen
Prioritäten	Keine Prioritäten
Geeignetste Station	Keine Berechnung der geeignetsten Station, da alle Stationen baugleich
Bestandsgrenze	4 Stunden
Steuerungshorizont	4 Stunden
Kritische Aufträge	Alle Eilaufträge und alle Aufträge mit Terminfrist (Reihenfolge nach frühester Zieltermin)

Die Ausführung aller durchgeführten Simulationen führte zu den folgenden Ergebnissen:

Tabelle 12: Übersicht über alle Simulationsergebnisse

	Referenzmodell	CONWIP	Dezentrale Produktionssteuerung
TT Terminfrist [%]	77	83	86
TAA [min]	-	6:38	3:27
DZ [min]	29:31	27:21	28:55
DZ Eil [min]	22:12	21:18	21:11
B [min]	33,3	25,8	19,7
A [%]	90,2	86,4	87,6
DS	298	295	299

TT: Termintreue TAA: Terminabweichung DZ: Durchlaufzeit

DZ Eil: Durchlaufzeit Eilaufträge B: Umlaufbestand A: Auslastung

DS: Durchsatz

Das Ergebnis zeigt, dass die in dieser Arbeit entwickelte dezentrale Steuerung im Vergleich zu der Referenz und dem CONWIP -Verfahren bei der Produktion individueller Produkte eine deutlich bessere Termintreue auf Kosten von etwas höheren Umlaufbeständen, einer etwas geringeren Auslastung und etwas höheren Durchlaufzeiten erreicht. Gegenüber der Referenz der zentralen SPS-Steuerung zeigt die dezentrale Steuerung Vorteile bei fast allen Werten, außer bei dem Durchsatz. Der Grund für den höheren Durchsatz bei dem Referenzmodell liegt in der höheren Auslastung des Produktionssystems begründet. Das Referenzmodell hat eine höhere Auslastung, da keine Umplanungen durchgeführt werden.

In diesem Kapitel wurde, das in dieser Arbeit entwickelte, automatisierte dezentrale Steuerungssystem validiert. Zuerst wurde der Demonstrator „Fernsteuerungsauto“ und das dazugehörige Anwendungsszenario vorgestellt. Anschließend wurde die Validierung in der realen Umgebung des Smart Automation Labors durchgeführt. Dabei wurden durch drei Untersuchungen die Vorteile bzw. die Funktionalitäten des dezentralen Steuerungssystems im Vergleich zu der Referenz mit zentraler Steuerung nachgewiesen. In der ersten Untersuchung wurde die Funktionsfähigkeit der dezentralen Steuerung validiert. Dabei wurden 50 bereits eingeplante Aufträge abgearbeitet und hierbei ergaben sich keine Unterschiede zwischen den beiden Verfahren. In der zweiten Untersuchung wurden unter anderem Eilaufträge und Terminfrist-Aufträge während des Betriebs im Produktionssystem eingespielt. Die dezentrale Steuerung hat die Eil- und Terminfrist-Aufträge im Gegensatz zur Referenzlösung korrekt eingeplant. In der dritten

Untersuchung wurde die Funktionsfähigkeit der Selbstorganisation durch die Einspielung von Störungen während der Laufzeit validiert. Hierbei hat sich gezeigt, dass die dezentrale Produktionssteuerung eigenständig Umplanungen einleitet und dadurch jederzeit eine optimale Auftragszuweisung gewährleistet. Um das entwickelte Verfahren mit anderen, heutzutage in der Industrie eingesetzten Produktionssteuerungsverfahren zu vergleichen, wurde das Labor in einer Simulationsumgebung modelliert. In dieser Simulationsumgebung wurde ein Produktionsszenario für die Produktion individueller Produkte implementiert und neben der Referenzlösung und der dezentralen Produktionssteuerung das CONWIP-Verfahren implementiert. Hierbei konnte nachgewiesen werden, dass durch den Einsatz der dezentralen Produktionssteuerung die Liefertreue deutlich gesteigert werden konnte. Es lässt sich somit festhalten, dass durch die automatisierte Umplanung der dezentralen Produktionssteuerung die Termintreue gesteigert werden kann.

8 Schlussbetrachtung

Zu Beginn der Arbeit wurde der Wandel der Industrie hin zu einer flexiblen Produktion mittels CPPS vorgestellt. Die wirksame Umsetzung von Flexibilität in der Produktion erfordert eine organisationstechnische Flexibilisierung, die durch dezentrale Strukturen erreicht werden kann. Daher wurde in dieser Arbeit das Verfahren zur automatisierten dezentralen Produktionssteuerung für CPPS mit digitaler Repräsentation der Beschäftigten entwickelt. Das dabei entstandene automatisierte Produktions-steuerungssystem stellt einen ersten Ansatz für eine vollumfänglich integrierte Lösung zur flexiblen Steuerung von CPPS dar.

Kern des Steuerungssystems ist ein dezentrales Steuerungsverfahren, das automatisiert auf den Arbeitsstationen ausgeführt wird und mit dem sich die Arbeitsstationen selbstständig untereinander koordinieren. Dieses Steuerungsverfahren beinhaltet Algorithmen zur Kapazitätssteuerung, zur Auftragsfreigabe und zur Reihenfolgebildung. Damit ist das Produktionssystem in der Lage, die Produktion selbstorganisiert zu steuern. Beschäftigte werden in das Steuerungssystem über eine digitale Repräsentation eingebunden. Hierzu werden notwendige Informationen über die Beschäftigten sowie deren Fähigkeiten, Einsatzzeiten und Bedürfnisse formalisiert und mit Hilfe von Datenbanken digital abgebildet. Diese digitale Abbildung wurde erweitert, sodass die Beschäftigten die Möglichkeit erhalten, sie betreffende maschinelle Entscheidungsvorgänge zu beeinflussen und bei Entscheidungen zu intervenieren. Diese Einbindung der Beschäftigten geschieht weitestgehend automatisiert. Die Repräsentation sorgt dafür, dass die Planung der Beschäftigten verbessert wird, die Zuweisungsprozesse optimiert und bei Entscheidungen der Produktionssteuerung die Bedürfnisse der Beschäftigten berücksichtigt werden. Die Planung wird im Wesentlichen durch die verbesserte Datenbasis über die Beschäftigten optimiert. Indem nicht nur Durchschnittswerte, sondern individuelle Prozesszeiten für die Planung herangezogen werden, kann die Produktionsausführung besser prognostiziert werden. Die bessere Prognose sorgt wiederum für bessere Planungsentscheidungen. Die Zuweisungsprozesse von Beschäftigten zu Arbeitsstationen wurden mit Hilfe der digitalen Repräsentation neu gestaltet. Wo bisher nur im Rahmen der Kapazitätsplanung mittelfristige Zuweisungen erfolgen, ermöglicht die digitale Repräsentation automatisiert kurzfristige Anpassungen, um die Flexibilität des Produktionssystems zu erhöhen. Bei dem Zuweisungsprozess können zudem die Wünsche und Bedürfnisse der Beschäftigten berücksichtigt werden. Dementsprechend werden durch die Einführung der digitalen Repräsentation nicht nur die Planung und Steuerung im produktionstechnischen Sinne verbessert, sondern es wird auch das Arbeitsumfeld der Beschäftigten aufgewertet.

Abschließend wurde das dezentrale Steuerungssystem zweistufig validiert. Die erste Stufe der Validierung fand im Labor statt, bei der die Funktionalitäten des dezentralen Steuerungssystems gegenüber der vorher im Labor des Lehrstuhls für Produktentstehung am Heinz Nixdorf Institut eingesetzten zentralen Produktionssteuerung validiert wurde. Für die Validierung wurde zuerst das dezentrale Steuerungssystem in einer

Laborumgebung implementiert. Dafür wurde ein Produktionssystem in einem Labor zu einem CPPS umgebaut. Hierbei wurde jede Arbeitsstation mit einem Computersystem ausgestattet. Das Computersystem beschreibt hierbei das System, in dem das in dieser Arbeit entwickelte Steuerungssystem installiert wurde. Dadurch können die Arbeitsstationen untereinander kommunizieren und dezentral gesteuert werden. Für die Implementierung der digitalen Repräsentation wurde dem Produktionssystem ein weiteres Element hinzugefügt, das über eine GUI verfügt, sodass Beschäftigten mit der digitalen Repräsentation kommunizieren können. Damit wird eine automatisierte Koordination unter allen Produktionselementen ermöglicht.

In der zweiten Stufe wurde das Produktionssystem in einer Simulationsumgebung nachgebildet, um zusätzlich das entwickelte Produktionssteuerungsverfahren mit weiteren Verfahren zu vergleichen. Dabei geht es um den Abgleich der Funktionalitäten zwischen dem dezentralen Steuerungssystem und anderen heutzutage in der Industrie üblicherweise verwendeten Steuerungsverfahren. Hierbei konnte nachgewiesen werden, dass durch den Einsatz der dezentralen Produktionssteuerung die Termintreue deutlich gesteigert werden konnte und sich das Produktionssystem mit Hilfe der Produktionssteuerung selbst organisieren kann.

Die in dieser Arbeit entwickelte automatisierte dezentrale Produktionssteuerung ist ein erster Ansatz für eine automatisierte Steuerung eines CPPS, welches selbst bei sich ändernden Randbedingungen und äußeren Ereignissen die Produktion optimal steuert. Die dezentrale Umsetzung des Produktionssteuerungssystems sorgt dabei für eine volle Umsetzbarkeit in CPPS und eine Übertragbarkeit sowie Skalierbarkeit auf andere Produktionssysteme. Die Entwicklung der digitalen Repräsentation sorgt darüber hinaus dafür, dass auch die Beschäftigten von dieser gewonnenen Flexibilität profitieren und gleichzeitig die Planung für diese erheblich verbessert wird.

Ausblick

Die Entwicklung der automatisierten dezentralen Produktionssteuerung wurde in sich vollständig behandelt. Die Anwendung fand hierbei jedoch nur in dem Smart Automation Labor und darüberhinausgehenden Simulationen statt. In weiteren Forschungsarbeiten könnte eine Übertragung auf andere Produktionssysteme stattfinden. Insbesondere würde die Übertragung hin zur industriellen Anwendung entscheidende Erkenntnisse hinsichtlich der Praxistauglichkeit bringen. Dies könnte die Grundlage für die Evaluation der automatisierten dezentralen Produktionssteuerung in einem industriellen Produktionsbetrieb sein.

Zudem wurde in dieser Arbeit die Schnittstelle zwischen bestehenden PPS-Systemen und der automatisierten dezentralen Produktionssteuerung wesentlich vereinfacht und größtenteils nur simuliert. Für eine industrielle Anwendung müsste eine entsprechende Integration mitsamt Programmierung der Schnittstellen vorgenommen werden.

Die Planungsgenauigkeit der Systematik könnte zudem durch eine bessere Verwertung von Vergangenheitsdaten erhöht werden. Anstelle des derzeit angewendeten Durchschnitts könnten hierbei stochastische Verfahren oder Prognosen mit Hilfe von künstlicher Intelligenz angewendet werden. Darüber hinaus wurden in den durchgeführten Untersuchungen die Verfahrensparameter zu Beginn festgelegt und im Untersuchungszeitraum nicht mehr verändert. In weiteren Untersuchungen könnte eine Variation der Parameter untersucht werden.

Die derzeit umgesetzte Lösung der digitalen Repräsentation ist noch im Prototypenstadium und bedarf zahlreicher Anpassungen. Die derzeitige Umsetzung als Python-Anwendung auf Minicomputern könnte hierbei in eine vollständig mobile Anwendung überführt werden. Derzeit wird sich zudem auf den Zuweisungsprozess von Beschäftigten zu Arbeitsstationen fokussiert. Hierbei könnte eine Erweiterung z.B. zur Planung der Anwesenheit durchgeführt werden. Bevor eine industrielle Übertragbarkeit der digitalen Repräsentation der Beschäftigten stattfinden kann, müsste generell die Anwendung von automatisierten Planungssystemen für Beschäftigte von verschiedenen Perspektiven untersucht werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Aca12] Acatech (2012), Cyber-Physical Systems: Innovationsmotoren für Mobilität, Gesundheit, Energie und Produktion, Springer, Dordrecht, 2012.

[Alf04] H.K. Alfares (2004): Survey, categorization, and comparison of recent tour scheduling literature. Annals of Operations Research 127 (2004) S. 145–175.

[ALW06] Abele, E., Liebeck, T., Wörn, A. (2006): Measuring Flexibility in Investment Decisions for Manufacturing Systems. CIRP Annals - Manufacturing Technology 55 (2006) 1, S. 433-436.

[AR11] Abele, E.; Reinhart, G. (2011): Zukunft der Produktion. Herausforderungen, Forschungsfelder, Chancen. München: Carl Hanser 2011. ISBN: 9783446425958

[Bak76] K.R. Baker (1976): Workforce allocation in cyclical scheduling problems: a survey, Operational Research Quarterly 27 (1976) 155–167.

[Bau13] Bauernhansl, T. (2013). Industrie 4.0: Nur ein Medienhype oder die schöne neue Produktionswelt? ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, 108, 573–574.

[BB07] Berthel, J., Becker, F.G. (2007): Personal-Management. 8. Auflage. Stuttgart 2007, S. 167.

[BBB+13] Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L. (2006): Personnel scheduling: A literature review. In: European Journal of Operational Research 226 (2013) 367–385

[BBS91] Bechtold, S.E., Brusco, M.J. , Showalter, M.J. (1991): A comparative-evaluation of labor tour scheduling methods, Decision Sciences 22 (1991) 683–699.

[Beg05] Begemann, C. (2005): Terminorientierte Kapazitätssteuerung in der Fertigung. Diss. Universität Hannover. Hannover: PZH Produktionstechnisches Zentrum GmbH 2005 (Berichte aus dem IFA, Band 02/2005).

[Ber15] Berlak, J. (2015): Manufacturing-Execution-System (MES) - Umsetzung in der dynamisch-flexiblen Produktion. In: Dickmann, P. (Hrsg.): Schlanker Materialfluss mit Lean Production, Kanban und Innovationen. Heidelberg: Springer, 2015, S. 234-246.

[BFH+16] Bonvoisin, J., Halstenberg, F., Buchert, T., Stark, R. (2016): A systematic literature review on modular product design, Journal of Engineering Design, 27:7, p.488-514, DOI: 10.1080/09544828.2016.1166482

[BGP+16] Barbian, M., Gräßler, I., Piller, F., Gülpen, C., Welp, P., Kamal, H., et al. (2016): Statusreport Digitale Chancen und Bedrohungen – Geschäftsmodelle für Industrie 4.0.

[BH16] Botthof, A., Hartmann, E. (2016): Zukunft der Arbeit in Industrie 4.0 – Neue Perspektiven und offene Fragen. In: Botthof, Alfons; Hartmann: Zukunft der Arbeit in Industrie 4.0. Springer Vieweg 2016. DOI 10.1007/978-3-662-45915-7

[BLG17] Biffl, S., Lüder, A., Gerhard, D. (2017): Multi-Disciplinary Engineering for Cyber-Physical Production Systems: Data Models and Software Solutions for Handling Complex Engineering Projects. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-56345-9.

[BMD+16] Christian Block, Friedrich Morlock, Thomas Dorka, Bernd Kuhlenkötter (2016): A Human Centered Multi-Agent-System for Production Planning and Control. In: Applied Mechanics and Materials. ISSN: 1662-7482, Vol. 840, pp 132-139

[BQB10] Brucker, P., Qu, R., Burke, E.(2011): Personnel scheduling: Models and complexity. In: European Journal of Operational Research 210 (2011) 467–473

[Bro10] Broy, M. (Hrsg.). (2010): Cyber-physical Systems – Innovation durch softwareintensive eingebettete Systeme. Heidelberg: Springer.

[BTK+12] Busch, F., Thomas, C., Kuhlenkötter, B., Deuse, J. (2012). A hybrid human–robot assistance system for welding operations methods to ensure process quality and forecast ergonomic conditions. In S. J. Hu (Hrsg.), Proceedings of the 4th CIRP conference on assembly technologies and systems (CATS): Technologies and systems for assembly quality, productivity and customization (S. 151–154).

[BW07] Böse, F.; Windt, K. (2007): Catalogue of Criteria for Autonomous Control in Logistics. In: Hülsmann, M; Windt, K: Understanding autonomous cooperation and control in logistics: the impact of autonmy on management, information, communication and material flow. Springer Science & Business Media, 2007

[CD06] Caramia, M.; Dell'Olmo, P. (2006): Effective resource management in manufacturing systems. Optimization algorithms for production planning. London, New York: Springer 2006. ISBN: 1846280052

[CDY+16] Campos Ciro, G., Dugardin, F., Yalaoui F., Kelly, R. (2016): Open shop scheduling problem with a multi-skills resource constraint: a genetic algorithm and an ant colony optimisation approach. In: International Journal of Production Research, 2016, Vol. 54, No. 16, 4854–4881

[CHM15] Claus, T. Hermann, F., Manitz, M.(2015): Produktionsplanung und – steuerung - Forschungsansätze, Methoden und deren Anwendungen. Springer-Verlag Berlin Heidelberg 2015

[Chr94] Christensen, J.H (1994): Holonic Manufacturing systems: Initial Architecture and Standards Directions. In: Holonic Manufacturing - a Promising new Technology for Manufacturing in the 21st Century; European Conference on Holonic Manufacturing Systems, 1, 1994, pp.1-19

[DAA17] Delgoshaei, A., Khairol M., Ariffin, A., Ali, A. (2017): A multiperiod scheduling method for trading-off between skilled-workers allocation and outsource service usage in dynamic CMS, International Journal of Production Research, 55:4, 997-1039, DOI: 10.1080/00207543.2016.1213445

[Din17] Dinkelmann, M.(2017): Methode zur Unterstützung der Mitarbeiterpartizipation im Change Management der variantenreichen Serienproduktion durch Lernfabriken. Dissertationsschrift ISBN: 978-3-8396-1113-5. Verlag: Fraunhofer Verlag

[DD07] Domschke, W.; Drexl, A. (2007): Einführung in Operations Research. Mit 63 Tabellen. 7. Aufl. Berlin: Springer 2007. ISBN: 3540709487.

[DDW+16] Denkena, B., Dittrich, M.A., Winter, F., Wagener, C.(2016): Simulation-based planning and evaluation of personnel scheduling in knowledge-intensive production systems. In: Prod. Eng. Res. Devel. (2016) 10:489–496

[DEE+12] Deuse, J., Eigner, M., Erohin, O., Krebs, M., Schallow, J., Schäfer, P. (2011). Intelligente Nutzung von implizitem Planungswissen der Digitalen Fabrik. ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, 106, 433–437.

[DGR+13] Wu, D., John Greer, M., Rosen, D.W., Schaefer, D. (2013): Cloud manufacturing: Strategic vision and state-of-the-art. Journal of Manufacturing Systems 32 (4): 564-579.

[DLS12] Denkena, B.; Lorenzen, L.-E.; Schmidt, J.(2012): Adaptive process planning. Production Engineering Research and Development (2012) 6, S. 55-67. Berlin: Springer 2012.

[Dru08] Drumm (2008): Personalwirtschaft. 6. Auflage. Berlin/ Heidelberg 2008, S. 197 ff.

[DSS09] Deuse, J., Schallow, J., Sackermann, R. (2009). Arbeitsgestaltung und Produktivität im globalen Wettbewerb. Tagungsband zum 55. Frühjahrskongress „Arbeit, Beschäftigungsfähigkeit und Produktivität im 21. Jahrhundert“. 04.03.–06.03.2009 der Gesellschaft für Arbeitswissenschaft e.V. (GfA). Technische Universität Dortmund, S. 19–23.

[DS10] Dyckhoff, H., Spengler, T. (2010). Produktionswirtschaft. Heidelberg: Springer., S. 29f.

[Egb13] Egbers, J.-F. (2013): Identifikation und Adaption von Arbeitsplätzen für leistungsgewandelte - Mitarbeiter entlang des Montageplanungsprozesses. Diss. Technische Universität München (2013). München: Utz 2013.

[EJK+04a] Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D. (2004): Staff scheduling and rostering: a review of applications, methods and models, European Journal of Operational Research 153 (2004) 3–27.

[EJK+04b] Ernst, A.T., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D. (2004):An annotated bibliography of personnel scheduling and rostering, Annals of Operations Research 127 (2004) 21–144.

[EMM17] Esmaeili, A., Mozayani, N., Motlagh, M., Matson, E. (2017): A socially-based distributed self-organizing algorithm for holonic multi-agent systems: Case study in a task environment. In: Cognitive Systems Research 43 (2017) 21–44

[Eng15] Engelhardt, P. (2015): System für die RFID-gestützte situationsbasierte Produktionssteuerung in der auftragsbezogenen Fertigung und Montage. Diss. Technische Universität München 2015. Forschungsberichte IWB

[Eng20] Engels, G. (2020): Der digitale Fußabdruck, Schatten oder Zwilling von Maschinen und Menschen. Gruppe. Interaktion. Organisation. Zeitschrift für Angewandte Organisationspsychologie (GIO), Vol. 51, 3, Springer Fachmedien Wiesbaden , pp. 363 – 370

[FHB08] Fletcher, S.-R.; Baines, T.-S.; Harrison, D.-K.(2008): An investigation of production workers' performance variations and the potential impact of attitudes. International Journal of Advanced Manufacturing Technology (2008) 35, S. 1113-1123. London: Springer 2008.

[FO13] Frey, C. B., Osborne, M. A. (2013). The future of employment - How susceptible are jobs to computerisation (p. 77). Oxford, England: Oxford Martin Programme on Technology & Employment.

[Fra07] Frank, U. (2007): Ein Vorschlag zur Konfiguration von Forschungsmethoden in der Wirtschaftsinformatik. In: Stephan Zelewski, F.L.: Wissenschaftstheoretische Fundierung und wissenschaftliche Orientierung der Wirtschaftsinformatik, GITO Verlag Berlin 2007, S. 155-184

[FS99] Frieling, E.; Sonntag, K. (1999): Lehrbuch Arbeitspsychologie. 2., vollständig überarbeitete und erweiterte Auflage. Bern: Hans Huber 1999. ISBN: 3-456-82932-9.

[GBS13] Göpfert, I.; Braun, D.; Schulz, M. (Hrsg.) (2013): Automobillogistik – Stand und Zukunftstrends. 2., aktualisierte und erweiterte Auflage. Wiesbaden: Springer 2013. ISBN: 978-3-658-01582-4.

[GGH+13] Ganschar, O., Gerlach, S., Hämmerle, M., Krause, T., Schlund, S. (2013). Produktionsarbeit der Zukunft-Industrie 4.0 (pp. 50-56). D. Spath (Ed.). Stuttgart: Fraunhofer Verlag.

[GGN17] Eric H. Grosse, Christoph H. Glock, W. Patrick Neumann (2017): Human factors in order picking: a content analysis of the literature, International Journal of Production Research, 55:5, 1260-1276, DOI: 10.1080/00207543.2016.1186296

[GHB18] Gräßler, I., Hentze, J., Bruckmann, T. (2018): V-Models for Interdisciplinary Systems Engineering. In: Dorian Marjanović, Mario Storga, Neven Pavkovic, Neven Bojcetic und Stanko Škec (Hg.): Proceedings of the DESIGN 2018 15th International Design Conference. DESIGN Conference. Dubrovnik, 21.-24.05.2018. Design Society, S. 747–756.

[GH20] Gräßer, I., Hentze, J. (2020): The new V-Model of VDI 2206 and its validation. In: at - Automatisierungstechnik 68 (5), S. 312–324. DOI: 10.1515/auto-2020-0015.

[Glo14] Glonegger, M. (2014): Berücksichtigung menschlicher Leistungsschwankungen bei der Planung von Variantenfließmontagesystemen. Diss. Technische Universität München 2014. Forschungsberichte IWB

[GOS+13] Glonegger, M.; Ottmann, W.; Schadl, M.; Distel, D. (2013): Identification of production rhythms in synchronized assembly lines by recording and evaluating current processing times. Advanced Materials Research Vol. 769 pp. 350-358.

[Grä04] Gräßler, I. (2004): Kundenindividuelle Massenproduktion: Entwicklung, Vorbereitung der Herstellung, Veränderungsmanagement. Springer Verlag 2004.

[Grä15] Gräßler, I. (2015): Umsetzungsorientierte Synthese mechatronischer Referenzmodelle. Implementation-oriented synthesis of mechatronic reference models. In: T. Bertram (Hg.): Konferenzband der VDI Mechatronik. Fachtagung Mechatronik 2015. Dortmund, S. 167–172.

[Gro13] N. Gronau (2013): Bionic Manufacturing: Steuerung der Fabrik mit Mitteln der Natur im Zeitalter von Industrie 4.0. Industrie Management 29 (6): 12--16 (2013)

[Gro14] Gronau, N.: Der Einfluss von Cyber-Physical Systems auf die Gestaltung von Produktionssystemen. In Kersten, W.; Koller, H.; Lädding, H.: Industrie

4.0 – Wie intelligente Vernetzung und kognitive Systmee unsere Arbeit verändern. Schriftenreihe der Hochschulgruppe für Arbeits- und Betriebsorganisation e.V. GITO Verlag Berlin, 2014, S. 279-295

[GP17] Gräßler, I; Pöhler, A. (2017): Integration of a digital twin as human representation in a scheduling procedure of a cyber-physical production system. In: Proceedings of 2017 IEEE International. Singapore, 10.-13.12.2017

[GP18] Gräßler, I; Pöhler, A. (2018): Intelligent Devices in a Decentralized Production System Concept. In: Proceedings of 11th CIRP Conference on Intelligent Computation in Manufacturing Engineering, Ischia, 19. - 21. Jul. 2017 CIRP (Centre for International Research in Production), Procedia CIRP, 2017.

[GPH17] Gräßler, I; Pöhler, A.; Hentze, J. (2017): Decoupling of product and production development in flexible production environments. In: Complex Systems Engineering and Development Proceedings of the 27th CIRP Design Conference Cranfield University, UK, CIRP (Centre for International Research in Production), 2017, S. 548–553

[GPP16] Gräßler, I; Pöhler, A.; Pottebaum, J. (2016): Creation of a Learning Factory for Cyber Physical Production System. In: Proceedings of the 6th CIRP Conference on Learning Factories, Procedia CIRP, Volume 54, 2016, Pages 107-112

[Gri12] Grinninger, J. (2012): Schlanke Produktionssteuerung zur Stabilisierung von Auftragsfolgen in der Automobilproduktion, Dissertation TU München

[Gru17] Grundstein, S.: Fertigungsregelung flexibler Fließfertigungen und Werkstattfertigungen zur Einhaltung von Lieferterminen Integration von Auftragsfreigabe, Arbeitsverteilung, Reihenfolgebildung und Kapazitätssteuerung unter Nutzung des Potenzials Cyber-Physischer Produktionssysteme. Dissertation, GRIN Verlag, 2017

[GSS+15] Grundstein, S.; Schukraft, S.; Scholz-Reiter, B., Freitag, M. (2015). Coupling order release methods with autonomous control methods - state of the art and potentials. International Journal of Production Management and Engineering, 3(1), S.43-56.

[GTY16] Gräßler, I; Taplick, P.; Yang, X. (2016): Educational Learning Factory of a Holistic Product Creation Process. In: Proceedings of the 6th CIRP Conference on Learning Factories, Procedia CIRP, Volume 54, 2016, S. 141-146.

[GY16] Gräßler, I.; Yang, X. (2016): Interdisciplinary Development of Production Systems Using Systems Engineering. In: Procedia CIRP 50, S. 653–658. DOI: 10.1016/j.procir.2016.05.008.

[Hac89] Hackstein, R. (1989). Produktionsplanung und -steuerung (PPS): Ein Handbuch für die Betriebspraxis. Düsseldorf:

[Har11] Hartwich, E. (2011): Grundlagen Change Management: Organisationen strategisch ausrichten und zur Exzellenz führen (Schriftenreihe der Führungsakademie Baden-Württemberg). Richard Boorberg Verlag; 1. Edition (9. Februar 2011)

[HCC+16] He, J., Chen, X., Chen, X., Liu, Q.(2016): Distributed production planning based on ATC and MOILP considering different coordination patterns. In: Journal of Intelligent Manufacturing (2016) 27:1067–1084

[Hee17] Hees, A.(2017): System zur Produktionsplanung für rekonfigurierbare Produktionssysteme. Technische Universität München 2017. Forschungsberichte IWB

[HGF+16] Hecklau, F., Galeitzke, M., Flachs, S., Kohl, H. (2016): Holistic approach for human resource management in Industry 4.0. In: 6th CLF - 6th CIRP Conference on Learning Factories. Procedia CIRP 54 (2016) 1 – 6

[HHL18] Hochdörffer, J., Hedler, M., Lanza, G. (2018): Staff scheduling in job rotation environments considering ergonomic aspects and preservation of qualifications. In: Journal of Manufacturing Systems 46 (2018) 103–114

[HRT+19] Hennig, M., Reisinger, G., Trautner, T., Hold, P., Gerhard, D., Mazak, A. (2019): TU Wien Pilot Factory Industry 4.0. Procedia Manufacturing, Volume 31, p. 200-205. DOI: 10.1016/j.promfg.2019.03.032

[IO94] Iwata, K., Onosato, M. (1994). Random manufacturing system: A new concept of manufacturing systems for production to order. Proceedings of the CIRP, 43(1), S. 379-384.

[Ise16] Isenberg, M. (2016): Produktionsplanung und -steuerung in mehrstufigen Batchproduktionen - Methoden der Loskomposition und -terminierung bei stufenspezifischen Auftragsfamilien. Dissertation Universität Bremen

[JRJ+13] Jentsch, D., Riedel, R., Jäntsch, A., Müller, E. (2013). Fabrikaudit Industrie 4.0; Strategischer Ansatz zur Potentialermittlung und schrittweisen Einführung einer Smart Factory. ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, 108, 678–681.

[JBP+13] Jana, T.K., Bairagi, B., Paul, S., Sarkar, B., Saha, J. (2013): Dynamic schedule execution in an agent based holonic manufacturing system Journal of Manufacturing Systems 32 (2013) 801– 816

[JBP+14] Jana, T.K., Bairagi, B., Paul, S., Sarkar, B., Saha, J. (2014): Multi-objective scheduling in an agent based Holonic manufacturing system. In: Decision Science Letters 3 (2014) 1–16

[JKB03] Jovane, F., Koren, F., Boer., C.V. (2003): Present and Future of Flexible Automattion: Towards New paradigms. Annals of the CIRP 52/2 (1): 543-557.

[Jun09] Jungwattanakita, J.; Reodechaa, M.; Chaovalitwongsea, P. Werner, F. (2009). A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. Computers & Operations Research, 36, S. 358-378.

[JYE19a] Josifovska, K., Yigitbas, E., Engels, G. (2019): Reference framework for digital twins within cyber-physical systems. 2019 IEEE/ACM 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS). pp. 25-31

[JYE19b] Josifovska, K., Yigitbas, E., Engels, G. (2019): A digital twin-based multi-modal ui adaptation framework for assistance systems in industry 4.0. International Conference on Human-Computer Interaction 2019. Pp. 398-409

[Kag15] Kagermann, H. (2015): Change Through Digitization—Value Creation in the Age of Industry 4.0. In H. Albach, H. Meffert, A. Pinkwart, & R. Reichwald (Eds.), Management of Permanent Change (pp. 23–45). Wiesbaden: Springer Fachmedien Wiesbaden.

[KHJ99+] Koren, Y., U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, Van Brussel, H. (1999): Reconfigurable Manufacturing Systems. CIRP Annals 48 (2): 527-540.

[Kle06] Kletti, J. (2006): MES - Manufacturing Execution System: Moderne Informationstechnologie zur Prozessfähigkeit der Wertschöpfung. Berlin: Springer 2006. ISBN: 3540280103.

[KLW11] Kagermann, H.; Lukas, W.-D., Wahlster, W. (2011): Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. VDI Nachrichten, 2011

[KMN15] Kuprat, T.; Mayer, J., Nyhuis, P. (2015). Aufgaben der Produktionsplanung im Kontext von Industrie 4.0. Industrie Management, 31(2), S. 11-14.

[Kra00] Kratzsch, S. (2000): Prozess- und Arbeitsorganisation in Fließmontagesystemen. Diss. Technische Universität Carolo-Wilhelmina zu Braunschweig (2000). Essen: Vulkan 2000. ISBN: 3-8027-8654-8. (Schriftenreihe des IWF).

[KS04] Klein, R., Scholl, A. (2004): Planung und Entscheidung. Konzepte, Modelle und Methoden einer modernen betriebswirtschaftlichen Entscheidungsanalyse. München: Vahlen 2004. ISBN: 3800630605.

[Kur05] Kurbel, K. (2005). Produktionsplanung und -steuerung im Enterprise Resource Planning und Supply Chain Management. München: Oldenbourg Wissenschaftsverlag.

[Kuy13] Kuyumku, A. (2013). Modellierung der Termintreue in der Produktion. Dissertation, Technische Universität Hamburg-Harburg.

[KWH13] Kagermann, H., Wahlster, W., Helbig, J. (2013). Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0 – Abschlussbericht des Arbeitskreises Industrie 4.0. Berlin: Forschungsunion im Stifterverband für die Deutsche Wissenschaft.

[LE99] Luczak, H., Eversheim, W. (1999). Produktionsplanung und -steuerung. Grundlagen, Gestaltung und Konzepte. Berlin: Springer.

[Lee08] Lee, E.A. (2008): "Cyber Physical Systems: Design Challenges", Technical Report No. UCB/EECS-2008-8; http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html

[Lei09] Leitao, P. (2009): Agent-based distributed manufacturing control: A state-of-the-art survey. In: Engineering Applications of Artificial Intelligence 22 (2009) 979–991

[LFB16] Lang, F., Fink, A., Brandt, T. (2016): Design of automated negotiation mechanisms for decentralized heterogeneous machine scheduling. In: European Journal of Operational Research 248 (2016) 192–2016

[Löd01] Lödding, H. (2001): Dezentrale Bestandsorientierte Fertigungsregelung. Diss. Universität Hannover. Düsseldorf: VDI-Verlag 2001 (Fortschritt-Berichte VDI: Reihe 2, Nr. 587).

[Löd08] Lödding, H. (2008): Verfahren der Fertigungssteuerung. Grundlagen, Beschreibung, Konfiguration. Berlin, Heidelberg: Springer 2008. ISBN: 3540202323.

[Löd13] Lödding, H. (2013). Handbook of manufacturing control. fundamentals, description, configuration. Berlin: Springer.

[Lop05] Lopitzsch, J. R. (2005): Segmentierte adaptive Fertigungssteuerung. Diss. Universität Hannover. Garbsen: PZH Produktionstechnisches Zentrum GmbH 2005 (Berichte aus dem IFA, Band 03/2005).

[LR14] Laßmann, N.; Rupp, R. (2014): Personalplanung Produkt-Nr. 23848-38431 Herausgeber: IG Metall Vorstand, FB Betriebs- und Branchenpolitik. Ausgabe: Juli 2014 Druckvorstufe

[Mei09] Meißner, S. (2009). Logistische Stabilität in der automobilen Variantenfließfertigung. Dissertation, Technische Universität München.

[Mei17] Meinecke, C. (2017): Ein Lösungsverfahren für die integrierte Planung der Produktion in der Werkstattfertigung und den überbetrieblichen Transport. Dissertation Universität Bremen.

[MNS19] Ma, A., Nassehi, A., Snider, C. (2019). Anarchic manufacturing. In: International Journal of Production Research, Taylor & Francis, volume 57, 8, S. 2514-2530, 2019

[MSK+16] Mishra, N., Singh, A., Kumari, A., Govindan, K., Ali, S. (2016) Cloud-based multi-agent architecture for effective planning and scheduling of distributed manufacturing, International Journal of Production Research, 54:23, 7115-7128, DOI: 10.1080/00207543.2016.1165359

[MUK00] Mehrabi, M. G., A. G. Ulsoy, Koren., Y. (2000): Reconfigurable manufacturing systems: key to future manufacturing. Journal of Intelligent Manufacturing 11 (4): 403-419.

[Mül92] Müller-Merbach, H. (1992): Operations-Research. Methoden und Modelle der Optimalplanung. 3. Aufl. München: Vahlen 1992.

[MPM17] Meudt, T., Pohl, M., Metternich, J. (2017): Die Automatisierungspyramide - Ein Literaturüberblick. http://tuprints.ulb.tu-darmstadt.de/id/eprint/6298

[MVD+15] Monostori, L., Valckenaers, P., Dolgui, D., Panetto, H., Brdys, M., Csáji, B. (2015): Cooperative control in production and logistics. In: Annual Reviews in Control 39 (2015) 12–29

[MVK06] Monostori, L.; Váncza, J., Kumara, S. R. T. (2006). Agent-based systems for manufacturing. Annals of the CIRP, 55(2), S. 697-720.

[Neb07] Nebl, T. (2007): Produktionswirtschaft. 6. Auflage. München: Oldenbourg Wissenschaftsverlag GmbH 2007. ISBN: 978-3-486-5849 3-6., S.600ff.

[NH08] Nyhuis, P. & Wiendahl, H.-P. (2008). Fundamentals of production logistics: Theory, tools and applications. Berlin: Springer.

[NM02] Neumann, K.; Morlock, M. (2002): Operations-Research. 2. Aufl. München: Carl Hanser 2002. ISBN: 3446221409.

[ÖBF+10] Österle, H.; Becker, J.; Frank, U.; Hees, T.; Karagiannis, D.; Krcmar, H.; Loos, P.; Mertens, P.; Oberweis, A.; Sinz, E.J. (2010): Memorandum zur gestaltungsorientierten Wirtschaftsinformatik. In: Zeitschrift für wissenschaftlichen Fabrikbetrieb 62 (2010), Nr. 6, S. 664 ff.

[Oec06] Oechsler, W.A.(2006): Personal und Arbeit. 8. Auflage. München/ Wien 2006, S. 160 f. J.

[Ost12] Ostgathe, M. (2012): System zur produktbasierten Steuerung von Abläufen in der auftragsbezogenen Fertigung und Montage. Diss. Technische

Universität München. München: Utz 2012. (Forschungsberichte iwb, Nr. 265).

[PBG+14] Petersen, M., Bandak, S., Gausemeier, J., Iwanek, P., Schneider, M. (2014): Methodik zur Berücksichtigung von Wechselwirkungen zwischen Produkt und Produktionssystem in den Frühen Phasen der Produktentwicklung – Ein Praxisbeispiel. 17 IFF-Wissenschaftstage 2014, S.13-21

[PBZ+12] Pach, C.; Bekrar, A.; Zbib, N.; Sallez, Y. & Trentesaux, D. (2012). An effective potential field approach to FMS holonic heterarchical control. Control Engineering Practice, 20(12), S. 1293-1309.

[Pil06] Piller, F. T. (2006): Mass Customization: Ein wettbewerbsstrategisches Konzept im Informationszeitalter. 4. Aufl. Wiesbaden: Deutscher Universitäts-Verlag 2006. ISBN: 9783835003552.

[Pin09] Pinedo, M.L.(2009): Planning and Scheduling in Manufacturing and Services. In: Springer Science+Business Media (2009). ISBN 978-1-4419-0909-1

[Pin16] Pinedo, M.L.(2016): Scheduling: Theory, Algorithms, and Systems. Springer, 2016

[PMD+14] Pantförder, D.; Mayer, F.; Diedrich, C.; Göhner, P.; Weyrich, M.; Vogel-Heuser, B. (2014): Agentenbasierte dynamische Rekonfiguration von vernetzten intelligenten Produktionsanlagen – Evolution statt Revolution. In: Bauernhansl, T., Ten Hompel, M.; Vogel-Heuser, B.: Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung, Technologie, Migration. Springer-Verlag Bering Heidelberg, 2014, S. 145-158

[PMT09] Pannequin, R.; Morel, G., Thomas, A. (2009). The performance of product-driven manufacturing control: An emulation-based benchmarking study. Computers in Industry, 60(3), S. 195-203.

[PS06] Pfeifer, T.; Schmitt, R. (2006): Autonome Produktionszellen: komplexe Produktionsprozesse flexibel automatisieren. Springer-Verlag, 2006

[PT12] Park, H., Tran, N. (2012). An autonomous manufacturing system based on swarm of cognitive agents. Journal of Manufacturing Systems, 31(3), S. 337-348.

[PTR+07] Peffers, K.; Tuuanen, T.; Rothenberger, M.A.; Chatterjee, S. (2007): A Design Science Research Methodology for Information Systems Research. In: Journal of Information Management Systems 24 (2007). Nr. 3. S. 45-78

[QML19] Qu, Y.J., Ming, X.G., Liu, Z.W. (2019). Smart manufacturing systems: state of the art and future trends. Int J Adv Manuf Technol 103, 3751–3768 (2019). https://doi.org/10.1007/s00170-019-03754-7

[Rai10] Raileanu, S. (2010): Production scheduling in a holonic manufacturing system using the open-control concept. In: U.P.B. Sci. Bull., Series C, Vol. 72, Iss. 3, 2010

[REFA93] REFA Verband für Arbeitsstudien e. V. (Hrsg.) (1993): Methodenlehre der Betriebsorganisation – Lexikon der Betriebsorganisation. München: Hanser 1993. ISBN: 3-446-17523-7.

[RGE+11] Reinhart, G., Glonegger, M.; Egbers, J.; Schilp, J., Göritz, A.; Weikamp, J. (2011): Taktzeitadaption unter Berücksichtigung der zirkadianen Rhythmik – Analyse unterschiedlicher Taktzeit-Szenarien zur Belastungsreduktion von Montagemitarbeitern. wt Werkstattstechnik online, 101 (2011), H.9, S. 595-599.

[REG+13] Reinhart, G.; Engelhardt, P.; Geiger, F.; Philipp, T. R.; Wahlster, W.; Zühlke, D.; Schlick, J.; Becker, T.; Löckelt, M.; Pirvu, B.; Stephan, P.; Hodek, S.; Scholz-Reiter, B.; Thoben, K.; Gorldt, C.; Hribernik, K. A.; Lappe, D.; Veigt, M. (2013) “Cyber-physische Produktionssysteme: Produktivitäts- und Flexibilitätssteigerung durch die Vernetzung intelligenter Systeme in der Fabrik“, wt Werkstatttechnik online, 103 (2), 2013, pp. 84-89.

[Rei09] Reißenweber, B. (2009): Feldbusse zur industriellen Kommunikation. Verlag Oldenbourg, München 2009

[Rei17] Reinhart, G. (2017): Handbuch Industrie 4.0. Geschäftsmodelle, Prozesse, Technik. Carl Hanser Verlag 2017

[RZ03] Reinhart, G. (2003); Zäh, M.: Marktchance Individualisierung. Berlin: Springer 2003. ISBN: 9783540005940.

[Sch06] Schuh, G. (2006): Produktionsplanung und -steuerung: Grundlagen, Gestaltung und Konzepte. 3., völlig überarbeitete Auflage. Berlin, Heidelberg : Springer, 2006. – 888 S. – ISBN 354040306X

[Sch12] Schuh, G.; Stich, V. (2012): Produktionsplanung und -steuerung 1. Grundlagen der PPS. 4. Aufl. Berlin: Springer 2012. ISBN: 9783642254222.

[Sch13a] Schuh, G. & Stich, V. (Hrsg.) (2013). Produktion am Standort Deutschland. Ergebnisse der Untersuchung 2013. Aachen.

[Sch13b] Schnell, G. (2013): Bussysteme in der Automatisierungs- und Prozesstechnik. Grundlagen und Systeme der industriellen Kommunikation. Vieweg+Teubner Verlag, 2013

[SD19] Stark, R. und Damerau, T. (2019): Digital Twin. In: Chatti, S. und Tolio, T. [Hrsg.] CIRP Encyclopedia of Production Engineering. [S.l.]: Springer

Berlin Heidelberg. DOI: https://doi.org/10.1007/978-3-642-35950-7_16870-1

[SDF+08] Scholz-Reiter, B.; de Beer, C.; Freitag, M., Jagalski, T. (2008). Bio-inspired and pheromone-based shop-floor control. International Journal of Computer Integrated Manufacturing, 21(2), S. 201-205.

[Sei06] Seibold, J. (2006): Hybride Fertigungssteuerung in der Großserienproduktion. Diss. Universität Hannover. Garbsen: PZH Produktionstechnisches Zentrum GmbH 2006 (Berichte aus dem IFA, Band 02/2006).

[SFL19] Stark, R., Fresemann, C., Lindow K. (2019): Development and operation of Digital Twins for technical systems and services. CIRP Annals. Manufacturing Technology 68 (2019), No.1, pp.129-132. ISSN: 0007-8506

[SG16] Siepmann D., Graef N. (2016) Industrie 4.0 – Grundlagen und Gesamtzusammenhang. In: Roth A. (eds) Einführung und Umsetzung von Industrie 4.0. Springer Gabler, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-48505-7_2

[SGG+13] Dieter Spath; Oliver Ganschar; Stefan Gerlach; Moritz Hämmerle; Tobias Krause; Sebastian Schlund (2013): Produktionsarbeit der Zukunft – Industrie 4.0. Fraunhofer Verlag, Stuttgart 2013

[SGP09] Scholz-Reiter, B.; Görges, M., Philipp, T. (2009). Autonomously controlled production systems - influence of autonomous control level on logistic performance. CIRP Annals - Manufacturing Technology, 58, S. 395-398.

[SHW89] Spearman, M. L.; Hopp, W. J.; Woodruff, D. L. (1989): A Hierachical Control Architecture for Constant Work in Process (Conwip) Production Systems. International Journal of Manufacturing and Operations Management 2 (1989) 3, S. 147-171.

[Sie10] Siemoneit, O. (2010): Eine Wissenschaftstheorie der Betriebswirtschaftslehre: Wissensformen, Erkenntnismethoden und Forschungskonzeptionen einer verwissenschaftlichten Techniklehre. Stuttgart, Universität Stuttgart. 2010, S. 45f.

[SJB08] Scholz-Reiter, B.; Jagalski, T., Bendul, J. (2008). Autonomous control of a shop floor based on bee’s foraging behaviour. In: Haasis, H.-D.; Kreowski, H.-J. & Scholz-Reiter, B. (Hrsg.), First international conference on dynamics in logistics. LDIC 2007, S. 415-423. Berlin: Springer.

[SKN17] Stark, R., Kind, S., Neumeyer, S. (2017): Innovations in digital modelling for next generation manufacturing system design. Cirp Annals-manufacturing Technology 2017, volume 66, p. 169-172

[SLB09] Schlick, C.; Luczak, H.; Bruder, R. (2009): Arbeitswissenschaft, 3., vollständig überarbeitete und erweiterte Auflage. Heidelberg: Springer 2009. ISBN: 978-3-540-78332-9.

[SM13] Sudo, Y., Matsuda, M. (2013). Agent based manufacturing simulation for efficient assembly operations. Procedia CIRP, 7, S. 437-442.

[SRW+15] Scholz-Reiter, B., Reinhart, G., Wahlster, W., Wittenstein, M., Zühlke, D. (2015): Intelligente Vernetzung in der Fabrik.: Industrie 4.0 Umsetzungsbeispiele für die Praxis. Fraunhofer Verlag, 2015

[Stü12] Stürmann, A. (2012): Montagesynchrone Auftragssteuerung. Diss. Universität Hannover. Garbsen: PZH Produktionstechnisches Zentrum GmbH 2012 (Berichte aus dem IPH, Band 01/2012).

[SVR10] Schneider, H. M. ; Buzacott, J. A. ; Rücker, T. (2010): Operative Produktionsplanung und -steuerung: Konzepte und Modelle des Informations- und Materialflusses in komplexen Fertigungssystemen. München : Oldenbourg, 2010. – 247 S. – ISBN 978–3–486–70031–2, S.14f.

[SW06] Schuh, G.; Westkämper, E. (2006): Liefertreue im Maschinen- und Anlagenbau. Stand, Potenziale, Trends. Stuttgart: 2006. ISBN: 392669002X.

[SWB+12] Steinhilper, R.,Westermann, H., Butzer, S., Haumann, M., Seifert, S. (2012). Komplexitätmessbar machen; Eine Methodik zur Quantifizierung von Komplexitätstreibern und -wirkungen am Beispiel der Refabrikation. ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb, 107, 360–365.

[SZ15] Sabar, M., Zenjari, A. (2015): Impact of Flexibility and Employees Preferences on Shift Scheduling: A Simulation Approach. International Journal of Economics and Management Engineering Feb. 2015, Vol. 5 Iss. 1, PP. 1-8

[Tem13] Tempelmeier, H. (2013): Produktion und Logistik: Einflussgrößen der Leistung von Fließproduktionssystemen unter stochastischen Bedingungen.

[Thi11] Thiel, K. (2011): MES - Integriertes Produktionsmanagement. Leitfaden, Marktübersicht und Anwendungsbeispiele. Carl Hanser Verlag GmbH & Co. KG 2011

[THT15] Thomas, A., Borangiu, T., Trentesaux, D. (2015): Holonic and multi-agent technologies for service and computing oriented manufacturing. J Intell Manuf 12/2015. DOI 10.1007/s10845-015-1188-4

[Trz15] Trzyna, D. (2015): Modellierung und Steuerung von Eilaufträgen in der Produktion. Dissertation Technische Universität Hamburg-Harburg. Band 26 1. Auflage Hamburg 2015, ISSN 1613-8244

[TTV12] Thomas, T., Trentesaux, D., Valckenaers, P. (2012): Intelligent distributed production control. In: J Intell Manuf (2012) 23:2507–2512. DOI 10.1007/s10845-011-0601-x

[TWW17] Thoben, K., Wiesner, S., Wuest, T. (2017): "Industrie 4.0" and Smart Manufacturing - A Review of Research Issues and Application Examples, Int. J. Autom. Technol., 2017, volume 11, p.4-16

[Ued93] Ueda, K. (1993). A genetic approach toward future manufacturing systems. Procedia CIRP, S. 221-228.

[UH76] Ulrich, P., Hill, W. (1976): Wissenschaftstheoretische Grundlagen der Betriebswirtschaftslehre (Teil I). WiSt Wirtschaftswissenschaftliches Studium - Zeitschrift für Ausbildung und Hochschulkontakt 5(7), S. 304–309

[UHF+00] Ueda, K.; Hatono, I.; Fujii, N., Vaario, J. (2000). Reinforcement learning approaches to biological manufacturing systems. CIRP Annals - Manufacturing Technology, 49(1), S. 343-346.

[Ulr71] Ulrich, H. (1971). Der systemorientierte Ansatz in der Betriebswirtschaftslehre. Kortzfleisch, Gert von (Hrsg.). Wissenschaftsprogramm und Ausbildungsziele der Betriebswirtschaftslehre. 2. bis 5. Juni 1971. Berlin: Duncker & Humblot (Tagungsberichte des Verbandes der Hochschullehrer für Betriebswirtschaft e.V, 1). ISBN 3-428-02559-8, S. 43–60

[Ung98] Unger, U. J. (1998): Beitrag zur Produktionsplanung und -steuerung komplexer Produkte in wandlungsfähigen Unternehmensstrukturen. Diss. Technische Universität Braunschweig. 1. Aufl. Göttingen: Cuvillier 1998.

[VBC+20] A. Villalonga, G. Beruvides, F. Castaño and R. E. Haber, "Cloud-Based Industrial Cyber–Physical System for Data-Driven Reasoning: A Review and Use Case on an Industry 4.0 Pilot Line," in IEEE Transactions on Industrial Informatics, vol. 16, no. 9, pp. 5975-5984, Sept. 2020, doi: 10.1109/TII.2020.2971057.

[VDI92] Verein Deutscher Ingenieure (1992): Lexikon der Produktionsplanung und -steuerung: Begriffszusammenhänge und Begriffsdefinitionen. 4. Aufl. Düsseldorf: VDI-Verlag 1992. ISBN: 3184010066.

[VDI07] Verein Deutscher Ingenieure (2007): VDI 5600: Fertigungsmanagementsysteme – Manufacturing Execution Systems (MES). Beuth Verlag Berlin, 2007

[VDI20] Verein Deutscher Ingenieure (2020): VDI 2206:2020-09: Entwicklung Cyber-Physischer Mechatronischer Systeme (CPMS). Beuth Verlag Berlin, 2020 (Gründruck)

[VV15] Valckenaers, P., Van Brussel, H (2015). Design for the Unexpected: From Holonic Manufacturing Systems towards a Humane Mechatronics Society. Butterworth-Heinemann, 2015, pp. 41 – 127

[Wah11] Wahlster, W. (2011). Industrie 4.0: Vom Internet der Dinge zur vierten industriellen Revolution. In Innovationskongress Märkte, Technologien, Strategien, Zentralverband Elektrotechnik- und Elektronikindustrie e.V. (ZVEI).

[WDF+14] Weyrich, M., Diedrich, C., Fay, A., Wollschlaeger, M., Kowalewski, S., Göhner, P., Vogel-Heuser, B. (2014): Industrie 4.0 am Beispiel einer Verbundanlage - Aspekte der Modellierung und dezentralen Architektur. In: Automation 2014. atp edition 7-8 / 2014

[Wie09] Wiendahl, H.-P. (2009): Adaptive Production Planning and Control – Elements and Enablers of Changeability. In: ElMaraghy, H. A. (Hrsg.): Changeable and Reconfigurable Manufacturing Systems. London: Springer 2009, S. 197-212. ISBN: 9781848820661.

[Wie14] Wiendahl, H.-P. (2014). Betriebsorganisation für Ingenieure. München: Carl Hanser.

[WiGeP20] Stark, et al.(2020): WiGeP-Positionspapier: „Digitaler Zwilling“

[WH07] Windt, K., Hülsmann, M. (2007). Changing paradigms in logistics – understanding the shift from conventional control to autonomous cooperation and control. In: Hülsmann, M. & Windt, K. (Hrsg.), Understanding autonomous cooperation & control - the impact of autonomy on management, information, communication, and material flow, S. 4-16. Berlin: Springer.

[WTG+12] Wang, L.; Tang, D.-B.; Gu, W.-B.; Zheng, K.; Yuan, W.-D., Tang, D.-S. (2012). Pheromone-based coordination for manufacturing system control. Journal of Intelligent Manufacturing, 23, S. 747-757.

[WWI+16] Wuest, T., Weimer, D., Irgens, C., Thoben, K. (2016): Machine learning in manufacturing: advantages, challenges, and applications, Production & Manufacturing Research, 4:1, 23-45, DOI: 10.1080/21693277.2016.1192517

[Wyc09] Wycisk, C. (2009). Flexibilität durch Selbststeuerung in logistischen Systemen. Dissertation, Universität Bremen.

[ZB05] Zäpfel, G., Braune, R. (2005): Moderne Heuristiken der Produktionsplanung. Am Beispiel der Maschinenbelegung. München: Vahlen 2005. ISBN: 3800632381

Anhang

Inhaltsverzeichnis Seite

A1 Smart Automation Labor

Die Implementierung des in dieser Arbeit entwickelten automatisierten dezentralen Steuerungssystems fand im Smart Automation Labor des Lehrstuhls für Produktentstehung am Heinz Nixdorf Institut der Universität Paderborn statt. Das Labor dient der Forschung in den Bereichen Produktions- und Automatisierungstechnik sowie deren Anbindung an die Produktentwicklung und der anwendungsnahen Untersuchung von Geschäftsprozessen. Für die Implementierung wurde vorab das Produktionssystem des Labors im Rahmen dieser Arbeit zu einem CPPS umgebaut (siehe Abschnitt 6.1). In diesem Kapitel wird auf den vorherigen Aufbau des Labors (siehe A1.1) und den Umbau des Labors (siehe A1.2) hin zu einem CPPS eingegangen. Dieser Umbau des Labors bildet die Grundlage für diese Arbeit. Auf diesen Umbau wird anschließend die automatisierte dezentrale Produktionssteuerung angewendet. Das Labor wird hierbei insbesondere in den Kapiteln 5 und 6 dieser Arbeit vorgestellt. Der Anhang soll die Maßnahmen zur Erstellung des CPPS veranschaulichen.

A1.1 Vorheriger Ausbauzustand

Das Labor diente ursprünglich der Forschung an flexiblen Automatisierungslösungen. Dabei wurde das Labor über eine zentrale SPS des Materialflusssystems gesteuert. Darauf basierend mussten Aufträge zuerst zentral eingespeichert werden und diese wurden anschließend an das Ende der Auftragswarteschlange gesetzt und dann entsprechend ausgeführt. Dabei wurden die einzelnen Arbeitsstationen durch die SPS ferngesteuert und der gesamte Ablauf der Produktionssteuerung fand zentral statt. Im Jahr 2012 ist das Labor in das Heinz Nixdorf Institut umgezogen und aufgrund der räumlichen Gegebenheiten wurden hierbei neue Fertigungsanlagen angeschafft. Die neuen Fertigungsanlagen waren eine Drehmaschine und eine Fräsmaschine. Die Fertigungsanlagen wurden zwar in Betrieb genommen, aber sie wurden nicht mehr mit der zentralen SPS verbunden, so dass kein automatisierter Laborbetrieb ausgeführt werden konnte. Das heißt, dass die Produktionsmaschinen ausschließlich als einzelne Produktionsinseln genutzt wurden. In der nachfolgenden Abbildung A1 ist der vorherige Ausbauzustand am Heinz Nixdorf Institut veranschaulicht. Dabei waren im Labor die Drehmaschine mit Portalroboter, die Fräsmaschine mit 6-Achs-Roboter und das nicht angeschlossene Transportsystem bzw. Einschienensystem mit Be- und Entladesystemen aufgebaut. Nachfolgend wird auf die vorherigen Ausbauzustände der einzelnen Elemente des Labors eingegangen.

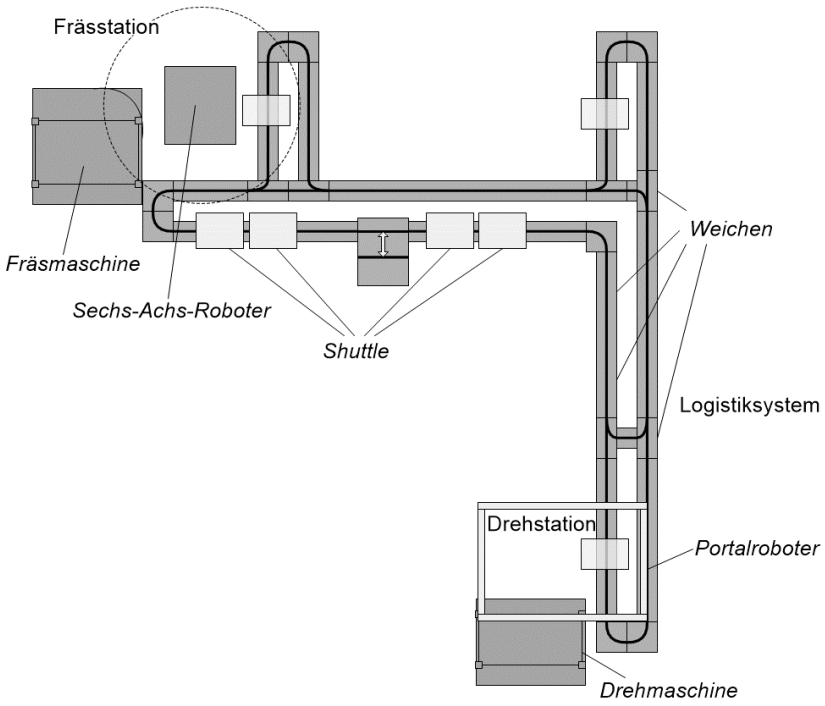

Abbildung A1: Layout des Smart Automation Labors vor dem Umbau

Transportsystem

Für den Transport zwischen den Arbeitsstationen (Drehstation und Frässtation) wurde ein Einschienen-Materiaflusssystem von der Firma Montech eingesetzt. Auf diesem Einschienensystem fahren fünf Shuttle, welche über die Weichen des Materialflusssystems geleitet wurden. Das Materialflusssystem war bereits in Betrieb genommen und konnte alle in Abbildung A1 dargestellten Arbeitsstationen anfahren, um Materialien oder Produktkomponenten zu be- und entladen. An den Arbeitsstationen befanden sich Haltestellen für die Shuttles, welche über ein externes IO-Signal gesteuert wurden. Auf jedem Shuttle befand sich ein Werkstückträger mit insgesamt 9 Einfassungen.

Das Materialflusssystem wurde ursprünglich mithilfe einer Siemens S7-SPS zentral gesteuert. An die SPS war ein HMI-Panel zur manuellen Bedienung angeschlossen. Über Magnetschalter mit induktiven Nährungssensoren fand die Steuerung der Weichen statt. Durch die induktiven Nährungssensoren konnten die Shuttle erkannt werden und anschließend über die Magnetschalter von den Weichen geleitet werden. Damit wurde ein bereits vorprogrammierte Fahrweg der Shuttles gesteuert. So konnte für jedes Shuttle ein individueller Fahrweg zu den Arbeitsstationen erstellt bzw. realisiert werden. Entlang dieses Fahrwegs sind die Shuttle kontinuierlich gefahren und sammelten dabei Materialien und fertiggestellte Produktkomponenten ein und lieferten diese ab.

Drehstation

Die Drehstation besteht aus einer Drehmaschine (EMCO E25) und einem Portalroboter. Der Portalroboter war hierbei eine Eigenentwicklung aus Aluminiumprofilen, einer Siemens S7-SPS und Antriebskomponenten von RK Rose+Krieger. Der Portalroboter war zwar aufgebaut und elektronisch bereits verkabelt, aber nicht funktionsfähig. Für die Funktionsfähigkeit fehlte das Programm für die SPS. Die Drehmaschine ist zwar über eine eigene Ein- und Ausfahrtstation für die Shuttles an das Materialflusssystem elektronisch angeschlossen, aber die fehlende Einrichtung der Schnittstellen zu der SPS des Materialflusssystems verhinderte die automatische Steuerung der Drehstation. Die Drehmaschine wurde vorwiegend für die Lehrveranstaltung „Übung NC-Programmierung“ eingesetzt.

Frässtation

Die Frässtation besteht aus einer Fräsmaschine (EMCO E350) und einem Sechs-Achs-Roboter von ABB (IRB1600). Die Fräsmaschine war in Betrieb genommen worden, die Schnittstelle zu dem Roboter und dem Materialflusssystem waren jedoch wie bei der Drehstation nicht eingerichtet. Daher wurde die Fräsmaschine lediglich für die Lehrveranstaltung „Übung NC-Programmierung“ eingesetzt. Der Roboter war für einen Stand-Alone-Betrieb eingerichtet und wurde für die Lehrveranstaltung „Übung Industrierobotik“ verwendet.

Steuerungssystem

Nach dem Umzug des Labors ins Heinz Nixdorf Institut war kein Steuerungssystem in dem Produktionssystem implementiert. Davor wurden die Produktionsanlagen über Signale der SPS des Materialflusssystems gesteuert. Das heißt, dass über IO-Schnittstellen Fertigungsprozesse angestoßen wurden und auch die Abholung und Bereitstellung von Materialien durch das Materialflusssystem geregelt wurden. Die Elemente des Materialflusssystems waren dabei über Profibus mit anderen Produktionselementen verbunden. So konnten die Produktionsanlagen gesteuert werden. Ferner konnten die fertiggestellten Produktkomponenten vom Materialflusssystem abgeholt werden. Mittels RFID konnte zusätzlich die Position der Shuttles nachvollzogen werden. Das Rohmaterial wurde dabei an den Arbeitsstationen gelagert und nur die fertig bearbeiteten Produktkomponenten wurden von den Robotern entnommen und auf die Shuttles gelegt. Die SPS des Materialflusssystems übernahm hierbei die Steuerung der einzelnen Prozesse und diente somit als Leitsystem für das gesamte Produktionssystems.

Im Rahmen des Umzugs wurde diese Steuerung nicht in das Produktionssystem des Labors im Heinz Nixdorf Institut portiert. Vor dem Umbau im Rahmen dieser Arbeit war somit kein Steuerungssystem in dem Produktionssystem im Labor vorhanden.

Demonstrator

Als Demonstrator für das Labor wurde eine Taschenlampe entwickelt. Die Taschenlampe bestand aus Eigenfertigungsteilen, die im Labor gefertigt wurden und weiteren Zukaufteilen. Die Abbildung A2 veranschaulicht die Taschenlampe.

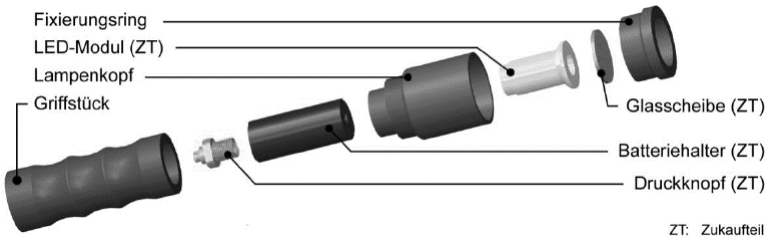

Abbildung A2: Vorheriger Demonstrator der Taschenlampe [PBG+14]

Als Anwendungsszenario wurde ursprünglich die Produktion von 1.000 individuellen Taschenlampen mit Losgröße 1 geplant [PBG+14]. Das Taschenlampengehäuse sollte gedreht werden und der Lampenkopf gefräst werden. Die beiden Komponenten sollten mit Zukaufteilen durch Shuttles zur Montagestation transportiert werden. Dieser Plan wurde jedoch nicht weiterverfolgt und die automatische Produktion in dem Labor nicht umgesetzt.

A1.2 Umbau

Die Implementierung des in dieser Arbeit entwickelten Steuerungssystems setzt eine dezentrale Produktionsstruktur voraus. Daher wurde das zentrale Produktionssystem zu einem dezentralen CPPS umgebaut. Im Kapitel 6 wurden unter anderem die Auswahlentscheidungen der eingesetzten Soft- und Hardware beschrieben. In diesem Abschnitt wird auf den Umbau des Labors eingegangen. Für weitere Bearbeitungsmöglichkeiten wurden zwei weitere Arbeitsstationen implementiert: eine 3D-Drucker-Station mit angebundenem Industrieroboter und eine Montagestation für die manuelle Montage. Das umgebaute Produktionssystem im Labor wird in der nachfolgenden Abbildung A3 dargestellt. Im Folgenden wird auf die einzelnen Arbeitsstationen und dabei insbesondere auf ihre Veränderungen eingegangen.

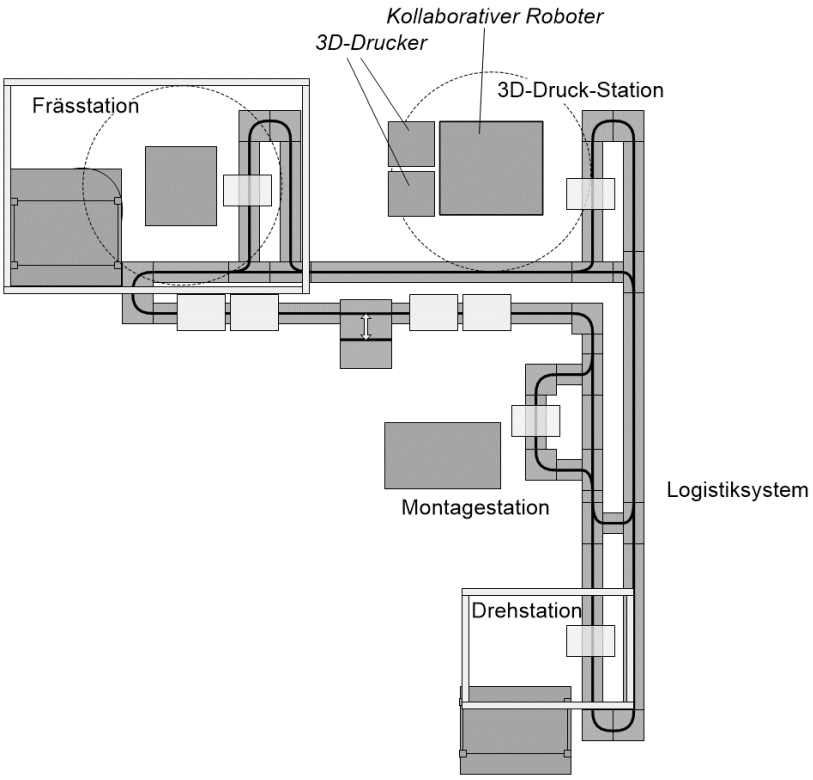

Abbildung A3: Darstellung des umgebauten Produktionssystems des Smart Automation Labors

Transportsystem

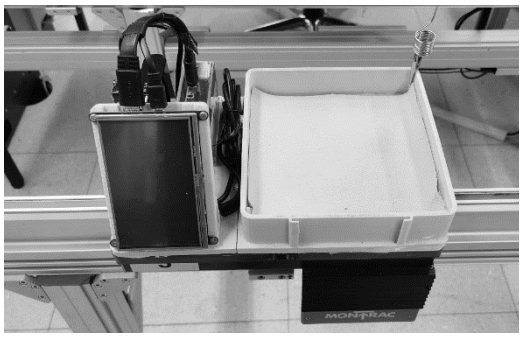

Abbildung A4: Shuttle

Die Grundstruktur des Transportsystems wurde nicht verändert. Für den neu eingeführten 3D-Drucker und die Montagestation wurden zwei Be- und Entladestationen in dem Transportsystem hinzugefügt. Darüber hinaus wurde die Steuerung der einzelnen Elemente des Transportsystems von Grund auf neugestaltet, damit die Logistikelemente als ein selbstorganisiertes Produktionselement in dem Labor mit anderen Produktionsstationen kommunizieren können. Zudem wurde ein neuer Aufsatz (siehe Abbildung A4) für das Shuttle entwickelt, das aus einem Raspberry Pi, einem Touchscreen, einem Barcodescanner und einem Werkstückträger besteht. In dem Raspberry Pi wurde das Steuerungssystem installiert. Damit wird das Shuttle zur Bearbeitung der Aufträge gesteuert. Zusätzlich wird mit dem Barcodescanner die Position des Shuttles erfasst. Mit dem Touchscreen wird eine Möglichkeit zur Visualisierung und Kommunikation mit den Beschäftigten geschaffen. Die Shuttle transportieren die fertiggestellten Komponenten zur Montagestation. Dort werden die Komponenten von Beschäftigten aus dem Shuttle entnommen. Dies erfordert keine

definierte Positionierung der Komponenten auf dem Werkstückträger. Daher wurde der Werkstückträger als ein einfaches Behältnis ausgelegt. Das Programm des Shuttles wird in Abschnitt A2.5 präsentiert.

Die Shuttle erkennen über Barcodes (siehe Abbildung A5) ihre eigene Position. Auf den Weichen sind Barcodes mit der Kennung der aktuellen Position und des nächsten Kontrollpunkts angebracht. Dadurch weiß das Shuttle, mit welcher Be- und Entladestation es als Nächstes kommunizieren muss. Das Shuttle kann sich durch diese Kommunikation selbst zu der geforderten Arbeitsstation leiten. Dafür geben die Arbeitsstationen die entsprechenden Fahrtaufträge vor. Fahrtaufträge werden z.B. kurz vor Fertigstellung eines Produktionsauftrags erstellt, damit die Shuttles die fertiggestellten Bauteile von den Arbeitsstationen abholen können und zur Montage bringen können. Sobald die Shuttles eine Anfrage für einen Fahrtauftrag erhalten haben, überprüfen diese, ob sie frei sind und wieviel Zeit sie bis zum Erreichen der Arbeitsstation benötigen. Nach der Überprüfung melden die Shuttles dies den Arbeitsstationen zurück. Die Arbeitsstation wählt anschließend das nächste freie Shuttle aus und weist dem Shuttle den Fahrtauftrag zu. Ist kein Shuttle frei, wird die Anfrage des Fahrtauftrags so lange wiederholt, bis ein Shuttle diesen Fahrtauftrag übernehmen kann.

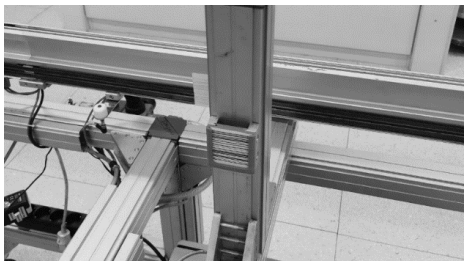

Abbildung A5: Barcodes

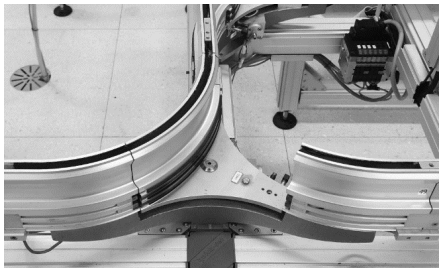

Abbildung A6: Weiche

Die Fahrt der Shuttles wird über die Weichen des Einschienensystems gesteuert (siehe Abbildung A6). Dafür wurden ebenso entsprechende Steuerungen mit Raspberry Pis realisiert. Allerdings sind die Weichen hierbei ausschließlich passive Elemente und schalten den Fahrtweg anhand der Befehle der Shuttles. Die Shuttles melden sich bei den Weichen mit der Zielstation an. Sobald die Weiche mit der Zielstation übereinstimmt, lässt diese die Shuttles in die Be- und Entladestation einfahren. Ansonsten lässt die Weiche das Shuttle weiterfahren. Das Umschalten von Weichen geschieht über Pneumatikventile. Die Raspberry Pis steuern die Pneumatikventile mithilfe von Relaiskarten. Der Aufbau des Schaltschranks für eine Weiche ist in der Abbildung A7 dargestellt. Das Programm der Weichen ist im Anhang A2.5 dargestellt.

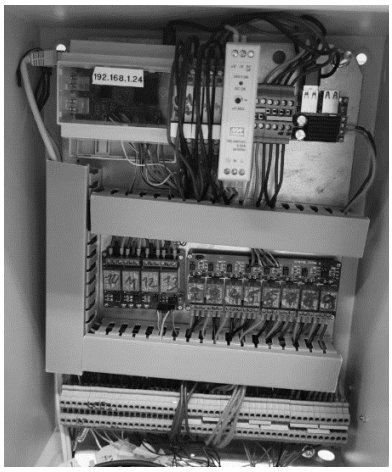

Abbildung A7: Schaltschrank

Drehstation

An der Drehstation (siehe Abbildung A8) wurde die Drehmaschine bereits in Betrieb genommen, aber nicht mit dem Produktionssystem im Labor verbunden. Hierfür wurde eine entsprechende Schnittstelle programmiert. Zudem wurde für das Drehen der Räder von dem Demonstrator „Fernsteuerungsauto“ (siehe Kapitel 6) ein CNC-Programm geschrieben. Für den Abtransport der fertiggedrehten Räder, wurde ein Behältnis als Vorrichtung gebaut, in dem die Räder nach dem Drehen gesammelt werden. Diese werden anschließend über Rutschen und Zylinder an eine Position transportiert, von der sie von dem Portalroboter entgegengenommen werden können. Darüber hinaus wurde ein Rohmateriallager für die Drehmaschine konstruiert. Darin wird das Rohmaterial, abgelängtes POM-Stangenmaterial, für die Räder des Fernsteuerungsautos gelagert. Der Portalroboter war konstruktiv fertiggestellt. Einzig ein Greifer für das Greifen des Behältnisses wurde neu konstruiert. Der Portalroboter wurde neu programmiert, so dass die fertiggedrehten Räder in die Shuttle ablegt werden können. Die Drehstation wurde mit dem Steuerungssystem für die dezentrale Produktionssteuerung ausgestattet. Für die Steuerung wurde hierbei auf Grund der zahlreichen Ein- und Ausgänge ein Beaglebone verwendet. Dabei wurde zusätzlich ein Monitor angeschlossen, damit der aktuelle Status der Drehstation visualisiert wird. Auf dem Beaglebone wird die gleiche Grundsoftware (siehe Anhang A2.6) wie auf den anderen Arbeitsstationen verwendet. Die Prozessausführung wurde hierbei explizit für die Ausführung des Drehprozesses geschrieben.

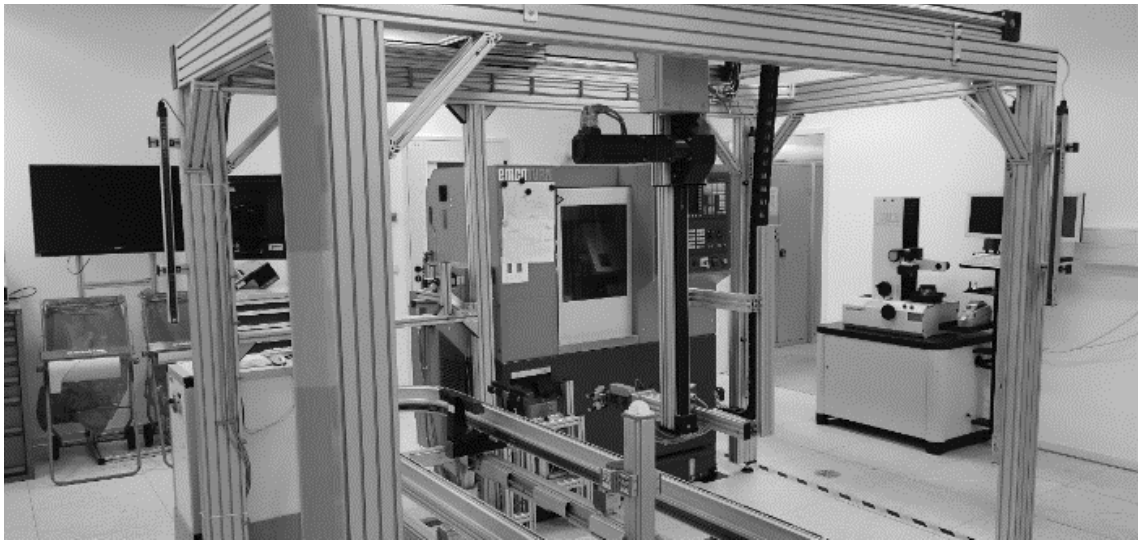

Abbildung A8: Drehstation

Frässtation

Analog zur Drehstation wurde bei der Frässtation (siehe Abbildung A9) ebenso eine Steuerung zur Prozessausführung für das Fräsen entwickelt und ein Materiallager konstruiert. In dem Materiallager wird das Rohmaterial bzw. das Plattenmaterial für das Fräsen abgelegt. Das Materiallager wird von dem Steuerungssystem zur

Prozessausführung über den Raspberry Pi gesteuert. Zudem wurde die Frässtation mit Lichtschranken eingehaust, um einen automatisierten Betrieb zu ermöglichen. Die Schnittstelle von der Fräsmaschine zu dem Steuerungssystem zur Prozessausführung wurde über den Roboter hergestellt. Das bedeutet, dass der Raspberry Pi an den Roboter angeschlossen ist, der wiederum mit der Fräsmaschine verbunden ist. Für den Roboter wurde ein neuer Greifer konstruiert, mit dem das Plattenmaterial transportiert werden kann. Der Roboter wurde vorher bereits in Betrieb genommen. Daher musste für den Einsatz des Roboters nur der Ablauf und die Fahrtwege einprogrammiert werden.

Abbildung A9: Frässtation

3D-Drucker-Station

Die 3D-Drucker-Station (siehe Abbildung A10) wurde komplett neu entwickelt und aufgebaut. Da der Roboter direkt über ROS gesteuert wird, wurde hierbei ein Computer mit IO-Karte für die Steuerung verwendet. Zudem wurde der kollaborative Roboter Baxter von Rethink Robotics eingesetzt, der keine Einhausung benötigt. Im Rahmen der Validierung findet jedoch kein Druckprozess statt. Die Fertigteile wurden direkt aus einem Lager entnommen (siehe Kapitel 7). Dafür wurde für den Roboter ein entsprechendes Lagerprogramm geschrieben.

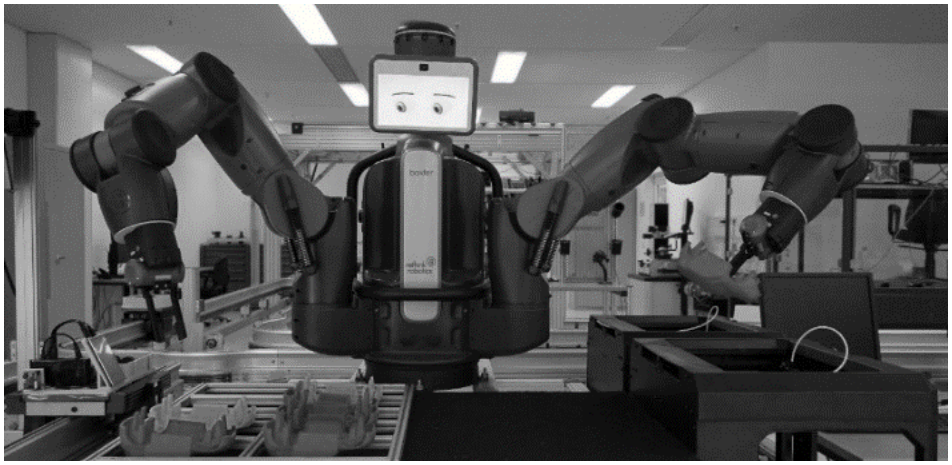

Abbildung A10: 3D-Drucker-Station

Montagestation

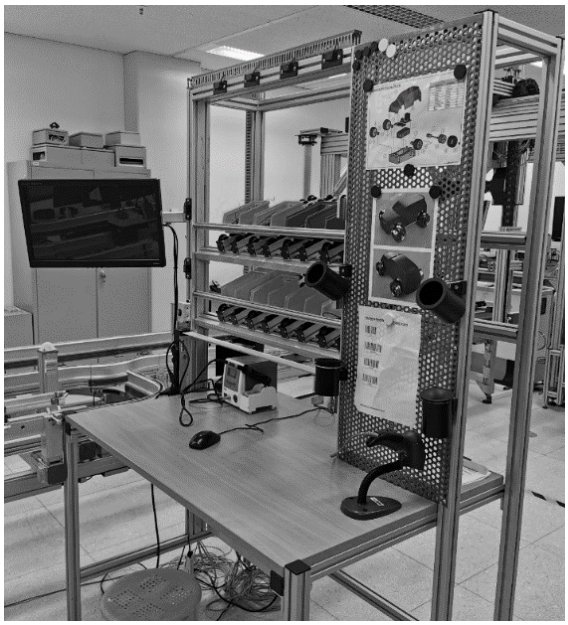

Abbildung A11: Montagestation

Die Montagestation wurde vollständig neu aufgebaut. An dieser wurden Untersuchungen zu dem Themenfeld Arbeit 4.0 und der digitalen Repräsentation durchgeführt. Die Montagestation wurde hierbei mit Aluprofil aufgebaut und besteht neben dem Steuerungssystem mit Raspberry Pi im Wesentlichen aus einem Touchscreen, einem Tiefenmonitor, einem selbstentwickeltem pick-by-light-System mit Materialzufuhr und entsprechenden Werkzeugen für die Montage. Die entwickelte Montage-schrittführung, die wesentlicher Teil des Programms zur Steuerung der Montagestation ist, wurde in Abschnitt 6.3 vorgestellt.

Demonstrator

Als Demonstrator wurde ein Fernsteuerungsauto (siehe Abbildung A12) entwickelt (siehe Abschnitt 7.1). Das Fernsteuerungsauto besteht aus mehreren Zukaufteilen, wie einem Raspberry Pi Zero, Motoren, Schrauben etc. und drei Eigenfertigungsteilen. Die drei Eigenfertigungsteile sind hierbei Räder aus der Drehstation, das Chassis aus der Frässtation und die Karosserie aus der 3D-Drucker-Station. Für alle Eigenfertigungsprozesse wurden hierbei Varianten erstellt, damit ein individuelles Fernsteuerungsauto gefertigt werden kann. Bei der Karosserie können zum Beispiel unterschiedliche Farben des Materials gewählt werden. Für die Erstellung der Aufträge wurde ein Konfigurator als Website mit PHP / Javascript (siehe Abbildung A12) erstellt. Die Steuerung des Fernsteuerungsautos erfolgt über einen Raspberry Pi Zero mit Bluetooth-Dongle. Der Raspberry Pi kann über Bluetooth mit dem Handy oder einem Konsolencontroller zur Fernsteuerung verbunden werden. Hierzu wurde für den Raspberry Pi ein entsprechendes Programm und für die Smartphonebedienung eine entsprechende App geschrieben.

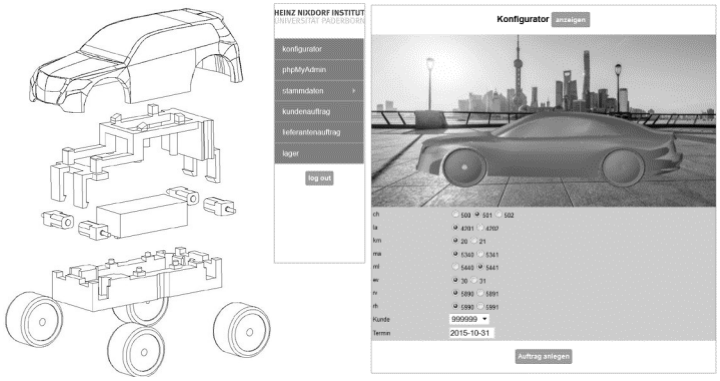

Abbildung A12: Montagestation

A2 Programmbeschreibung

Nachfolgend wird das in dieser Arbeit entwickelte Programm zur Steuerung des Smart Automation Labors, in dem die dezentrale Produktionssteuerung mit der digitalen Repräsentation der Beschäftigten enthalten sind, dargestellt.

Der Programmcode wurde in Python und C geschrieben. Für die Nachvollziehbarkeit und die mögliche Darstellbarkeit wurden die Teile des Programmcodes, die nicht für die Kernabläufe der dezentralen Steuerung notwendig sind, nicht dargestellt. Dazu zählen die Importroutinen, die Programmierung von GUIs, die Bearbeitung spezifischer Laborprobleme und der Austausch und die Bearbeitung von Daten. Zudem werden die spezifischen Lösungen und Steuerungsroutinen, wie z.B. die Programmierung von Schnittstellen zu bestehenden Anlagen wie der Fräsmaschine nicht dargestellt. Diese sind für die Ausführung der automatisierten dezentralen Produktionssteuerung nicht weiter relevant, umfassen jedoch sehr viel Programmcode, so dass diese nicht mit in den Anhang aufgenommen wurden. Daraus ergibt sich der Kerncode über alle Ablaufroutinen, die für die Planung und Steuerung der Produktion und die Implementierung der digitalen Repräsentation notwendig sind. Dieser Kerncode wird nachfolgend vollständig dargestellt.

Zuerst werden die gemeinsamen Funktionen des Steuerungssystems in Datenbank, Kommunikationen zwischen den Produktionselementen und weitere Funktionen veranschaulicht. Dann wird die zentrale Station des Steuerungssystems, in der die Aufträge ankommen, vorgestellt. Die zentrale Station dient als Auftragsspeicher und auf dieser sind gemeinsam verwendete Datenbanken abgelegt. Anschließend wird auf den Code des Transportsystems eingegangen. Dann folgt die Darstellung des Codes zur Ausführung der einzelnen Arbeitsstationen. Zum Schluss wird der Code zur Umsetzung der digitalen Repräsentation angehängt.

A2.1 Gemeinsame Funktionen

Nachfolgend werden alle Programmteile vorgestellt, die von den meisten Produktionselementen gemeinsam benutzt werden. Diese sind in dem repository „common“ abgespeichert und dienen dazu, generelle Funktionen abzubilden.

A2.1.1 Datenbank

Die wichtigsten, anlagenübergreifenden Dokumente werden in der nachfolgenden Abbildung A13 in vereinfachter Weise dargestellt.

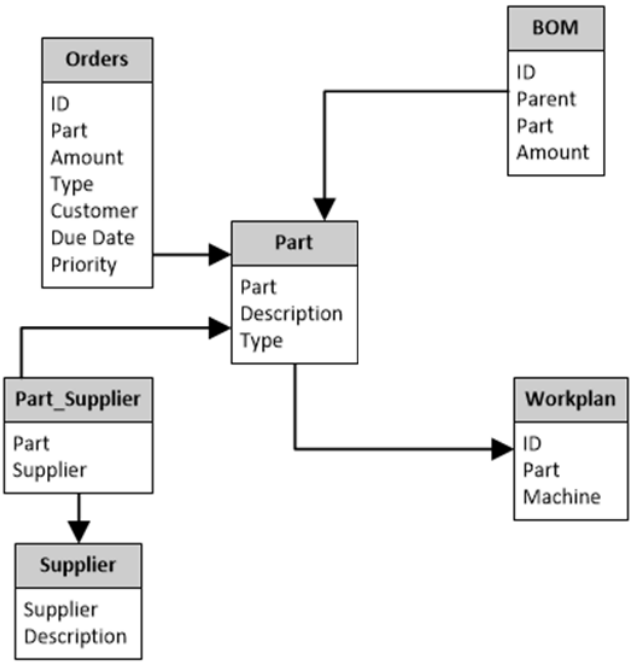

Abbildung A13: Anlagenübergreifenden Dokumente

Für den Austausch zwischen den Stationen werden die Aufträge (Orders) untereinander ausgetauscht. Diese werden von Station zu Station angefragt. Die Stückliste (BOM – Bill of Material) dient dazu, den Produktionsprozess abzubilden. Ein Produkt wird so lange von Lagerstationen und Produktionsanlagen abgefragt, bis alle Teile für dessen Produktion verteilt wurden und so die Produktion starten kann.

mysql_class.py

```
#standart library:
from __future__ import absolute_import
import threading
import collections
import logging

#extra library:
import pymysql
import pymysql.cursors as pycur

class OrderedDictCursorMixin(pycur.DictCursorMixin):
    dict_type = collections.OrderedDict

class OrderedDictCursor(OrderedDictCursorMixin, pycur.Cursor):

class sqldb(object):
    def __init__(self,host,user,passwd,db):
        self.args = {"host" : host,
                     "user" : XXX,
```

```
                "passwd" : XXX,
                "db" : XXX,
                "charset" : 'utf8mb4',
                "cursorclass" : pymysql.cursors.DictCursor}
    self.cursors = {"dictcur":pymysql.cursors.DictCursor,
                    "orderddictcur":OrderedDictCursor,
                    "list":pycur.Cursor}
    self.rlock = threading.RLock()

#----------------------------

def _select(self,sql):

    with self.rlock:
        with self.connection.cursor() as cursor:
            rowcount = cursor.execute(sql)
            rows = cursor.fetchall()
            tmpdesc = cursor.description
            desc = []
            [desc.append(x[0]) for x in tmpdesc]

        return {"rows":rows, "description":desc, "rowcount":rowcount}

#----------------------------

def _update(self,sql):

    with self.rlock:
        with self.connection.cursor() as cursor:
            cursor.execute(sql)
        self.connection.commit()
        return True

#----------------------------

def sqlquery(self, sql, curtype="dict"):

    with self.rlock:

        if curtype == "dict":
            self.args["cursorclass"] = self.cursors["dictcur"]
        elif curtype == "ordereddict":
            self.args["cursorclass"] = self.cursors["orderddictcur"]
        else:
            self.args["cursorclass"] = self.cursors["list"]

        self.connection = pymysql.connect(**self.args)
        try:
            if sql.startswith("SELECT"):
                return self._select(sql)

            elif sql.startswith(("UPDATE","INSERT","DELETE")):
                return self._update(sql)

        finally:
            self.connection.close()
```

A2.1.2 Kommunikation

p2p_framework.py

```
from __future__ import absolute_import, print_function
```

```
import threading
import struct
import socket
import time
import sys
import signal
import os
import logging
from datetime import datetime

# import user defined libraries
from . import tcp_framework
from . import udp_framework
from . import ip_module
#=============================================================================
class P2P_Interface(object):
    #-----------------------------------------------------------------------
    def __init__(self, shutdown, name, type, router_address, logger=None,
                 tcp_ports=54545, udp_ports=56565, broadcast=True):

        self.name = name
        self.type = type

        self.shutdown = shutdown
        self.broadcast = broadcast

        # initialising logger if not given (using logging module)
        self.logger = logger or logging.getLogger(__name__)

        self.own_address = ''
        self.own_address = ip_module.get_lan_ip()


        self.msg_list = []
        self.msg_lock = threading.Lock()
        self.msg_event = threading.Event()
        self.msg_event.clear()
        self.msg_id = 0
        self.failed_to_send_list = []

        self.tcp_peer = tcp_framework.Peer(tcp_ports,
                                    self.add_message_to_list,
                                    self.shutdown)

        self.udp_peer = udp_framework.Peer(udp_ports,
                                    self.add_message_to_list,
                                    self.shutdown)

        self.address_book = {'Broadcast' : ['<broadcast>', None, 100]}
        if self.broadcast:
            t_address_book = threading.Thread( target = self.update_address_book,
                                    args=[6,12])
            t_address_book.start()

        self.handlers = {}
        t_handlemessages = threading.Thread( target = self.handlemessages)
        t_handlemessages.start()

    def mysignal_handler(self,signal,frame):
        print("you pressed ctrl+c !!")
        self.shutdown[0] = True
        sys.exit(0)

    #-----------------------------------------------------------------------

    def handlemessages( self ):
```

```
    while not self.shutdown[0]:
        self.msg_event.wait(1)
        if self.msg_list:
            self.msg_lock.acquire()
            self.handlemessage(self.msg_list.pop())
            self.msg_event.clear()
            self.msg_lock.release()

#---------------------------------------------------------------------------
def update_address_book( self, cycle_time, timeout ):

    while not self.shutdown[0]:

        self.sendmessage('UDP', 'Broadcast', '', 'introduction', '')
        blacklist = []

        for name in self.address_book:
            if self.address_book[name][2] == timeout:
                blacklist.append(name)
            else:
                self.address_book[name][2] += 1

        for name in blacklist:
            del self.address_book[name]
            self.logger.info('Removed {} from the address book.'.format(name))

        time.sleep(cycle_time)

#---------------------------------------------------------------------------
def handlemessage( self, msg ):

    if msg['command'] == 'introduction':
        if not msg['sendername'] in self.address_book:
            self.logger.info('Added {} to the address book.'.format(msg['sendername']))

        self.address_book[msg['sendername']] = \
                        [msg['senderaddr'], msg['sendertype'], 0]

    elif msg['command'] in self.handlers:
        self.handlers[msg['command']](msg)

    else:
        self.logger.error("Error: No event handler given for command {}".format(msg['command']))


#---------------------------------------------------------------------------
def encodemsg( self, recvaddr, recvtype, command, data, prio, ack):

    ctime = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    fmt_msg = "Qhh%ds%ds%ds%ds%ds%ds%ds" % (len(recvaddr),
                                    len(recvtype),
                                    len(self.own_address),
                                    len(self.name),
                                    len(self.type),
                                    len(command),
                                    len(data),
                                    len(ctime)
                                    )

    recvaddr = recvaddr.encode("utf-8")
    recvtype = recvtype.encode("utf-8")
    own_address = self.own_address.encode("utf-8")
    name = self.name.encode("utf-8")
    type = self.type.encode("utf-8")
```

```
    command = command.encode("utf-8")
    data = data.encode("utf-8")
    ctime = ctime.encode("utf-8")

    msg_encoded = struct.pack(fmt_msg,
                        self.msg_id,        # message ID          Q
                        prio,               # priority            h
                        ack,                # acknowledgement     h
                        recvaddr,           # receiver address    s
                        recvtype,           # receiver type       s
                        own_address,   # sender address           s
                        name,          # sender name              s
                        type,          # sender type              s
                        command,            # command             s
                        data,               # data                s
                        ctime)              #current time        s

    fmt_wrapper = "48s%ds" % len(msg_encoded)

    fmt_msg = fmt_msg.encode("utf-8")

    msg_wrapped = struct.pack(fmt_wrapper, fmt_msg, msg_encoded)

    return msg_wrapped


#---------------------------------------------------------------------------
def decodemsg( self, encoded_msg ):
    msg_dict = {}

    try:
        msg = struct.unpack(encoded_msg[:48], encoded_msg[48:])

        msg_dict['message_id'] = msg[0]
        msg_dict['priority'] = msg[1]
        msg_dict['acknowledgement'] = msg[2]
        msg_dict['recvaddr'] = msg[3].decode("utf-8")
        msg_dict['recvtype'] = msg[4].decode("utf-8")
        msg_dict['senderaddr'] = msg[5].decode("utf-8")
        msg_dict['sendername'] = msg[6].decode("utf-8")
        msg_dict['sendertype'] = msg[7].decode("utf-8")
        msg_dict['command'] = msg[8].decode("utf-8")
        msg_dict['data'] = msg[9].decode("utf-8")

    except Exception as error:
        self.logger.error('Error: Message could not be decoded. {}'.format(error))
        return None

    try:
        msg_dict['strtime'] = msg[10].decode("utf-8")
        msg_dict['time'] = datetime.strptime(msg_dict['strtime'],
                                             "%Y-%m-%d %H:%M:%S")
    except IndexError:
        pass

    return msg_dict


#---------------------------------------------------------------------------
def add_message_to_list( self, encoded_msg ):

    msg = self.decodemsg(encoded_msg)
    self.msg_lock.acquire()
    self.msg_list.append(msg)
    self.msg_list.sort(key=lambda msg: msg['priority'])
    self.msg_lock.release()
```

```
    self.msg_event.set()

#---------------------------------------------------------------------------
def add_handler( self, command, handler ):

    self.handlers[command] = handler


#---------------------------------------------------------------------------
def get_address_book( self ):

    address_book = []

    for name in self.address_book:
        address_book.append([name, self.address_book[name][1]])

    return address_book


#---------------------------------------------------------------------------
def sendmessage( self,contype, recvname, recvtype, command, data, prio=1, ack=0):

    error_text = 'ERROR'
    encoded_msg = None

    try:
        if contype == 'TCP' and recvname == 'Broadcast':
            contype = 'UDP'
        recvaddr = ''
        if recvname in self.address_book:
            recvaddr = self.address_book[recvname][0]
        else:
            error_text = 'Name of receiver is unknown!'
            raise

        encoded_msg = \
            self.encodemsg(recvaddr, recvtype, command, data, prio, ack)

        success = False

        if contype == 'TCP':
            success = self.tcp_peer.connectandsend(recvaddr, encoded_msg)
        elif contype == 'UDP':
            success = self.udp_peer.send(recvaddr, encoded_msg)
        else:
            error_text = \
                'No valid connection type given! Choose either TCP or UDP.'
            raise

        if success == False:
            error_text = 'Error during transmission of message!'
            raise

    except Exception as error:
        if error_text == "ERROR":
            error_text = error
        self.logger.error('An error occured while sending a message: {}'.format(error_text))
        self.failed_to_send_list.append(encoded_msg)

    self.msg_id += 1
```

tcp_framework.py

```
from __future__ import absolute_import
import socket
import threading
import logging

#=============================================================================
class Peer:

    #-----------------------------------------------------------------------
    def __init__(self, serverports, add_message_to_list, shutdown, logger=None):

        if type(serverports) in (list,tuple):
            self.__sendport = int(serverports[0])
            self.__recvport = int(serverports[1])
        else:
            self.__sendport = int(serverports)
            self.__recvport = int(serverports)

        self.shutdown = shutdown

        if logger is None:
            self.logger = logging.getLogger(__name__)
            self.logger.setLevel(30)
        else:
            self.logger = logger

        # create a seperated thread for listening for incoming messages
        t = threading.Thread( target = self.__mainloop,
                              args=[add_message_to_list])
        t.start()


    #-----------------------------------------------------------------------
    def __mainloop( self, add_message_to_list ):

        s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
        s.setsockopt( socket.SOL_SOCKET, socket.SO_REUSEADDR, 1 )
        s.bind( ( '', self.__recvport ) )
        s.listen(5)
        s.settimeout(2)

        while not self.shutdown[0]:
            try:
                clientsock, clientaddr = s.accept()
                clientsock.settimeout(None)

                msg = clientsock.recv(2048)
                add_message_to_list(msg)

                clientsock.close()
                clientsock = None
            except socket.timeout:

                continue
            except Exception as error:
                self.logger.critical("{}".format(error))
                self.shutdown[0] = True
                raise

        s.close()


    #-----------------------------------------------------------------------
```

```
def connectandsend( self, host, msg ):

    try:
        s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
        s.connect( ( host, int(self.__sendport) ) )
        s.sendall( msg )
        s.close()

    except Exception as error:
        self.logger.error("{}".format(error))
        return False

    return True
```

udp_framework.py

```
# import standard libraries
from __future__ import absolute_import
import socket
import threading
import logging

#===============================================================================
class Peer:
    """
    This class is used for receiving and sending messages using UDP. Therefore
    a seperated thread is started which will be listening for incoming messages
    at the specified port. Any message received will be added to a list.
    """

    #---------------------------------------------------------------------------
    def __init__(self, serverports, add_message_to_list, shutdown, logger=None):
    #---------------------------------------------------------------------------
        """
        Initializing of an object of the class Peer.

        serverport  :          Port number which is used for listening and
                               sending messages using UDP. Any port number can
                               be chosen but make sure all clients are using
                               the same port number for UDP communication.
        add_message_to_list :  A function adding a received message to the
                               list of messages.
        shutdown    :          A parameter used for stopping the infinite loop
                               of the main thread which is listening for
                               incoming messages.
        """

        # initialize object variables
        if type(serverports) in (list,tuple):
            self.__sendport = int(serverports[0])
            self.__recvport = int(serverports[1])
        else:
            self.__sendport = int(serverports)
            self.__recvport = int(serverports)

        self.shutdown = shutdown

        if logger is None:
            self.logger = logging.getLogger(__name__)
            self.logger.setLevel(30)
        else:
            self.logger = logger

        # create a seperated thread for listening for incoming messages
        t = threading.Thread( target = self.__mainloop,
                              args=[add_message_to_list])
        t.start()


    #---------------------------------------------------------------------------
    def __mainloop( self, add_message_to_list ):
    #---------------------------------------------------------------------------
        """
        This method creates a socket at the specified port. An infinite loop is
        used for listening for incoming messages. If a message is received it
        will be added to a list and the loop continues listening for new
        messages.
```

```
    This method is private.

    add_message_to_list :   A function adding a received message to the
                            list of messages.
    """

    # create a socket for UDP communication
    s = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
    # set options of the socket
    s.setsockopt( socket.SOL_SOCKET, socket.SO_REUSEADDR, 1 )
    # bind socket to the defined port
    s.bind( ('', self.__recvport) )
    # set socket timeout to 2 seconds
    s.settimeout(2)

    # infinte loop
    while not self.shutdown[0]:
        try:
            # if a message is received ...
            msg, addr = s.recvfrom(2048)
            # msg contains the actual message
            # addr contains the address of the sender

            # ... add it to the list
            add_message_to_list(msg)

        except socket.timeout:
            # If the listening socket times out (happens every 2 seconds)
            # an error is raised which will be caught by this block.
            # The command 'continue' will return to the while loop and
            # check whether the shutdown parameter has changed. If the
            # shutdown parameter has turned to True the while loop
            # terminates. Otherwise it will continue listening.
            continue

        except Exception as error:
            self.logger.critical("{}".format(error))
            self.shutdown[0] = True
            raise

    # close the socket
    s.close()


#---------------------------------------------------------------------------
def send( self, host, msg ):
#---------------------------------------------------------------------------
    """
    This method is used for sending messages using UDP. The method returns
    True after successfully sending a message. If an error occures during
    sending the message, the method returns False and an error message is
    printed within the console.

    This method is public.

    host    :   contains the address of the receiver
    msg     :   contains the content of the message
    """

    try:
        # create a socket for UDP communication
        s = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )

        # if the sending operation is a broadcast, set the options for
        # broadcasting
        if host == '<broadcast>' or host.endswith(".255"):
```

```
            s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        # concatenate the address of the receiver with the port number
        #addr = (host, self.__serverport)
        addr = (host, self.__sendport)

        # send the message to the address
        s.sendto(msg, addr)

        # close the socket
        s.close()

    except Exception as error:
        # if any error occures, print the error message in the console
        # and return False
        self.logger.error("{}".format(error))
        return False

    # return True after successfully sending the message
    return True
```

ip_module.py

```
from __future__ import absolute_import
from __future__ import print_function
import os
import socket
import sys

if os.name != "nt":
    import fcntl
    import struct

    def get_interface_ip(ifname):
        ifnamebyte = ifname[:15].encode("utf-8")
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        return socket.inet_ntoa(fcntl.ioctl(s.fileno(), 0x8915, struct.pack('256s',
                                ifnamebyte))[20:24])

def get_lan_ip():
    ip = socket.gethostbyname(socket.gethostname())
    if ip.startswith("127.") and os.name != "nt":
        interfaces = [
            "eth0",
            "eth1",
            "eth2",
            "wlan0",
            "wlan1",
            "wifi0",
            "ath0",
            "ath1",
            "ppp0",
            ]
        for ifname in interfaces:
            try:
                ip = get_interface_ip(ifname)
                break
            except IOError:
                pass
    return ip



def main():
            print("hello ip ")
            print("number of arguments: ",len(sys.argv),"ARGUMENTS")
            print("argument[0] :",sys.argv[1])
            print(type(sys.argv[1]))
            my_ip = get_lan_ip()
            print(my_ip)


if __name__ == "__main__":
            main()
```

A2.1.3 Weitere Funktionen

decorator.py

```
def trace(func):
    def wrapper(*args):
        print("Calling: %s..." % func.__name__)
        return func(*args)
    return wrapper

@trace
def myFunc():
    print "Here I am"

myFunc()

# ---------------------------------------------------------------------

def count(func):
    class static:
        calls = 0
    def wrapper(*args):
        static.calls += 1
        print("'%s', call #%i" % (func.__name__, static.calls))
        return func(*args)
    return wrapper

@count
def func1():
    pass

@count
def func2():
    pass

for i in xrange(15):
    if not (i % 5):
        func1()
    else:
        func2()

# ---------------------------------------------------------------------

class Function:

    def __init__(self):
        self.funcs = {}

    def overload(self, *types):
        def decorator(func):
            self.funcs[types] = func
            def wrapper(*args):
                return func(*args)
            return wrapper
        return decorator

    def __call__(self, *args):
        types = tuple(arg.__class__ for arg in args)
        try:
            return self.funcs[types](*args)
        except:
            raise TypeError("No function for arguments: %s" % str(args))

test = Function()
```

```
@test.overload(int)
def _test(x):
   print("int function")

@test.overload(int, int)
def _test(x, y):
   print("int, int function")

@test.overload(float)
def _test(x):
   print("float function")

@test.overload(str)
def _test(x):
   print("string function")

# ------------------------------------------------------------------

import time
import random

def profile(func):
   def wrapper(*args):
      start = time.time()
      ret = func(*args)
      finish = time.time()
      name = func.__name__
      print("'%s' took: %0.5fms" % (name, (finish-start) * 1000))
      return ret
   return wrapper

@profile
def shuffle(lst):
   for i in xrange(len(lst)-1, 0, -1):
      j = random.randint(0, i)
      lst[i], lst[j] = lst[j], lst[i]

@profile
def sort(lst):
   for i in xrange(len(lst) - 1):
      low = i
      for j in xrange(i + 1, len(lst)):
         if lst[j] < lst[low]:
            low = j
      lst[i], lst[low] = lst[low], lst[i]

for size in [16, 32, 64, 128, 256, 512]:
   print("===== Testing size: %i =====" % size)
   lst = range(size)
   shuffle(lst)
   sort(lst)

# ------------------------------------------------------------------

def multi(dispatch_fn):
   def _inner(*args, **kwargs):
      return _inner.__multi__.get(
         dispatch_fn(*args, **kwargs),
         _inner.__multi_default__
      )(*args, **kwargs)

   _inner.__multi__ = {}
   _inner.__multi_default__ = lambda *args, **kwargs: None  # Default default
   return _inner

def method(dispatch_fn, dispatch_key=None):
```

```
    def apply_decorator(fn):
        if dispatch_key is None:
            # Default case
            dispatch_fn.__multi_default__ = fn
        else:

            dispatch_fn.__multi__[dispatch_key] = fn
        return dispatch_fn
    return apply_decorator


# Dispatch on types


class Person(object):
    def __init__(self, name):
        self.name = name

@multi
def get_name(obj):
    return obj.__class__

@method(get_name, dict)
def get_name(obj):
    return obj['name']

@method(get_name, Person)
def get_name(obj):
    return obj.name

@method(get_name)  # Default
def get_name(*args, **kwargs):
    return "No name"


get_name(Person('Steve'))  # => Steve
get_name({'name': 'Tom'})  # => Tom
get_name(2)  # => No name


@multi
def area(shape):
    return shape.get('type')

@method(area, 'square')
def area(square):
    return square['width'] * square['height']

@method(area, 'circle')
def area(circle):
    return circle['radius'] ** 2 * 3.14159

@method(area)
def area(unknown_shape):
    raise Exception("Can't calculate the area of this shape")


area({'type': 'circle', 'radius': 0.5})  # => 0.7853975
area({'type': 'square', 'width': 1, 'height': 1})  # => 1
area({'type': 'rhombus'}) # => throws Exception

registry = {}

class MultiMethod(object):
    def __init__(self, name):
        self.name = name
```

```
        self.typemap = {}
    def __call__(self, *args):
        types = tuple(arg.__class__ for arg in args) # a generator expression!
        function = self.typemap.get(types)
        if function is None:
            raise TypeError("no match")
        return function(*args)
    def register(self, types, function):
        if types in self.typemap:
            raise TypeError("duplicate registration")
        self.typemap[types] = function

def multimethod(*types):
    def register(function):
        function = getattr(function, "__lastreg__", function)
        name = function.__name__
        mm = registry.get(name)
        if mm is None:
            mm = registry[name] = MultiMethod(name)
        mm.register(types, function)
        mm.__lastreg__ = function
        return mm
    return register

def run_async(func):
    """
        run_async(func)
            function decorator, intended to make "func" run in a separate
            thread (asynchronously).
            Returns the created Thread object

            E.g.:
            @run_async
            def task1():
                do_something

            @run_async
            def task2():
                do_something_too

            t1 = task1()
            t2 = task2()
            ...
            t1.join()
            t2.join()
    """
    from threading import Thread
    from functools import wraps

    @wraps(func)
    def async_func(*args, **kwargs):
        func_hl = Thread(target = func, args = args, kwargs = kwargs)
        func_hl.start()
        return func_hl

    return async_func

if __name__ == '__main__':
    from time import sleep

    @run_async
    def print_somedata():
        print 'starting print_somedata'
        sleep(2)
        print 'print_somedata: 2 sec passed'
        sleep(2)
```

```
        print 'print_somedata: 2 sec passed'
        sleep(2)
        print 'finished print_somedata'

    def main():
        print_somedata()
        print 'back in main'
        print_somedata()
        print 'back in main'
        print_somedata()
        print 'back in main'

    main()

# --------------------------------------------------------------------

def timeit(func):
    def timer(*args, **kwargs):
        start = time.time()
        ret = func(*args, **kwargs)
        time_taken = time.time() - start
        print("%s took %ss"%(func.func_name, time_taken))
        return ret
    return timer
timed_adder = timeit(adder)

# --------------------------------------------------------------------

import logging

def logit(func):
    def logger(*args, **kwargs):
        fancyargs = ['%s=%s'%(k,v) for k,v in zip(func.func_code.co_varnames, args)]
        ret = func(*args, **kwargs)
        logging.debug("%s given %s returned %s"%(func.func_name, fancyargs, ret))
        return ret
    return logger
logged_adder = logit(adder)

# --------------------------------------------------------------------

def synchronized(lock):
    def decorator(func):
        def wrapped(*args, **kwargs):
            lock.acquire()
            try:
                return func(*args, **kwargs)
            finally:
                lock.release()
        return wrapped
    return decorator

import threading.Lock

@synchronized(threading.Lock())
def increment_db(a):
    pass

# --------------------------------------------------------------------

def requires_int(func):
    def wrapper(*args, **kwargs):
        if not all((type(x)==int for x in args)):
            raise ValueError(func.func_name + " requires all int parameters")
        return func(*args, **kwargs)
    return wrapper
```

```
@requires_int
def adder(a, b):
   return a + b

# ----------------------------------------------------------------------

DEBUGGING = True
def requires_int(func):
   def wrapper(*args, **kwargs):
      if not all((type(x)==int for x in args)):
         raise ValueError(func.func_name + " requires all int parameters")
      return func(*args, **kwargs)
   return wrapper if DEBUGGING else func
@requires_int
def adder(a, b):
   return a + b

# ----------------------------------------------------------------------

def precondition(cond):
   def decorator(func):
      def wrapper(*args, **kwargs):
         argkeys = {k:v for k,v in zip(func.func_code.co_varnames, args) if k in cond.func_code.co_varnames}
         if not cond(**argkeys):
            raise ValueError("%s invalid %s" % (func.func_name, cond.func_code.co_varnames))
         return func(*args, **kwargs)
      return wrapper
   return decorator

@precondition(lambda e: e>0)
def pow(b, e):
   return math.pow(b, e)

# ----------------------------------------------------------------------

def postcondition(cond):
   def decorator(func):
      def wrapper(*args, **kwargs):
         ret = func(*args, **kwargs)
         if not cond(ret):
            fancyargs = ['%s=%s'%(k,v) for k,v in zip(func.func_code.co_varnames, args)]
            raise ValueError("%s invalid return for %s %s" % (func.func_name, fancyargs, kwargs))
         return ret
      return wrapper
   return decorator

@postcondition(lambda r: r>0)
def sqrt(a):
   return math.sqrt(a)
```

```
from functools import wraps

class IgnoreError(object):
   def __init__(self, errors, errorreturn = None, errorcall = None):
      self.errors = errors
      self.errorreturn = errorreturn
      self.errorcall = errorcall

   def __call__(self, function):
      @wraps(function)
      def returnfunction(*args, **kwargs):
         try:
            return function(*args, **kwargs)
         except Exception as error:
```

```
            if type(error) not in self.errors:
                raise error
            if self.errorcall is not None:
                self.errorcall(error, *args, **kwargs)
            return self.errorreturn
        return returnfunction

'''
better way:
from contextlib import suppress

def func():
    with suppress(ValueError):
        #doStuff

'''

if __name__ == "__main__":

    @ErrorIgnore([ValueError])
    def test(a, b=4):
        print(a)
        raise ValueError
        print(b)

    @ErrorIgnore([TypeError])
```

```
from threading import Thread
from functools import wraps

def run_async(func):
    """
        run_async(func)
            function decorator, intended to make "func" run in a separate
            thread (asynchronously).
            Returns the created Thread object

            E.g.:
            @run_async
            def task1():
                do_something

            @run_async
            def task2():
                do_something_too

            t1 = task1()
            t2 = task2()
            ...
            t1.join()
            t2.join()
    """

    @wraps(func)
    def async_func(*args, **kwargs):
        func_hl = Thread(target = func, args = args, kwargs = kwargs)
        func_hl.start()
        return func_hl

    return async_func


if __name__ == "__main__":
```

```
import time

@run_async
def test(x):
    print(x,"_start")
    time.sleep(x)
    print(x,"_done")

t1 = test(1)
t2 = test(2)
test(5)

t1.join()
t2.join()
```

xmlconfig.py

```
import xml.etree.ElementTree as ElementTree
import collections
import os
from .xmltodict import XmlDictConfig


class XmlListConfig(list):
    def __init__(self, aList):
        for element in aList:
            if element:
                if len(element) == 1 or element[0].tag != element[1].tag:
                    self.append(XmlDictConfig(element))
                elif element[0].tag == element[1].tag:
                    self.append(XmlListConfig(element))
            elif element.text:
                text = element.text.strip()
                if text:
                    self.append(text)


class XmlDictConfig(dict):

    def __init__(self, parent_element,valuetype=None):
        if parent_element.items():
            self.update(dict(parent_element.items()))
        for element in parent_element:
            if element:
                if len(element) == 1 or element[0].tag != element[1].tag:
                    aDict = XmlDictConfig(element,valuetype)
                else:
                    aDict = {element[0].tag: XmlListConfig(element)}
                if element.items():
                    aDict.update(dict(element.items()))
                self.update({element.tag: aDict})
            elif element.items():
                self.update({element.tag: dict(element.items())})
            else:
                text = element.text
                if valuetype is not None:
                    text = valuetype(element.text)
                self.update({element.tag: text})

def get_config(*configs):
    relist = []
    filenames = []
    for config in configs:
        if isinstance(config,(tuple,list)):
            base = os.path.basename(config[0])
            filenames.append(os.path.splitext(base)[0])
            root = ET.parse(config[0]).getroot()
            xmldict = XmlDictConfig(root,valuetype=config[1])
            relist.append(xmldict)
        else:
            base = os.path.basename(config)
            filenames.append(os.path.splitext(base)[0])
            root = ET.parse(config).getroot()
            relist.append(XmlDictConfig(root))
    configtuple = collections.namedtuple('Configs',filenames)
    return configtuple(*relist)
```

A2.2 Zentrale Station

Programmcode

```
# __standard libraries__
import os
import sys
import time
import datetime
import threading
import xml.etree.ElementTree as ET

# __external libraries:__
import pygame
from pygame.locals import *
# __user libraries:__
import p2p_framework_multiacc as p2p_framework
import pygamebaseclass as pygclass
import rgb
#import mysql_class
import pygbutton

# ____global variables____

# __get_config__
tree = ET.parse('config.xml')
root = tree.getroot()


# ____p2p_____
shutdown = [False]
router_ip = root.find("router_ip").text
name = root.find("p2p_name").text
type = root.find("p2p_type").text

# ___mysql___
db_host = root.find("db_host").text
db_user = root.find("db_user").text
db_passwd = root.find("db_passwd").text
db_db = root.find("db_db").text

# ____fkt______

# __p2phandler__
def printfunc(msg):
        print "Sendername:",msg["sendername"],"  Sendertype:",msg["sendertype"],"
        cmd:",msg["command"],"  data:",msg["data"]

# _________classes______

class pygbase(pygclass.pygamebase):

        count = 0
        updated = ["table"]

        buttons = {}
        control_b = []
        control_b_active = []

        addressbook = None



        def user_events(self):
```

```
                for event in pygame.event.get():
                        if event.type == QUIT:
                                self.quit_fkt()

                        # __Button__

                        for button in self.buttons:
                                if 'click' in self.buttons[button][0].handleEvent(event):
                                        print "You clicked: "+button

                                        if button[-5:] == "_open":
                                                currentTime = datetime.datetime.now()
                                                refTime =
currentTime.hour*3600+currentTime.minute*60-120

                                                msg = "machine_"+button[-6:-5]+"&"+str(refTime)
                                                recv = button[:-5]

                                                self.p_p2p.sendmessage('TCP', recv,'schleife',
'REQUEST', msg, 2)

                                        elif button[-7:] == "_unlock":
                                                recv = button[:-7]
                                                self.p_p2p.sendmessage('TCP', recv,'schleife', 'UNLOCK',
"", 2)

                                        elif button == "hi":
                                                print "test"

        #self.buttons["hi"][0]._propSetRect(pygame.Rect(30,800,60,30))


        def add_button(self,rect,caption,name=None,visible=True,surfobj=None):
                button = pygbutton.PygButton(rect, caption)
                if visible == False:
                        button._propSetVisible(False)
                if name is None:
                        name = caption
                if surfobj is None:
                        surfobj = self.gameDisplay
                self.buttons[name] = (button,surfobj)

        def init_material(self):

                for image in imagelist:
                        imagepath = "./material/"+image
                        imagevar = pygame.image.load(imagepath).convert()
                        self.material[image] = imagevar


                # additional operations
                self.material["overview.jpg"] = pygame.transform.scale(self.material["overview.jpg"],
(self.w_size[0],self.w_size[1]))
                self.material["overview.jpg"].set_alpha(75)


                # __Button__
                self.add_button((50, 50, 60, 30),"test",name="hi",visible=True)

        def draw_table(self):

                #header
                ydisplace = self.job_table_x
                xdisplace = self.job_table_y
                self.message_to_screen_left("Auftrag:", rgb.black, xdisplace, ydisplace, "small")
                xdisplace += 120
```

```
                self.message_to_screen_left("Deadline:", rgb.black, xdisplace, ydisplace, "small")
                xdisplace += 120
                self.message_to_screen_left("Prio:", rgb.black, xdisplace, ydisplace, "small")

                #content
                # for stuff in stuff: do stuff

        def draw_button(self):
                for button in self.buttons:
                        self.buttons[button][0].draw(self.buttons[button][1])
                        bRect = self.buttons[button][0]._propGetRect()
                        self.dirty_rect.append(bRect)

        def draw_control(self):
                xdisplace = self.c_table_x #+ 80
                ydisplace = self.c_table_y + 35
                for button in self.control_b_active:
                        self.message_to_screen_left(button, rgb.black, xdisplace, ydisplace, "tiny")
                        ydisplace += 40
                hrect = pygame.Rect(self.w_size[0]/2,0,self.w_size[0]/2,self.w_size[1])
                self.dirty_rect.append(hrect)

        def update(self):
                #database refresh
                refreshrate = 10

                if self.count >= (refreshrate*float(self.current_fps)) and self.count >=refreshrate:
                        self.count = 0
                        print "update from database"

                        # _____buttons_____
                        try:
                                self.addressbook = self.p_p2p.get_address_book()
                        except:
                                print "Error in Buttonupdate"

                        if self.addressbook is not None:

                                control_b_help = self.control_b[:]

                                xdisplace = self.c_table_x+80
                                ydisplace = self.c_table_y+30 + 40 * len(self.control_b)

                                for client in self.addressbook:
                                        if client[1] == "schleife":
                                                if client[0] not in self.control_b:
                                                        self.add_button((xdisplace, ydisplace, 60,
30),caption="open",name=client[0]+"_open",visible=True)
                                                        self.add_button((xdisplace+80, ydisplace, 60,
30),caption="unlock",name=client[0]+"_unlock",visible=True)
                                                        ydisplace += 40
                                                        self.control_b.append(client[0])
                                                        self.control_b_active.append(client[0])
                                                        if "control" not in self.updated:
                                                                self.updated.append("control")

                                                else:
                                                        if client[0] not in self.control_b_active:
                                                                self.control_b_active.append(client[0])

        self.buttons[client[0]+"_open"][0]._propSetVisible(True)

        self.buttons[client[0]+"_unlock"][0]._propSetVisible(True)
                                                        control_b_help.remove(client[0])

                                for button in control_b_help:
```

```
                                        self.control_b_active.remove(button)
                                        self.buttons[button+"_open"][0]._propSetVisible(False)
                                        self.buttons[button+"_unlock"][0]._propSetVisible(False)

                self.count += 1

        def draw(self):
                #self.dirty_rect.append(self.fullframe)

                # background
                self.gameDisplay.fill(rgb.dark_grey)
                self.gameDisplay.blit(self.material["overview.jpg"], (0,0))

                # testing surfaces_____________________
                self.job_table_surf.fill(rgb.white)
                self.job_table_surf.set_alpha(75)
                self.gameDisplay.blit(self.job_table_surf,(self.job_table_x,self.job_table_y))
                self.dirty_rect.append((self.job_table_x,self.job_table_y,self.job_table_dx,self.job_table_dy))


                # testing surfaces_____________________
                if "table" in self.updated:
                        self.draw_table()
                        self.updated.remove("table")
                        help_rect = pygame.Rect(0,0,(self.w_size[0]/2),self.w_size[1])
                        self.dirty_rect.append(help_rect)

                elif "control" in self.updated:
                        self.draw_control()
                        self.updated.remove("control")
                        help_rect = pygame.Rect((self.w_size[0]/2),0,(self.w_size[0]/2),self.w_size[1])
                        self.dirty_rect.append(help_rect)

                self.draw_button()

# ______main_________

def main():

        # __p2p setup__
        p2p = p2p_framework.P2P_Interface(shutdown,name,type, router_ip)

        # __p2phandler__
        p2p.add_handler('PRINT', printfunc)

        # __mysqldb__
        mydb = mysql_class.sqldb(db_host, db_user, db_passwd, db_db)

        # __pygame__
        gamebase = pygbase(shutdown=shutdown, caption="Security Station", p_p2p=p2p)

        #start mainloop
        gamebase.GameLoop()

if __name__ == "__main__":
        main()
```

A2.3 Transportsystem

Programmcode

```
from xml.dom.minidom import parse
import xml.dom.minidom
import sys
import os
import time
import threading
import time
import datetime
import xml.etree.ElementTree as ET
import signal
from subprocess import call
#-------------------------------- project includes ----------------------------------#
sys.path.append('/home/pi/Desktop/LabControlSoftware-master/common')

from p2p_framework import P2P_Interface
from mysql_class import sqldb
from shuttle import Shuttle

from shuttledriver import Driver
import shuttlegui
from shuttlegui import gui
#------------------------------------------------------------------------------------#
#---------------------------------------- global variables --------------------------#
# getting Configuration data from config.xml file
configs_file = ET.parse("/home/pi/Desktop/LabControlSoftware-master/config.xml")
configs = configs_file.getroot()
# general configs
common_configs = configs.find('common_configs')
router_ip = common_configs.find('router_ip').text
trnasportTime = int(common_configs.find('transport_time').text) * 60 # in seconds

# configs specefic for shuttles
shuttle_configs = configs.find("shuttle_configs")
name = shuttle_configs.find('name').text
Type = shuttle_configs.find('type').text

#get database configs
db_configs = configs.find("database_configs")
host = db_configs.find('host').text
user = db_configs.find('user').text
passwd = db_configs.find('passwd').text
db = db_configs.find('db_name').text

mydb = sqldb(host,user,passwd,db)

global_shuttle_container = []

module = {}

machines = {}

global_shuttle_driver = []

global_shuttle_gui = []

shuttle_container_lock = threading.Lock()

def getCurrentTimeInSeconds(self):
    currenttime = datetime.datetime.now()
    currenttimeinseconds = ( currenttime.hour * 60 + currenttime.minute ) * 60
    return currenttimeinseconds
```

```
def shuttle_status(shutdown,interface,transportTime):
        got_job_status = [False]
        status_busy = False
        shuttle_created = False
        global global_shuttle_container
        Interface = interface
        Machines_dic = {}
        machine4 = {}
        general_information = {'KA_Nummer':'','EndTime':'','Priority':0}
        EndTime = ''
        AV_Nummern_dic = {}
        Arbeitsvorgaenge_dic = {}

    def get_job_from_db(message):
        print "get_job_from_db....general_information: ",general_information
        print "get_job_from_db....Arbeitsvorgaenge_dic: ",Arbeitsvorgaenge_dic
        print "get_job_from_db .... got_job_status: ",got_job_status[0]
        print "got response from commander \n"
                print message
        if message['data'] != "Error":
                        try:
                                print "getting data from database server ....\n"
                                KA_Nummer_cmd = message['data']
                        sql = "SELECT * FROM kundenauftrag WHERE KA_Nummer = %s " %
(KA_Nummer_cmd)
                        data = mydb.sqlquery(sql)
                                general_information['KA_Nummer'] = data["rows"][0]["KA_Nummer"]
                        general_information['EndTime']=  data["rows"][0]["KA_Termin"]
                        general_information['Priority'] = data["rows"][0]["KA_Prio"]
                        print "got auftrag data: ",general_information
                        global_shuttle_gui[0].set_nr(general_information['KA_Nummer'])
                                Artikelnummern = {}
                                Artikelnummern['UB']= data["rows"][0]["UB"]
                                Artikelnummern['RF']= data["rows"][0]["RF"]
                                Artikelnummern['CH']= data["rows"][0]["CH"]
                                print "Artikelnummern: ",Artikelnummern

                                for Artikel in Artikelnummern:
                                        sql = "SELECT * FROM arbeitsplan WHERE
arbeitsplan.Artikelnummer = %s " % (Artikelnummern[Artikel])
                                        data = mydb.sqlquery(sql)
                                        AV_Nummern_dic[Artikel] = data["rows"][0]['AV_Nummer']
                                print "AV_nummer from Arbeitsplan : ", AV_Nummern_dic

                                for nummer in AV_Nummern_dic:
                                        sql = "SELECT * FROM arbeitsvorgang WHERE
arbeitsvorgang.AV_Nummer = %s " % (AV_Nummern_dic[nummer])
                                        data = mydb.sqlquery(sql)
                                        print "Arbeitsvorgaenge: ",data
                                        Arbeitsvorgaenge_dic[nummer]                                    =
{'AV_Zykluszeit':data["rows"][0]['AV_Zykluszeit'],'FM_Nummer':data["rows"][0]['FM_Nummer']}

                                Machines_dic['machine_1']                                               =
{'Name':'machine_1','ProcessingTime':Arbeitsvorgaenge_dic['RF']['AV_Zykluszeit']}
                                Machines_dic['machine_2']                                               =
{'Name':'machine_2','ProcessingTime':Arbeitsvorgaenge_dic['UB']['AV_Zykluszeit']}
                                print type(Arbeitsvorgaenge_dic['UB']['AV_Zykluszeit'])
                                print Arbeitsvorgaenge_dic['UB']['AV_Zykluszeit']
                                Machines_dic['machine_2'] = {'Name':'machine_2','ProcessingTime':'001'}
                                Machines_dic['machine_3']                                               =
{'Name':'machine_3','ProcessingTime':Arbeitsvorgaenge_dic['CH']['AV_Zykluszeit']}
                                print "Machines_dic (get_job_from_db) :",Machines_dic

                                machine4 = {'Name':'machine_4','ProcessingTime':'001'} # montagestation
                                print "machine4 (get_job_from_db) :",machine4
```

```
                                print "Got data from the database !!\n"
                                got_job_status[0] = True
                                print "got_job_status: ",got_job_status[0]
                                except:
                                   #currentJob = 0
                                   got_job_status[0] = False
                                   #errormsg =str(KA_Nummer)+" reset"
                                  # Interface.sendmessage('TCP', "cmd", "Raspberry", "Error", errormsg)
                                   print "db Connection Error"
        else:
                                time.sleep(10)
                                got_job_status[0] = False

        Interface.add_handler("newJob", get_job_from_db)
        Interface.add_handler("retry", get_job_from_db)


        while not shutdown[0]:
        time.sleep(3)
        print "status_busy: ",status_busy
        print "got_job_status: ",got_job_status[0]
              if not status_busy:
                                if not got_job_status[0] :
                                # I have to check if the commander in address book or not
                                        print "sending request to the cmd ..!!\n"
                                        Interface.sendmessage('TCP', "cmd", "Raspberry", "getJob", "0",1)

              # a request should be sent to the cmd to get  KA_Nummer

              #Priority=2
                                #Machines_dic ={'machine_2':{'Name':'machine_2','ProcessingTime':'001'}}
                                machine4 = {'Name':'machine_4','ProcessingTime':'001'} # montagestation
                                print "Machines_dic (looooooooooooop) : " , Machines_dic
                                print "machine 4 (looooooooooooop): ",machine4
              # EndTime = "15:30"
              #print " expected End Time >> ",EndTime
              # I am passing 2 to contract number (contract number should be in the range [0 - 9])
              #general_information['EndTime'] = "19:57"
              if  got_job_status[0]:
                    print
"=========================================================================="
                    print "END TIME",general_information['EndTime']
                    global global_shuttle_driver
                                myShuttle                                                                                                =
Shuttle(shutdown,general_information['Priority'],general_information['EndTime'],Machines_dic,machine4,Interface.a
dd_handler,Interface.sendmessage,Interface.get_address_book,trnasportTime,2,global_shuttle_driver)
                                print "myShuttle: "+str(myShuttle)+"\n"
                                if myShuttle:
                                        myShuttle.addHandlerFunc('PRINT', myShuttle.print_message)

myShuttle.addHandlerFunc('SCHEDULED',myShuttle.get_EDF_response)
                                        myShuttle.addHandlerFunc('SCHEDULEFAIL',myShuttle.schedule_fail)

myShuttle.addHandlerFunc('SCHEDULEDM4',myShuttle.get_machine_4_response)

myShuttle.addHandlerFunc('SCHEDULEFAILM4',myShuttle.schedule_fail)
                                        myShuttle.addHandlerFunc('TASK_DONE',myShuttle.task_done)
                                        shuttle_container_lock.acquire()
                                        global_shuttle_container.append(myShuttle)
                                        shuttle_container_lock.release()
                                        print "shuttle container: " ,global_shuttle_container
                                        print "shuttle added to the container ...!! "
                                        status_busy = True

              time.sleep(5)
        print" ------------------------------------------------------------"
```

```
        if(got_job_status[0] and status_busy):

            if(myShuttle.getStatus()):
                        print "the  current Contract finished .....\n"
                        print "deleting the current shuttle object \n"
                        shuttle_container_lock.acquire()
                        del global_shuttle_container[0]
                        shuttle_container_lock.release()
                        del myShuttle
                        print " getting new Contract from database \n"
                        status_busy = False
                        got_job_status[0] = False

        print "shuttle container: " ,global_shuttle_container


def main():

        try:
#------------------------------------ INITIALIZATIONS ----------------------------------#
                    global shutdown
                    global global_shuttle_container
                    global global_shuttle_driver
                    global global_shuttle_gui

         print "main statrted ....."
                    print "Process ID : ",os.getpid()
                    shutdown = [False]
                    sql = "SELECT * FROM module"
                    module = mydb.sqlquery(sql)
                    #print module
                    #print "----------------------------------------"
                    for i in range(len(module["rows"])):
                                temp_type =  module["rows"][i]["type"]
                                if temp_type == "maschine":
                                            machines[module["rows"][i]["name"]]                                                    =
{"Beschreibung":module["rows"][i]["Beschreibung"],"FM_Nummer":module["rows"][i]["FM_Nummer"]}

                    #print machines



                    print "Configuration data:"
                    print "\t router_ip: ",              router_ip
                    print "\t transport Time: ",trnasportTime

                    print "\t name: ",     name
                    print "\t type: "       ,Type
                    print"\n"

                    myInterface = P2P_Interface(shutdown,name,Type,router_ip)
                    time.sleep(10)
                    print myInterface

                    #start shuttle status thread  (to track the status of the shuttle [busy or free])_
                    t_shuttle_status                    =                    threading.Thread(target                    =
shuttle_status,args=(shutdown,myInterface,trnasportTime,))
                    t_shuttle_status.start()

                    # start driver thread
                    t_shuttle_driver = Driver(shutdown,myInterface,global_shuttle_container)
                    t_shuttle_gui = gui(global_shuttle_container)

                    global_shuttle_driver.append(t_shuttle_driver)
```

```
                global_shuttle_gui.append(t_shuttle_gui)


                # use a infinite loop for the user prompting
                while not shutdown[0]:
                        time.sleep(5)

                        '''# save the users input in a variable
                        input_text = raw_input('>>>')

                        # if the user enters 'EXIT', the inifinte while-loop quits and the
                        # program can terminate
                        if input_text == 'EXIT':
                                shutdown[0] = True

                        # if the user enters 'ADDR', the address book will be printed in the
                        # console
                        elif input_text == 'ADDR':
                                address_book = myInterface.get_address_book()
                                print_address_book(address_book)


                        elif input_text.startswith('CANCEL'):
                                machineslist = myShuttle.get_required_machines_list()
                                for machine in machineslist:
                                        myShuttle.sendMessageFunc('TCP',machine,'','CANCEL','hello')

        myShuttle.sendMessageFunc('TCP',machine4['Name'],'','CANCEL','hello')
                        elif input_text.startswith('TIME'):
                                print "current time: ",datetime.datetime.now()

                        elif input_text.startswith('TARGET'):
                                global_shuttle_container[0].get_target_list()
                                print "global_shuttle_container[0]",global_shuttle_container
                        '''

        except :
                print ("Error happened in the main function")
                shutdown[0] = True
                sys.exit()




if __name__ == "__main__":
        main()
```

Weichen

Zusätzlich zu den Shuttlen existieren Weichen, welche die Shuttle in eine Station fahren lassen. Dieses Signal erhalten diese hierbei direkt von den Shuttlen.

Programmcode

```
# __standard libraries__
from __future__ import print_function, unicode_literals, division, absolute_import
import os
import sys
import collections
import pprint
import time
import datetime
import threading
from functools import partial, wraps
import logging

# https://sourceforge.net/p/raspberry-gpio-python/wiki/
import RPi.GPIO as GPIO

# __user libraries:__
from libraries.p2p_framework import P2P_Interface
from libraries.xmlconfig import get_config
# import libraries.mysql_class
from libraries.decorator import echo,run_async

# --global--
DEBUG = 1
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

class loopControl(threading.Thread):

    setHIGH = GPIO.HIGH
    setLOW = GPIO.LOW
    FALLING = GPIO.RISING
    RISING = GPIO.FALLING
    BOTH = GPIO.BOTH

    def __init__(self,shutdown,gpio,p2p,config):
        threading.Thread.__init__(self)

        # --variables--
        self.shutdown = shutdown
        self.p2p = p2p

        self.timer = threading.Timer(5.0 ,self._mainLoop)
        self.stationflag = False
        self.mainloopflag = False
        self.jumpflag = False
        self.jumpinsert = None

        self.input = gpio["in"]
        self.output = gpio["out"]
        self.FMlist = [v for v in config["FMlist"].values()][0]

        self.jobqueue = collections.deque()

        # --locks--
        self.appendlock = threading.Lock()
```

```
    # --p2p_handler--
    self.p2p.add_handler('REQUEST', self.requestHandler)
    self.p2p.add_handler('UNLOCK', self.stationunlock)
    self.p2p.add_handler('CMD', self.cmdHandler)

    # --actions--
    self.shuttleIn = [partial(self._waitingPoint, output=self.output["StopW1"], target="close"),
                      self._openW1,
                      partial(self._waitingPoint, output=self.output["StopW1"], target="open"),
                      partial(self._wait_for_edge, input=self.input["EnterW1"], target=self.FALLING),
                      partial(self._waitingPoint, output=self.output["StopW1"], target="close"),
                      partial(self._wait_for_edge, input=self.input["ExitW1"], target=self.BOTH),
                      self._closeW1,
                      self.jump]

    self.shuttleOut = [partial(self._setstationflag, True),
                       partial(self._waitingPoint, output=self.output["StopW1"], target="close"),
                       self._openW2,
                       self._unlock,
                       partial(self._waitingPoint, output=self.output["StopSt"], target="open"),
                       partial(self._wait_for_edge, input=self.input["StopSt"], target= self.FALLING),
                       partial(self._waitingPoint, output=self.output["StopSt"], target="close"),
                       partial(self._wait_for_edge, input=self.input["ExitW2"], target=self.BOTH),
                       partial(self._setstationflag, False),
                       self._closeW2,
                       self.jump]

    self.shuttleMain = [partial(self._waitingPoint, output=self.output["StopW1"], target="open"),
                        partial(self._wait_for_edge, input=self.input["EnterW1"], target=self.FALLING),
                        partial(self._waitingPoint, output=self.output["StopW1"], target="close"),
                        partial(self._wait_for_edge, input=self.input["ExitW2"], target=self.BOTH),
                        self._setmainloopflag,
                        self.jump]

    self.reset = [partial(self._waitingPoint, output=self.output["StopW1"], target="close"),
                  partial(self._waitingPoint, output=self.output["StopSt"], target="close"),
                  self._closeW1,
                  self._closeW2,
                  self.jump]

    # --setup--
    self.GPIOsetup()
    self._setTimeStamp()

    # --start threads--
    self.inputthread = self.inputloop()
    self.start()


@run_async
def inputloop(self):
    while not self.shutdown[0]:
        if GPIO.input(self.input["StopSt"]) == GPIO.LOW and not self.stationflag:
            self._stationlock()

        if GPIO.input(self.input["EnterW1"]) == GPIO.LOW and not self.timer.is_alive() and not self.mainloopflag:
            self.mainloopflag = True
            self.timer = threading.Timer(5.0 ,self._mainLoop)
            self.timer.start()
        elif GPIO.input(self.input["EnterW1"]) == GPIO.HIGH and self.timer.is_alive():
            self.timer.cancel()
            self.mainloopflag = False

        time.sleep(0.2)

@echo(DEBUG)
```

```
def _setstationflag(self,val):
    self.stationflag = val


def _setmainloopflag(self):
    self.mainloopflag = False


def _setTimeStamp(self):
    self.TimeStamp = datetime.datetime.now()


@echo(DEBUG)
def stationunlock(self,msg):
    with self.appendlock:
        self.jobqueue.extend(self.shuttleOut)


@echo(DEBUG)
def _stationlock(self):
    GPIO.output(self.output["SperrSt"], self.setLOW)
    self.stationflag = True


@echo(DEBUG)
def _unlock(self):
        GPIO.output(self.output["SperrSt"], self.setHIGH)


def _exitHandler(self):
    self.shutdown[0] = True
    logger.info("____Program stopped____")


@echo(DEBUG)
def cmdHandler(self,msg):
    logger.debug("New Command: {}".format(msg["data"]))
    if msg["data"].startswith("openW1"):
        self.jumpinsert = [partial(self._waitingPoint, output=self.output["StopW1"], target="close"),
                           self._openW1,
                           partial(self._waitingPoint, output=self.output["StopW1"], target="open"),
                           self.jump]
        self.jumpflag = True

    elif msg["data"].startswith("openW2"):
        self.jumpinsert = self.shuttleOut
        self.jumpflag = True

    elif msg["data"].startswith("closeW1"):
        self.jumpinsert = [partial(self._waitingPoint, output=self.output["StopW1"], target="close"),
                           self._closeW1,
                           self.jump]
        self.jumpflag = True

    elif msg["data"].startswith("closeW2"):
        self.jumpinsert = [partial(self._waitingPoint, output=self.output["StopSt"], target="close"),
                           self._closeW2,
                           self.jump]
        self.jumpflag = True

    elif msg["data"].startswith("reset"):
        self.jumpinsert = self.reset
        self.jumpflag = True

    elif msg["data"].startswith("clear"):
        self.jobqueue = collections.deque(self.reset)
```

```
    elif msg["data"].startswith("unlock"):
        self.jumpinsert = [partial(self._setstationflag,True),
                           self._unlock,
                           partial(self._wait_for_edge, input=self.input["StopSt"], target= self.FALLING),
                           partial(self._setstationflag,False),
                           self.jump]
        self.jumpflag = True

    elif msg["data"].startswith("reboot"):
        if "force" not in msg["data"]:
            while self.jobqueue:
                time.sleep(0.2)
        self.jobqueue = collections.deque(self.reset)
        time.sleep(2)
        self._exitHandler()
        os.system("sudo reboot")


@echo(DEBUG)
def requestHandler(self,msg):
    try:
        noTime = False
        sendtime = msg["time"]
    except KeyError:
        noTime = True
        sendtime = self.TimeStamp

    # logger.debug("{} | {} | {}".format(sendtime,self.TimeStamp,sendtime>self.TimeStamp))
    if sendtime > self.TimeStamp or noTime:
        currentTime = datetime.datetime.now()
        refTime = currentTime.hour*3600+currentTime.minute*60

        tmpmsg = msg["data"].split("&")

        try:
            if tmpmsg[0] in self.FMlist and int(tmpmsg[1]) <= refTime:
                with self.appendlock:
                    self.jobqueue.extend(self.shuttleIn)
            else:
                raise IndexError

        except IndexError:
            with self.appendlock:
                self.jobqueue.extend(self.shuttleMain)


@echo(DEBUG)
def _mainLoop(self):
    if self._openW1 not in self.jobqueue:
        self._setTimeStamp()
        with self.appendlock:
            self.jobqueue.extend(self.shuttleMain)


@echo(DEBUG)
def GPIOsetup(self):

    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)

    # --inputs--
    inputlist = [k for k in self.input.values()]
    GPIO.setup(inputlist, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    # --outputs--
```

```
    # setup weichensperre
    GPIO.setup([self.output["SperrW1"],self.output["SperrW2"]],GPIO.OUT,initial=GPIO.HIGH)

    # Haltepunkte
    GPIO.setup([self.output["StopSt"],self.output["StopW1"]], GPIO.OUT,initial=GPIO.HIGH)

    # Setup Weiche 1
    if GPIO.input(self.input["OpenW1"]):
        GPIO.setup(self.output["W1"], GPIO.OUT, initial=GPIO.LOW)
    else:
        GPIO.setup(self.output["W1"], GPIO.OUT, initial=GPIO.HIGH)

    # setup Weiche 2
    if GPIO.input(self.input["OpenW2"]):
        GPIO.setup(self.output["W2"], GPIO.OUT, initial=GPIO.LOW)
    else:
        GPIO.setup(self.output["W2"], GPIO.OUT, initial=GPIO.HIGH)

    # setup Station
    GPIO.setup(self.output["StopSt"], GPIO.OUT,initial=GPIO.HIGH)
    GPIO.setup(self.output["SperrSt"], GPIO.OUT,initial=GPIO.HIGH)

    # startpos
    GPIO.output([self.output["SperrW1"],self.output["SperrW2"]],GPIO.LOW)
    GPIO.output([self.output["W2"],self.output["W1"]],GPIO.HIGH)
    time.sleep(1)
    while GPIO.input(self.input["CloseW1"]) != GPIO.LOW or GPIO.input(self.input["CloseW2"]) != GPIO.LOW:
        time.sleep(0.1)
    GPIO.output([self.output["SperrW1"],self.output["SperrW2"]],GPIO.HIGH)


@echo(DEBUG)
def _openW1(self):
    GPIO.output(self.output["SperrW1"],self.setLOW)
    GPIO.output(self.output["W1"],self.setLOW)

    while GPIO.input(self.input["OpenW1"]) != GPIO.LOW:
        time.sleep(0.1)

    GPIO.output(self.output["SperrW1"],self.setHIGH)


@echo(DEBUG)
def _openW2(self):
    GPIO.output(self.output["SperrW2"],self.setLOW)
    GPIO.output(self.output["W2"],self.setLOW)

    while GPIO.input(self.input["OpenW2"]) != GPIO.LOW:
        time.sleep(0.1)

    GPIO.output(self.output["SperrW2"],self.setHIGH)


@echo(DEBUG)
def _closeW1(self):
    GPIO.output(self.output["SperrW1"],self.setLOW)
    GPIO.output(self.output["W1"],self.setHIGH)
    while GPIO.input(self.input["CloseW1"]) != GPIO.LOW:
        time.sleep(0.1)
    GPIO.output(self.output["SperrW1"],self.setHIGH)


@echo(DEBUG)
def _closeW2(self):
    GPIO.output(self.output["SperrW2"],self.setLOW)
    GPIO.output(self.output["W2"],self.setHIGH)
```

```
    while GPIO.input(self.input["CloseW2"]) != GPIO.LOW:
        time.sleep(0.1)
    GPIO.output(self.output["SperrW2"],self.setHIGH)


@echo(DEBUG)
def _wait_for_edge(self,input,target,ttime=1000):
    waiting = None
    while waiting is None and not self.jumpflag:
        stime = time.time()
        waiting = GPIO.wait_for_edge(input, target, timeout=ttime)
        # waiting = GPIO.wait_for_edge(input, target)
        timeit = time.time()-stime
        if timeit < 0.09:
            waiting = None

    if target == GPIO.RISING:
        while GPIO.input(input) == GPIO.LOW:
            time.sleep(0.1)
    elif target == GPIO.FALLING:
        while GPIO.input(input) == GPIO.HIGH:
            time.sleep(0.1)


@echo(DEBUG)
def _waitingPoint(self,output,target):
    if target == "open":
        GPIO.output(output,GPIO.LOW)
    elif target == "close":
        GPIO.output(output,GPIO.HIGH)
    else:
        raise ValueError


@echo(DEBUG)
def jump(self):
    self.jumpflag = False
    if self.jumpinsert is not None:
        with self.appendlock:
            self.jobqueue.extendleft(reversed(self.jumpinsert))
        self.jumpinsert = None


@echo(DEBUG)
def run(self):
    while not self.shutdown[0]:
        if self.jobqueue:
            nextjob = self.jobqueue.popleft()
            # input("PAUSE: {}".format(nextjob))
            try:
                funcname = nextjob.__name__
            except:
                funcname = "somefunc"
            if (not self.jumpflag and funcname != "jump") or funcname == "jump":
                logger.debug("_______Start:{}_________".format(funcname))
                if DEBUG:
                    starttimer = time.time()
                nextjob()
                logger.debug("_______Done:{}_________".format(time.time()-starttimer))
        elif self.jumpflag:
            self.jump()
        else:
            time.sleep(0.1)


def main():
```

```
    shutdown = [False]

    file= os.path.abspath(__file__)
    filefolder = os.path.dirname(file)
    configpath = os.path.join(filefolder,'xml/config.xml')
    gpiopath = os.path.join(filefolder,'xml/GPIO.xml')
    config,GPIOconfig = get_config(configpath,(gpiopath,int))

    p2p = P2P_Interface(shutdown,
                        name = config["p2p"]["name"],
                        type = config["p2p"]["type"],
                        router_address = config["p2p"]["router_ip"])
    # p2p.logger.setLevel(20)

    loop = loopControl(shutdown,GPIOconfig,p2p,config)

    while not shutdown[0]:
        try:
            uin = input()
            if uin == "print":
                print("_____")
                print(loop.jobqueue)
                print("_____")
        except:
            loop._exitHandler()
            raise

if __name__ == "__main__":
    main()
```

A2.4 Arbeitsstation

Programmcode

```
#---------------------------------- general includes ----------------------------------#
import os
import sys
import threading
import xml.etree.ElementTree as ET

#---------------------------------- project includes ----------------------------------#
from machine_4 import Machine
sys.path.append('../../common')
from p2p_framework import P2P_Interface

def main():

	try:
		#---------------------------------- INITIALIZATIONS ----------------------------------#
		print "main statrted ....."
		print "process ID: ", os.getpid()
		shutdown = [False]

		# Open XML document using minidom parser
		# getting Configuration data from config.xml file
		configs_file = ET.parse("../../config.xml")
		configs = configs_file.getroot()
		# general configs
		common_configs = configs.find('common_configs')
		router_ip = common_configs.find('router_ip').text
		trnasportTime = int(common_configs.find('transport_time').text) * 60 # in seconds
		# machine specific configs
		machine_configs = configs.find('machine_configs')
		name = machine_configs.find('name').text
		Type = machine_configs.find('type').text
		print "Configuration data:"
		print "\t router_ip: ",		router_ip
		print "\t transport Time: ",trnasportTime

		print "\t name: ",	name
		print "\t type: "	,Type

		myInterface = P2P_Interface(shutdown,name,Type,router_ip)
		myScheduler =
Machine(trnasportTime,myInterface.add_handler,myInterface.sendmessage,shutdown)
		#adding hanlder to the requests
		myScheduler.addHandlerFunc('ADD', myScheduler.taskArrived)
		myScheduler.addHandlerFunc('CANCEL', myScheduler.cancelRequest)

		#thread to keep track of the time to start the next task
		t_handleTasks = threading.Thread( target = myScheduler.tasksHandler)
		t_handleTasks.start()

		#start update backup data thread
		t_updateData = threading.Thread(target = myScheduler.update_backup_data)
		t_updateData.start()

		#start gui handler function
		#t_gui = threading.Thread(target = myScheduler.gui_funtion)
		#t_gui.start()


		##run function to set create gui flag to 1 every 2 seconds for testing
		#t_set_create_row_gui = threading.Thread(target = myScheduler.set_create_row_gui_flag)
		#t_set_create_row_gui.start()
```

```
                #------------------------------------ END OF INITIALIZATIONS ----------------------------------#

                #------------------------------------ START THE MAIN LOOP  ------------------------------#
                while not shutdown[0]:
                        # save the user's input in a variable
                        input_text = raw_input('>>>')

                        #if the user enters 'EXIT', the inifinte while-loop quits and the
                # program can terminate
                        if input_text == 'EXIT':
                                shutdown[0] = True
                                del myInterface
                        elif input_text == 'PRINTQUEUE':
                                myScheduler.print_elements_queue();
                        elif input_text == 'PRINTFREESLOTS':
                                myScheduler.printSlots()
                        elif input_text.startswith('TIME'):
                                print "current time: ",datetime.datetime.now()
        except KeyboardInterrupt:
                shutdown = [True]
                sys.exit()


if __name__ == "__main__":
        main()

#--------------------------------- general includes ----------------------------------#
import datetime
import operator
import sys
import signal
import os
import time
import threading
from threading import Timer
from subprocess import call
from xml.dom.minidom import parse
import xml.dom.minidom
import xml.etree.ElementTree as ET
from Tkinter import *
import copy

#--------------------------------- project includes -----------------------------------#
#sys.path.append("../../tests")
#from gpiotest import gpio_Interface # works only for beagle bone
from freeTimeSlots import FreeTimeSlot
sys.path.append('../includes')
import machine_queue
from machine_queue import queueElement
import timeTest
from timeTest import myTime

class Machine():


        def __init__(self,trnsportTime,addHandler,sendMessage,shutdown):

                self.sendMessageFunc = sendMessage
                self.addHandlerFunc = addHandler
                self.shutdown = shutdown
                self.__create_row_gui = False
                self.__gui_rows_counter = 0
                self.__new_task_name = ''
                self.__backup_file = 0
                self.__backup_file_lock = threading.Lock()
```

```
                print "current time :"+ str(datetime.datetime.now()) + '\n'
                self.__scheduleFail = False
                self.__scheduleSuccess = False
                self.__freeTimeSlots = 0
                #self.__gpioInterface = gpio_Interface()# works only for beagle bone
                #self.__gpioInterface.clearpins()# works only for beagle bone
                self.__transportationTime = trnsportTime
                self.__newTask = {}
                self.__taskDic={}
                self.__FreeSlots=[]
                self.__lastArrivingTask = ""
                self.__transportationTime = trnsportTime
                signal.signal(signal.SIGINT,self.kill_signal_handler)

        # handler will be executed when "Ctl+C" will be pressed
        def kill_signal_handler(self,signal,frame):
                #self.__gpioInterface.clearpins()
                print "you pressed ctrl+c !!"
                call(["kill","-9",str(os.getpid())])

# function to keep track of the time to start the next task
        def tasksHandler(self):
                while not self.shutdown[0]:
                        currentTime = self.getCurrentTimeInSeconds()
                        for key,value in self.__taskDic.iteritems():
                                if(value._StartTime <= currentTime and value._Status =='Scheduled'):
                                        # should hier tell the machine which program to run
                                        value._Status ='Running'
                                        print "current time :",  datetime.datetime.now()
                                        print "value._ProcessingTime :", value._ProcessingTime/60
                                        print "task started ",value._Name
                                        print "contract number: ", value._ContractNumber
                                        #self.__gpioInterface.outputval(value._ContractNumber)   #   works
only for beagle bone
                                        t=Timer(value._ProcessingTime,self.timeout,[value._Name])       #
argument has to be passed as an array
                                        t.start()
                                        print "Timer started"

                        time.sleep(30)

        # will be executed when the task finished (could be used to call some output routines to check if the machine
already done)
        def timeout(self,name):
                print "task finished ....."
                #self.__gpioInterface.clearpins() # works only for beagle bone
                msg={}
                msg['sendername']= name
                self.__removeTask(msg)

        def __del__(self):
                print "machine destructor....."


        def update_backup_data(self):
                print "update backup thread started ....."
                while not self.shutdown[0]:
                        print "update backup called....."

                        self.__backup_file_lock.acquire()
                        self.__backup_file = ET.parse('backup_machine4.xml')
                        root = self.__backup_file.getroot()
                        currentTime = root.find('current_time')
                        currentTime.text = str(self.getCurrentTimeInSeconds())
                        queue = root.find('machine_queue')
                        for element in self.__taskDic:
```

```
                                        temp_element = queue.find(element)
                                        temp_Status = temp_element.find('Status')
                                        temp_Status.text = str(self.__taskDic[element]._Status)
                                        if self.__taskDic[element]._Status == 'Running':
                                                self.__taskDic[element]._ProcessingRmainingTime = self.__taskDic[element]._EndTime - self.getCurrentTimeInSeconds()
                                                #print "remaining processing time: ",self.__taskDic[element]._ProcessingRmainingTime

                                        temp_EndTime = temp_element.find('EndTime')
                                        temp_EndTime.text = str(self.__taskDic[element]._EndTime)
                                        temp_Priority = temp_element.find('Priority')
                                        temp_Priority.text = str(self.__taskDic[element]._Priority)
                                        temp_ProcessingTime = temp_element.find('ProcessingTime')
                                        temp_ProcessingTime.text = str(self.__taskDic[element]._ProcessingTime)
                                        temp_ProcessingRmainingTime = temp_element.find('ProcessingRmainingTime')
                                        temp_ProcessingRmainingTime.text = str(self.__taskDic[element]._ProcessingRmainingTime)

                                        #print("%s End time %d:%d saved ...."%(element,self.__taskDic[element]._EndTime/3600,((self.__taskDic[element]._EndTime%3600)/60)))

                                self.__backup_file.write('backup_machine4.xml')
                                self.__backup_file_lock.release()
                                time.sleep(30)


        #show the shuttles in the machine queue
        def print_elements_queue(self):
                print "|\tTask name\t|\tStart Time\t|\tFinish Time\t|\tProcessing Time\t|\tDeadline\t|\tStatus"
                print "|\t--------\t|\t----------\t|\t-----------\t|\t---------\t|\t------\t\t|\t-------"
                for key,value in self.__taskDic.iteritems():

        print("|\t%s\t|\t%d:%d\t\t|\t%d:%d\t\t|\t%d:%d\t\t|\t%d:%d\t\t|\t%s\t\t"%(key,value._StartTime/3600,(value._StartTime%3600)/60,value._WorstCaseFinishingTime/3600,(value._WorstCaseFinishingTime%3600)/60,value._ProcessingRmainingTime/3600,(value._ProcessingRmainingTime%3600)/60,value._EndTime/3600,(value._EndTime%3600)/60,value._Status))



#####################################################
        #Remove specific shuttle from machine queue
        def __removeTask(self,msg):
                if msg['sendername'] in self.__taskDic:
                        print "Removing..... ",self.__taskDic[msg['sendername']]._Name

                        #acquire lock before using xml file
                        self.__backup_file_lock.acquire()
                        # remove task from the Backup file
                        self.__backup_file = ET.parse('backup_machine4.xml')
                        root = self.__backup_file.getroot()
                        queue = root.find('machine_queue')
                        temp_task = queue.find(msg['sendername'])
                        queue.remove(temp_task)
                        self.__backup_file.write('backup_machine4.xml')
                        self.__backup_file_lock.release()
                        # remove task from tasks dictionary
                        del self.__taskDic[msg['sendername']]
                        print "Task removed ....."
                else:
                        print ("%s: you are trying to remove a not existing task"% self.__removeTask.__name__)


#############################################################
#handler to cancel task scheduling and remove it from the task queue of the machine
```

```
        def cancelRequest(self,msg):
                print "I got cancel request from: ",msg['sendername']
                self.__removeTask(msg)


#####################################################
        def __converttoseconds(self,strtime):

                temp =timeTest.getendtimestr(strtime)
                temp = myTime(temp)
                seconds = (temp.hours * 60 + temp.minutes) * 60
                return seconds

#####################################################
        def getCurrentTimeInSeconds(self):
                currenttime = datetime.datetime.now()
                currenttimeinseconds = ( currenttime.hour * 60 + currenttime.minute ) * 60
                return currenttimeinseconds

    # Determine the delay until this method should be called again
    if self.state == STATE_PAUSED:
       wait_seconds = None
       self._logger.debug('Scheduler is paused; waiting until resume() is called')
    elif next_wakeup_time is None:
       wait_seconds = None
       self._logger.debug('No jobs; waiting until a job is added')
    else:
       wait_seconds = min(max(timedelta_seconds(next_wakeup_time - now), 0), TIMEOUT_MAX)
       self._logger.debug('Next wakeup is due at %s (in %f seconds)', next_wakeup_time,
                wait_seconds)

          #handler for arriving tasks
          def taskArrived(self,message): # cannot make it private because it is called outside the class in addhandler
                print message
                tmpmsg = message['data']
                tempint = list(tmpmsg)
                taskNum = int(tempint[0])
                print "task number: ", taskNum
                priority = int(tempint[1])
                processingTime =int(tempint[2]+tempint[3]+tempint[4]) * 60 # converted to seconds
                endTime = tempint[5]+tempint[6]+tempint[7]+tempint[8]
                minStartTime = tempint[9]+tempint[10]+tempint[11]
                print "minStartTime: " , minStartTime
                minStartTime = int(minStartTime) * 60 + self.getCurrentTimeInSeconds()
                if(self.__converttoseconds(endTime) < self.getCurrentTimeInSeconds()): # should be <
tasksDic[name]._MinStartTime
                        print "end time less than current time ....."
                        return False

                self.__newTask[message['sendername']]=
{'priority':priority,'processingTime':processingTime,'endTime':endTime,'ContractNumber':taskNum,'minStartTime':m
inStartTime}

        self.__newTask[message['sendername']]['endTime']=self.__converttoseconds(self.__newTask[message['sen
dername']]['endTime'])
                self.__addNewTask() # create new task object and add it to the the task queue of the machine
                self.__scheduleSuccess = self.__scheduleTasks(message['sendername'])
                print "self.__scheduleSuccess: ", self.__scheduleSuccess
                if (self.__scheduleSuccess):
                        #print("schedule for %s done,start time: %f, End Time:
%f"%(self.__taskDic[message['sendername']]._Name,self.__taskDic[message['sendername']]._StartTime/3600.0,self.
__taskDic[message['sendername']]._WorstCaseFinishingTime/3600.0))
                        response = str(self.__taskDic[message['sendername']]._StartTime) +' '+
str(self.__taskDic[message['sendername']]._WorstCaseFinishingTime)
                        print "ContractNumber: ",(self.__taskDic[message['sendername']]._ContractNumber)
                        # add new entry to xml backup file
```

```
                        self.__backup_file = ET.parse('backup_machine4.xml')
                        root = self.__backup_file.getroot()
                        queue = root.find('machine_queue')
                        task_element = ET.SubElement(queue,message['sendername'])
                        ET.SubElement(task_element,'EndTime')
                        ET.SubElement(task_element,'Priority')
                        ET.SubElement(task_element,'ProcessingTime')
                        ET.SubElement(task_element,'ProcessingRmainingTime')
                        ET.SubElement(task_element,'Status')
                        self.__backup_file.write('backup_machine4.xml')

                        # set this flag to true to create new row in the gui
                        self.__create_row_gui = True
                        self.__new_task_name = message['sendername']
                        #print response
                        self.sendMessageFunc('TCP', message['sendername'],", 'SCHEDULEDM4', response)

                if(not self.__scheduleSuccess):
                        del self.__taskDic[message['sendername']]
                        response = '00'
                        self.sendMessageFunc('TCP', message['sendername'],", 'SCHEDULEFAILM4',
response)
                self.__scheduleFail = False
                print "End of task arrived ....."
                self.print_elements_queue()
                print "\n"
                self.printSlots()
                return True


        # create new task object and add it to the the task queue of the machine
        def __addNewTask(self):
                tempdic = {}
                key = self.__newTask.keys()
                self.__lastArrivingTask = key[0]
                print key[0]
                tempdic[key[0]] =  self.__newTask[key[0]]
                self.__taskDic[key[0]]                                                                  =
queueElement(tempdic[key[0]]['priority'],tempdic[key[0]]['endTime'],tempdic[key[0]]['processingTime'],key[0],temp
dic[key[0]]['ContractNumber'],tempdic[key[0]]['minStartTime'])
                print "task added to self.__dic :",self.__taskDic
                self.__newTask.clear()
                tempdic.clear()
                return True

        # Apply the scheduling algorithm
        def __scheduleTasks(self,name):
                        tasksEndTime ={}
                        currenttimeinseconds = self.getCurrentTimeInSeconds()
                        for key,value in self.__taskDic.iteritems():
                                tasksEndTime[key] =value._EndTime
                        del tasksEndTime[name]  # delete the new task from the list
                        # sort the tasks according to their endtime (earliest deadline first)
                        sortdtasks = sorted(tasksEndTime.items(),key=operator.itemgetter(1))

                        if not sortdtasks: # list is empty ,schedule the first element(task_dic contain only one
element)
                                self.__taskDic[name]._WorstCaseFinishingTime                            =
self.__taskDic[name]._EndTime
                                self.__taskDic[name]._StartTime    =    self.__taskDic[name]._EndTime    -
self.__taskDic[name]._ProcessingTime
                                self.__taskDic[name]._DeadlineForRelatedTasks                           =
self.__taskDic[name]._EndTime - self.__taskDic[name]._ProcessingTime - self.__transportationTime
                                self.__taskDic[name]._Status ="Scheduled"
                                print("First task satrt time: %d:%d ,endTime: %d:%d ,related tasks deadline:
%d:%d"%(self.__taskDic[name]._StartTime/3600.0,(self.__taskDic[name]._StartTime%3600)/60,self.__taskDic[na
```

```
me]._EndTime/3600.0,(self.__taskDic[name]._EndTime%3600)/60,self.__taskDic[name]._DeadlineForRelatedTasks
/3600.0,(self.__taskDic[name]._DeadlineForRelatedTasks%3600)/60))
                                        return True

                                print "sorted tasks according to Ealiest Deadline First " , sortdtasks

                                if sortdtasks:  # if the list is not empty
                                        print "the largest deadline is ..",sortdtasks[-1] #

                                        if(self.__taskDic[name]._EndTime > self.__taskDic[sortdtasks[-
1][0]]._EndTime): #compare the new task deadline with the biggest one in the list

                                                print "yes greater than the largest deadline ....."

                                                if ((self.__taskDic[sortdtasks[-1][0]]._EndTime +
self.__taskDic[name]._ProcessingTime + self.__transportationTime) < self.__taskDic[name]._EndTime): # (case 1)
                                                        print "yes it's possible to assign a direct time slot after the
last task  "
                                                        self.__taskDic[name]._WorstCaseFinishingTime =
self.__taskDic[name]._EndTime
                                                        self.__taskDic[name]._StartTime =
self.__taskDic[name]._EndTime - self.__taskDic[name]._ProcessingTime
                                                        self.__taskDic[name]._DeadlineForRelatedTasks =
self.__taskDic[name]._EndTime - self.__taskDic[name]._ProcessingTime - self.__transportationTime
                                                        self.__taskDic[name]._Status ="Scheduled"
                                                        print("task satrt time: %d:%d ,endTime: %d:%d ,related
tasks deadline:
%d:%d"%(self.__taskDic[name]._StartTime/3600.0,(self.__taskDic[name]._StartTime%3600)/60,self.__taskDic[na
me]._EndTime/3600.0,(self.__taskDic[name]._EndTime%3600)/60,self.__taskDic[name]._DeadlineForRelatedTasks
/3600.0,(self.__taskDic[name]._DeadlineForRelatedTasks%3600)/60))
                                                        return True

                                                else: # tasks overlap (the task has a greater dead line but overlaps with
the last task )(case 4)
                                                        print "deadline greater but it's not possible to assign a direct
time slot after the last task  \n"
                                                        self.__freeSlots = self.getFreeTimeSlots()
                                                        for i in range(len(self.__FreeSlots)):
                                                                if(self.__FreeSlots[i].getDuration() >=
(self.__taskDic[name]._ProcessingTime+self.__transportationTime)):
                                                                        print "the time slot is suitable for the
task \n"
                                                                        self.__taskDic[name]._Status
="Scheduled"


                                                                self.__taskDic[name]._WorstCaseFinishingTime =
self.__taskDic[self.__FreeSlots[i].getNextTask()]._StartTime - self.__transportationTime
                                                                self.__taskDic[name]._StartTime =
self.__taskDic[name]._WorstCaseFinishingTime - self.__taskDic[name]._ProcessingTime

                                                                self.__taskDic[name]._DeadlineForRelatedTasks =
self.__taskDic[name]._StartTime - self.__transportationTime
                                                                print("time slot num %d is suitalbe for
the task "%(i))
                                                                return True
                                                        else:
                                                                print("time slot num %d is not suitalbe
for the task "%(i))

                                        else: #  check for free time slots and see if it is possible to assign a free slot to
the task
                                                print "the deadline is not the largest....."
                                                self.__freeSlots = self.getFreeTimeSlots()
                                                print "come back from getFreeTimeSlots\n"
                                                for i in range(len(self.__FreeSlots)):
```

```
                                                print "entered for loop"
                                                print "self.__taskDic[name]._MinStartTime: ",self.__taskDic[name]._MinStartTime
                                                condition_1 = self.__FreeSlots[i].getDuration() >= (self.__taskDic[name]._ProcessingTime+self.__transportationTime)
                                                condition_2 = self.__FreeSlots[i].getEndTime() - (self.__taskDic[name]._ProcessingTime+self.__transportationTime) > self.__taskDic[name]._MinStartTime
                                                if(condition_1 and condition_2):
                                                        print "the time slot is suitable for the task "
                                                        tempFinishTime = self.__taskDic[self.__FreeSlots[i].getNextTask()]._StartTime - self.__transportationTime
                                                        if (tempFinishTime <self.__taskDic[name]._EndTime):

        self.__taskDic[name]._WorstCaseFinishingTime = tempFinishTime
                                                                self.__taskDic[name]._StartTime = self.__taskDic[name]._WorstCaseFinishingTime - self.__taskDic[name]._ProcessingTime

        self.__taskDic[name]._DeadlineForRelatedTasks = self.__taskDic[name]._StartTime - self.__transportationTime
                                                                self.__taskDic[name]._Status ="Scheduled"
                                                                print("time slot num %d is suitalbe for the task "%(i))
                                                                return True
                                                        else :

        self.__taskDic[name]._WorstCaseFinishingTime = self.__taskDic[name]._EndTime

        if((self.__FreeSlots[i].getPreviousTask() == '') and (self.__taskDic[name]._WorstCaseFinishingTime - self.__taskDic[name]._MinStartTime) >= (self.__taskDic[name]._ProcessingTime+self.__transportationTime)): # no previous tasks

        self.__taskDic[name]._StartTime = self.__taskDic[name]._WorstCaseFinishingTime - self.__taskDic[name]._ProcessingTime

        self.__taskDic[name]._DeadlineForRelatedTasks = self.__taskDic[name]._StartTime - self.__transportationTime
                                                                        self.__taskDic[name]._Status ="Scheduled"
                                                                        print("time slot num %d is suitalbe for the task "%(i))
                                                                        return True


        elif((self.__taskDic[name]._WorstCaseFinishingTime - self.__taskDic[self.__FreeSlots[i].getPreviousTask()]._EndTime) >= (self.__taskDic[name]._ProcessingTime+self.__transportationTime)):

        self.__taskDic[name]._StartTime = self.__taskDic[name]._WorstCaseFinishingTime - self.__taskDic[name]._ProcessingTime

        self.__taskDic[name]._DeadlineForRelatedTasks = self.__taskDic[name]._StartTime - self.__transportationTime
                                                                        self.__taskDic[name]._Status ="Scheduled"
                                                                        print("time slot num %d is suitalbe for the task "%(i))
                                                                        return True
                                                                else:
                                                                        print("time slot num %d is not suitalbe for the task form inner if condition  "%(i))
                                                                        break


                                                else:
```

```
                                                print("time slot num %d is not suitalbe for the
task "%(i))

################################################

        def getFreeTimeSlots(self):
                timeSlotsList =[]
                tasksEndTime ={}
                sortedtasks =[]


                for key,value in self.__taskDic.iteritems():
                        tasksEndTime[key] =value._WorstCaseFinishingTime

                sortedtasks = sorted(tasksEndTime.items(),key=operator.itemgetter(1))
                if self.__lastArrivingTask and not (max(tasksEndTime.iteritems(),key=operator.itemgetter(1))[0]
== self.__taskDic[self.__lastArrivingTask]._EndTime):
                                del tasksEndTime[self.__lastArrivingTask]  # delete the new task from the list
                                print "the new task is not the biggest one , it will be deleted form sortedtasks"
                                self.__lastArrivingTask = ''
                                sortedtasks = sorted(tasksEndTime.items(),key=operator.itemgetter(1))
                                print "sortedtasks: ",sortedtasks

                for i in range(len(sortedtasks)):
                        tempEnd = self.__taskDic[sortedtasks[-i-1][0]]._StartTime
                        timeSlotsList.append(FreeTimeSlot())
                        timeSlotsList[i].setEndTime(tempEnd)
                        timeSlotsList[i].setNextTask(self.__taskDic[sortedtasks[-i-1][0]]._Name)

                for i in range(0,len(sortedtasks)-1):
                        tempStart = self.__taskDic[sortedtasks[-i-2][0]]._WorstCaseFinishingTime
                        timeSlotsList[i].setStartTime(tempStart)
                        timeSlotsList[i].setPreviousTask( self.__taskDic[sortedtasks[-i-2][0]]._Name)
                if sortedtasks:
                        timeSlotsList[len(sortedtasks)-1].setStartTime(self.getCurrentTimeInSeconds())    #start
time for the last time slot = currenttime

                for i in range(len(sortedtasks)):
                        tempDuration = timeSlotsList[i].getEndTime() - timeSlotsList[i].getStartTime()
                        timeSlotsList[i].setDuration(tempDuration)

                for i in range(len(sortedtasks)):
                        print("previous task: %s , slot number: %d , next task:
%s"%(timeSlotsList[i].getPreviousTask(),i,timeSlotsList[i].getNextTask()))


                self.__FreeSlots = timeSlotsList
                return timeSlotsList

############################################
        def __updateFreeSlots(self):
                timeSlotsList =[]
                tasksEndTime ={}
                orderedtasks =[]

                for key,value in self.__taskDic.iteritems():
                        tasksEndTime[key] =value._WorstCaseFinishingTime

                orderedtasks = sorted(tasksEndTime.items(),key=operator.itemgetter(1))
                for i in range(len(orderedtasks)):
                        tempEnd = self.__taskDic[orderedtasks[-i-1][0]]._StartTime
                        timeSlotsList.append(FreeTimeSlot())
                        timeSlotsList[i].setEndTime(tempEnd)
                        timeSlotsList[i].setNextTask(self.__taskDic[orderedtasks[-i-1][0]]._Name)
```

```
                for i in range(0,len(orderedtasks)-1):
                        tempStart = self.__taskDic[orderedtasks[-i-2][0]]._WorstCaseFinishingTime
                        timeSlotsList[i].setStartTime(tempStart)
                        timeSlotsList[i].setPreviousTask( self.__taskDic[orderedtasks[-i-2][0]]._Name)
                if orderedtasks:
                        timeSlotsList[len(orderedtasks)-1].setStartTime(self.getCurrentTimeInSeconds())  #start
time for the last time slot = currenttime

                for i in range(len(orderedtasks)):
                        tempDuration = timeSlotsList[i].getEndTime() - timeSlotsList[i].getStartTime()
                        timeSlotsList[i].setDuration(tempDuration)

                self.__FreeSlots = timeSlotsList
                return timeSlotsList

##############################################

        def printSlots(self):
                print "|\tSlot num.\t|\tstart Time\t|\tDuration\t|\tEnd Time"
                print "|\t---------\t|\t----------\t|\t---------\t|\t---------"
                self.__FreeSlots = self.__updateFreeSlots()
                for i in range(len(self.__FreeSlots)):
                        tempStart = self.__FreeSlots[i].getStartTime()
                        tempDuration = self.__FreeSlots[i].getDuration()
                        tempEnd  = self.__FreeSlots[i].getEndTime()

        print("|\t%d\t\t|\t%d:%d\t\t|\t%d:%d\t\t|\t%d:%d\t\t|"%(i,tempStart/3600,(tempStart%3600)/60,tempDuration
/3600,(tempDuration%3600)/60,tempEnd/3600,(tempEnd%3600)/60))

##############################################
        def set_create_row_gui_flag(self):
                count = 0
                while True:
                        element = queueElement(2,3456,34,'shuttle_'+str(count),3,2)
                        self.__taskDic[element._Name]= element
                        self.__new_task_name = element._Name
                        self.__create_row_gui = True
                        print "flag set right ......."
                        print "task dict: ",self.__taskDic
                        count = count + 1
                        time.sleep(10)

        def gui_funtion(self):
                root = Tk()

                machine_name_frame = Frame(root)
                machine_name_frame.pack(side=TOP, fill=X, padx=1, pady=1)
                machine_name= StringVar()
                machine_name_label = Label( machine_name_frame, textvariable=machine_name,font = 80,
relief=RAISED,height = 5 ,width = 20 )
                machine_name.set("FREASEMACHINE")
                machine_name_label.pack(side=LEFT)

                Aktuelle_Zeit = StringVar()

                Aktuelle_Zeit_label = Label( machine_name_frame, textvariable=Aktuelle_Zeit,
relief=RAISED,height = 5 ,width = 40 )
                Aktuelle_Zeit.set("Zeit")
                Aktuelle_Zeit_label.pack(side=LEFT)

                rows_list = []
                status_dic ={}
                # function responsible for updating the GUI after (1000 msec)
                def outer_update(root,task_dic):
                        def update():
```

```
                                    if (self.__create_row_gui): # need to check for the count of rows
                                            print "creating new row for: ",self.__taskDic[self.__new_task_name]

                                            self.__gui_rows_counter = self.__gui_rows_counter + 1
                                            print "gui rows counter: " ,self.__gui_rows_counter
                                            self.__create_row_gui = False
                                            row = Frame(root)
                                            row.pack(side=TOP, fill=X, padx=1, pady=1)

                                            number = StringVar()
                                            label1 = Label( row, textvariable=number, relief=RAISED,height =
5 ,width = 20 )
                                            number.set(self.__gui_rows_counter)
                                            label1.pack(side=LEFT)

                                            Auftrag = StringVar()
                                            label2 = Label( row, textvariable=Auftrag, relief=RAISED,height =
5 ,width = 40 )
                                            Auftrag.set(task_dic[self.__new_task_name]._ContractNumber)
                                            label2.pack(side=LEFT)

                                            Zeit = StringVar()
                                            label2 = Label( row, textvariable=Zeit, relief=RAISED,height = 5
,width = 40 )
                                            Zeit.set(str(task_dic[self.__new_task_name]._StartTime) + ' - ' +
str(task_dic[self.__new_task_name]._EndTime))
                                            label2.pack(side=LEFT)

                                            Status = StringVar()
                                            label2 = Label( row, textvariable=Status, relief=RAISED,height = 5
,width = 40 )
                                            Status.set(task_dic[self.__new_task_name]._Status)
                                            label2.pack(side=LEFT)

                                            status_dic[self.__new_task_name]= Status
                                            rows_list.append(row)
                                            print rows_list
                                    for st in status_dic:
                                            print " staus name: ",st
                                            print status_dic
                                            status_dic[st].set(self.__taskDic[st]._Status)
                                    Aktuelle_Zeit.set( datetime.datetime.now())
                                    root.update_idletasks()
                                    print task_dic
                                    root.after(1000,update)
                            update()
```

A2.5 Digitale Repräsentation

Programmcode

```
#-*-coding:utf-8-*-
from __future__ import print_function
import json

import ipaddress
import time
from functools import partial

#kivy libraries
from kivy.app import App
from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.boxlayout import BoxLayout
from kivy.utils import platform
from kivy.config import Config, ConfigParser
from kivy.properties import ObjectProperty, NumericProperty, StringProperty
from kivy.storage.jsonstore import JsonStore
from kivy.uix.popup import Popup
from kivy.uix.label import Label
#from kivy.uix.slider import Slider
#from kivy.logger import Logger

#user libraries
from libraries.p2p_framework import P2P_Interface
if platform == "android":
    from jnius import autoclass
    from libraries.alertdialog import alertDialog
    from android.config import JAVA_NAMESPACE
    NativeInvocationHandler = autoclass("org.jnius.NativeInvocationHandler")
    drawable = autoclass("org.test.pelabor.R$drawable")
from config.jsonsettings import settings_json

from __future__ import print_function
from __future__ import absolute_import
import os
import json
from functools import partial

from libraries.p2p_framework import P2P_Interface

from kivy.logger import Logger
#from plyer import notification
from kivy.utils import platform
from jnius import autoclass, PythonJavaClass, java_method, cast

if platform == "android":
    from android.config import JAVA_NAMESPACE
    from android.broadcast import BroadcastReceiver
    from libraries.androidNotify import notification
    NativeInvocationHandler = autoclass("org.jnius.NativeInvocationHandler")
    PythonActivity = autoclass("{}.PythonActivity".format(JAVA_NAMESPACE))
    PythonService = autoclass("{}.PythonService".format(JAVA_NAMESPACE))
    current_activity = cast('android.app.Activity', PythonService.mService)
    drawable = autoclass("org.test.pelabor.R$drawable")
    String = autoclass('java.lang.String')
    AndroidInt = autoclass('java.lang.Integer')
    Intent = autoclass('android.content.Intent')
    Context = autoclass('android.content.Context')

shutdown = [False]
notify_answ = "{}.NOTIFY_ANSW".format(PythonService.mService.getPackageName())
```

```
class P2P_Relay(P2P_Interface):
    status = True

    def __init__(self, shutdown, name, tcp_ports=54545, udp_ports=56565,
                 broadcast=True, relay=False, target="localhost"):
        self.relay = relay #other P2P_Relay Instance
        self.target = target

        super(P2P_Relay, self).__init__(shutdown, name, logger=None,
                                        tcp_ports=tcp_ports, udp_ports=udp_ports,
                                        broadcast=True)

        self.add_handler("targetIP",self.set_target_ip)
        self.add_handler("setStatus", self.set_status)

    def handlemessage(self, msg):
        if msg['command'] == 'introduction':
            pass

        elif msg['command'] in self.handlers:
            # execute the handler and provide the message as a parameter
            self.handlers[msg['command']](msg)

        else:
            self.relay.sendRelay(msg["command"], msg["data"], msg["priority"])

    def sendRelay(self, command ,data, prio=1):
        if self.status:
            self.sendByIP(self.target, command, data, prio)

    def set_target_ip(self, msg):
        print("New target:",msg["data"])
        self.target = msg["data"]

    def set_status(self, msg):
        if msg["data"] == True:
            self.status = True
        elif msg["data"] == False:
            self.status = False


def set_name(msg):
    extern.activeacc[0] = msg["data"]

def notifyUser(msg):
    options = {"title":"Test Title","message":"Test Message"}
    options.update(msg["data"])

    if intern.status:
        intern.sendRelay("NOTIFY",options)
    else:
        notification(**options)

def on_broadcast(context, intent):
    try:
        notificationManager = PythonService.mService.getSystemService(Context.NOTIFICATION_SERVICE)
        #notificationManager.cancel(id)
        notificationManager.cancelAll()
        extras = intent.getExtras()
        data = extras.getString("data")
        id = extras.getString("strId")
        #print("_>"*50,id,"----",data,"<_"*50)
        extern.sendRelay("ANSWER", [id,data])
    except:
        print("_-"*50,"broadcast","_-"*50)
```

```
def broadcast_me(msg):
    extra = msg.get("data","empty")
    i = Intent()
    i.setAction(notify_answ)
    i.putExtra(String("data"),String(extra))
    current_activity.sendBroadcast(i)

def exitme(msg):
    global shutdown
    shutdown[0] = True
    exit()

if __name__ == "__main__":
    options = {"name":"Android", "targetIP":"127.0.0.1"}
    print("--"*50,"SERVICE_RUNNING","--"*50)
    try:
        args = json.loads(os.getenv('PYTHON_SERVICE_ARGUMENT'))

        if type(args) == dict:
            options.update(args)
    except:
        pass

    intern = P2P_Relay(shutdown, "Relay",
                       tcp_ports=(30003,30002),
                       udp_ports=(30005,30004),
                       broadcast=False)

    extern = P2P_Relay(shutdown, options["name"],relay=intern, target=options["targetIP"])
    intern.relay = extern

    intern.add_handler("EXIT", exitme)
    intern.add_handler("setNAME", set_name)

    extern.add_handler("EXIT", exitme)
    extern.add_handler("NOTIFY", notifyUser)
    extern.add_handler("BCAST", broadcast_me)

    BRec = BroadcastReceiver(on_broadcast, actions=[notify_answ])
    BRec.start()


P2P = True
DEBUG = True
def dprint(*args):
    if DEBUG:
        print(*args)

class SkillSlider(BoxLayout):
    value = NumericProperty(None)
    desc = StringProperty(None)


class TaskEntry(BoxLayout):
    time = StringProperty(None)
    desc = StringProperty(None)


class Testme(BoxLayout):
    screen_manager_1 = ObjectProperty(None)
    app = ObjectProperty(None)
    screen_history = ["Skills","Skills"]
    store = JsonStore("config/store.json")
```

```
    def __init__(self, config):
        self.config = config
        self.shutdown = [False]

        name = self.config.get("PE Labor", "name")
        ip = self.config.get("PE Labor", "ip")

        if P2P:
            self.p2p = P2P_Interface(self.shutdown, "intern", tcp_ports=(30002,30003), udp_ports=(30004,30005),
broadcast=False)
            self.startService(servicearg={"name":name, "targetIP":ip})

            self.p2p.add_handler("SKILLS",self.update_skills)
            self.p2p.add_handler("NOTIFY",self.notifyUser)

        super(Testme, self).__init__()

        if ip in ["127.0.0.1", "localhost"]:
            self.screen_manager_1.current = "Options"

        self.load_skills()
        self.load_tasks()

    def scrollme(self):
        now = time.strftime("%H:%M")
        tasks list = [tasks for tasks in self.ids if tasks.startswith("task_")]
        task = [tasks for tasks in tasks list if self.ids[tasks].desc.startswith(now[:2])]
        self.ids.scrollme.scroll_to(self.ids[task[0]])

    def alertUser(self, callback=None, title="Test", text="Displayed?", button=None, strId="None"):
        callback = callback if callback is not None else self.answerUser
        dialog = alertDialog(callback=callback, title=title, text=text, button=button, strId=strId)
        dialog.open()

    def answerUser(self, answer=False, id="None"):
        #print("+"*20,"User answer is:", answer, "+"*20)
        self.send("ANSWER", [id,answer])

    def notifyUser(self, msg):
        self.alertUser(callback=self.answerUser, title=msg["data"]["title"],
                    text=msg["data"]["message"], button=msg["data"].get("actions",None),
                    strId=msg["data"].get("strId","None"))
        #self.alertUser()
        #print("--"*50,"GET NOTIFIED","--"*50)

    def load_tasks(self):
        for x in range(0,24):
            if x < 10:
                x = "0{}".format(x)
            task = TaskEntry(time="{}:00".format(x), desc="{} Just saying".format(x))
            self.ids[("task_{}").format(x)] = task
            self.ids.TaskBox.add_widget(task)

    def send(self, cmd, data=""):
        self.p2p.sendByIP("localhost", cmd, data)

    def submit_skills(self):
        skilldict = self.save_skills()
        if P2P:
            dprint("SUBMIT")
            self.send("SKILLS", skilldict)

    def load_skills(self):
        for skill, value in self.store.get("skill").items():
            newskill = SkillSlider(desc=skill, value=int(value))
            self.ids.skillBox.add_widget(newskill)
```

```
            self.ids[("skill_{}").format(skill)] = newskill

    def save_skills(self):
        self.store.delete("skill")
        skilldict = dict([self.ids[x].stats for x in self.ids if x.startswith("skill_")])
        self.store.put("skill",**skilldict)
        return skilldict

    def update_skills(self, msg):
        skills = [self.ids[x].desc for x in self.ids if x.startswith("skill_")]
        for skill, value in msg["data"].items():
            if skill in skills:
                if str(value) == "remove":
                    try:
                        self.ids["skillBox"].remove_widget(self.ids["skill_{}".format(skill)])
                        self.ids.pop("skill_{}".format(skill), None)
                    except:
                        pass
                else:
                    self.ids["skill_{}".format(skill)].value = int(value)
            else:
                newskill = SkillSlider(desc=skill, value=int(value))
                self.ids.skillBox.add_widget(newskill)
                self.ids[("skill_{}").format(skill)] = newskill
        self.save_skills()

    def changeRelay(self, ip):
        self.send("targetIP", ip)

    def changeName(self, name):
        self.send("setNAME", name)

    def change_screen(self, instance, text):
        if text == "Options":
            self.app.open_settings()
            instance.text = self.screen_history[1]
        else:
            self.screen_history = [self.screen_history[1],text]
            self.screen_manager_1.current = text

    def startService(self, packagename="org.test.pelabor", servicename="Apprelay", servicearg=None):
        if platform == 'android':
            #dprint("--"*60,"SERVICE","--"*60)
            #serviceclassname = "".join([packagename,".","Service",servicename])
            self.service = autoclass('org.test.pelabor.ServiceApprelay')
            #self.service = autoclass(serviceclassname)
            mActivity = autoclass('{}.PythonActivity'.format(JAVA_NAMESPACE)).mActivity
            if servicearg is not None:
                argument = json.dumps(servicearg)
            else:
                argument = ""

            self.service.start(mActivity, argument)
            #dprint("--"*60,"SERVICE__2","--"*60)

        elif platform == "win":
            #import subprocess
            import os
            os.startfile(os.path.abspath("service\\Apprelay.py"))
            time.sleep(2)
            self.changeRelay(servicearg["targetIP"])
        else:
            dprint("startService only implemented for Android yet")

        # Determine the delay until this method should be called again
        if self.state == STATE_PAUSED:
```

```
            wait_seconds = None
            self._logger.debug('Scheduler is paused; waiting until resume() is called')
        elif next_wakeup_time is None:
            wait_seconds = None
            self._logger.debug('No jobs; waiting until a job is added')
        else:
            wait_seconds = min(max(timedelta_seconds(next_wakeup_time - now), 0), TIMEOUT_MAX)
            self._logger.debug('Next wakeup is due at %s (in %f seconds)', next_wakeup_time,
                               wait_seconds)

                #handler for arriving tasks
                def taskArrived(self,message): # cannot make it private because it is called outside the class in addhandler
                        print message
                        tmpmsg = message['data']
                        tempint = list(tmpmsg)
                        taskNum = int(tempint[0])
                        print "task number: ", taskNum
                        priority = int(tempint[1])
                        processingTime =int(tempint[2]+tempint[3]+tempint[4]) * 60 # converted to seconds
                        endTime = tempint[5]+tempint[6]+tempint[7]+tempint[8]
                        minStartTime = tempint[9]+tempint[10]+tempint[11]
                        print "minStartTime: " , minStartTime
                        minStartTime = int(minStartTime) * 60 + self.getCurrentTimeInSeconds()
                        if(self.__converttoseconds(endTime) < self.getCurrentTimeInSeconds()): # should be < tasksDic[name]._MinStartTime
                                return False

                        self.__newTask[message['sendername']]= {'priority':priority,'processingTime':processingTime,'endTime':endTime,'ContractNumber':taskNum,'minStartTime':minStartTime}

                self.__newTask[message['sendername']]['endTime']=self.__converttoseconds(self.__newTask[message['sendername']]['endTime'])
                        self.__addNewTask() # create new task object and add it to the the task queue of the machine
                        self.__scheduleSuccess = self.__scheduleTasks(message['sendername'])
                        print "self.__scheduleSuccess: ", self.__scheduleSuccess
                        if (self.__scheduleSuccess):
                                #print("schedule for %s done,start time: %f, End Time: %f"%(self.__taskDic[message['sendername']]._Name,self.__taskDic[message['sendername']]._StartTime/3600.0,self.__taskDic[message['sendername']]._WorstCaseFinishingTime/3600.0))
                                response = str(self.__taskDic[message['sendername']]._StartTime) +' '+ str(self.__taskDic[message['sendername']]._WorstCaseFinishingTime)
                                print "ContractNumber: ",(self.__taskDic[message['sendername']]._ContractNumber)
                                # add new entry to xml backup file
                                self.__backup_file = ET.parse('backup_machine4.xml')
                                root = self.__backup_file.getroot()
                                queue = root.find('machine_queue')
                                task_element = ET.SubElement(queue,message['sendername'])
                                ET.SubElement(task_element,'EndTime')
                                ET.SubElement(task_element,'Priority')
                                ET.SubElement(task_element,'ProcessingTime')
                                ET.SubElement(task_element,'ProcessingRmainingTime')
                                ET.SubElement(task_element,'Status')
                                self.__backup_file.write('backup_machine4.xml')

                                # set this flag to true to create new row in the gui
                                self.__create_row_gui = True
                                self.__new_task_name = message['sendername']
                                #print response
                                self.sendMessageFunc('TCP', message['sendername'],", 'SCHEDULEDM4', response)

                        if(not self.__scheduleSuccess):
                                del self.__taskDic[message['sendername']]
                                response = '00'
                                self.sendMessageFunc('TCP', message['sendername'],", 'SCHEDULEFAILM4', response)
```

```
                    self.__scheduleFail = False
                    print "End of task arrived ....."
                    self.print_elements_queue()
                    print "\n"
                    self.printSlots()
                    return True


        # create new task object and add it to the the task queue of the machine
        def __addNewTask(self):
                    tempdic = {}
                    key = self.__newTask.keys()
                    self.__lastArrivingTask = key[0]
                    print key[0]
                    tempdic[key[0]] =  self.__newTask[key[0]]
                    self.__taskDic[key[0]] =
queueElement(tempdic[key[0]]['priority'],tempdic[key[0]]['endTime'],tempdic[key[0]]['processingTime'],key[0],temp
dic[key[0]]['ContractNumber'],tempdic[key[0]]['minStartTime'])
                    print "task added to self.__dic :",self.__taskDic
                    self.__newTask.clear()
                    tempdic.clear()
                    return True

        # Apply the scheduling algorithm
        def __scheduleTasks(self,name):
                            tasksEndTime ={}
                            currenttimeinseconds = self.getCurrentTimeInSeconds()
                            for key,value in self.__taskDic.iteritems():
                                    tasksEndTime[key] =value._EndTime
                            del tasksEndTime[name]  # delete the new task from the list
                            # sort the tasks according to their endtime (earliest deadline first)
                            sortdtasks = sorted(tasksEndTime.items(),key=operator.itemgetter(1))


class TestApp(App):
    def build(self):
        if platform == "android":
            activity = autoclass('org.kivy.android.PythonActivity').mActivity
            activity.removeLoadingScreen()
        Config.set('kivy', 'keyboard_mode', 'system')
        self.test = Testme(self.config)
        return self.test

    def on_pause(self):
        print("+"*100)
        self.test.send("setStatus", False)
        return True

    def on_resume(self):
        self.test.send("setStatus", True)

    def on_stop(self):
        self.test.send("setStatus", False)

    def build_config(self, config):
        config.setdefaults("PE Labor",{"name":"Android","ip":"127.0.0.1"})

    def build_settings(self, settings):
        settings.add_json_panel("Options", self.config, data=settings_json)

    def on_config_change(self, config, section, key, value):
        dprint("{}|{}|{}".format(section,key,value))
        if section == "PE Labor":
            if key == "ip":
                try:
                    iptest = ipaddress.ip_address(value)
```

```
                self.test.changeRelay(value)
            except ValueError:
                config.set(section, key, "127.0.0.1")
                config.write()

                popup = Popup(title='Error',
                        content=Label(text='The IP "{}" is invalid \nPlease enter a valid one'.format(value)),
                        size_hint=(0.9, 0.4))
                popup.open()

        elif key == "name":
            self.test.changeName(value)


if __name__ == "__main__":
    testAPP = TestApp()
    testAPP.run()
```

A3 Verzeichnis betreuter Studienarbeiten

Die nachstehend aufgeführten studentischen Arbeiten wurden im Kontext der vorliegenden Dissertation am Lehrstuhl für Produktentstehung der Universität Paderborn angefertigt. Die Definition der Zielsetzung, die Bearbeitung sowie die Auswertung, Interpretation und Visualisierung von Ergebnissen erfolgten unter wissenschaftlicher Anleitung der Betreuenden Prof. Dr.-Ing. Iris Gräßler und Alexander Pöhler. Die erzielten Ergebnisse sind zum Teil in die Dissertation eingeflossen.

[Min15] Min Rao (Alexander Pöhler): Entwicklung eines mechatronischen Demonstrators zum Einsatz im Produktionslabor. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Bachelorarbeit, 2015

[Gou16] Abdullah Gouda (Alexander Pöhler): Development of a decentralized production control-scheduling algorithm. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Masterarbeit, 2016

[Bel17] Julia Belke (Alexander Pöhler): Entwicklung einer Simulationsumgebung für dezentrale Produktionssteuerungen. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Studienarbeit, 2017

[Gra19] Tobias Grauthoff (Alexander Pöhler): Entwicklung einer Simulationsplattform zur Validierung von Forschungsergebnissen zu Lean Production und Arbeit 4.0. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Studienarbeit, 2019

[Voß15] Marleen Voß (Alexander Pöhler): Entwicklung eines Laborkonzeptes zur Veranschaulichung von Produktionssteuerungen im Kontext von Industrie 4.0. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Bachelorarbeit, 2015

[Kif19] Katharina Kiffmeier (Alexander Pöhler): Digitale Abbildung von Beschäftigten im Rahmen der Produktionsplanung und -steuerung. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Studienarbeit, 2019

[Töt17] Sunny Tötemeier (Alexander Pöhler): Entwicklung einer dynamischen Montageprozesssteuerung unter Berücksichtigung individueller Anforderungen. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Bachelorarbeit, 2017

[Hei17] Christian Heinrichmeier (Alexander Pöhler): Entwicklung mechatronischer Demonstratoren zur Veranschaulichung von Industrie 4.0 und Systems Engineering. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Masterarbeit, 2017

[Anw16] Mohamed Anwar (Alexander Pöhler): Development of decentralized production management systems based on Cyber Physical Production Sytems. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Masterarbeit, 2016

[Koc15] Uwe Koch (Alexander Pöhler): Menschen-zentrierter Industrie 4.0-Arbeitsplatz. Universität Paderborn, Heinz Nixdorf Institut / Lehrstuhl für Produktentstehung, unveröffentlichte Masterarbeit, 2015

Zudem haben Niklas Büker, Tobias Grauthoff, Julia Belke und Abdullah Gouda diese Arbeit durch ihre Tätigkeiten als studentische, bzw. wissenschaftliche Hilfskräfte unterstützt. Das Peer-to-Peer-Framework basiert auf einer Projektarbeit von Marcel Wellpott. Zudem hat Fabian Ritter diese Arbeit im Rahmen seiner Tätigkeiten am Heinz Nixdorf Institut unterstützt.

Erklärung zur Zitation von Inhalten aus studentischen Arbeiten

In Ergänzung zu meinem Antrag auf Zulassung zur Promotion in der Fakultät für Maschinenbau der Universität Paderborn erkläre ich gemäß §11 der Promotionsordnung und unter Beachtung der Regelung zur Zitation studentischer Arbeiten:

Die von mir vorgelegte Dissertation habe ich selbstständig verfasst, und ich habe keine anderen als die dort angegebenen Quellen und Hilfsmittel benutzt. Es sind Inhalte studentischen Ursprungs (studentische Arbeiten) in dieser Dissertation enthalten.

Ich habe die verwendeten Arbeiten entsprechend der Regelung „Zitation aus studentischen Arbeiten in Dissertationen“ zitiert.

Paderborn, den 11.12.2022:

Das Heinz Nixdorf Institut – Interdisziplinäres Forschungszentrum für Informatik und Technik

Das Heinz Nixdorf Institut ist ein Forschungszentrum der Universität Paderborn. Es entstand 1987 aus der Initiative und mit Förderung von Heinz Nixdorf. Damit wollte er Ingenieurwissenschaften und Informatik zusammenführen, um wesentliche Impulse für neue Produkte und Dienstleistungen zu erzeugen. Dies schließt auch die Wechselwirkungen mit dem gesellschaftlichen Umfeld ein.

Die Forschungsarbeit orientiert sich an dem Programm „Dynamik, Mobilität, Vernetzung: Eine neue Schule des Entwurfs der technischen Systeme von morgen“. In der Lehre engagiert sich das Heinz Nixdorf Institut in Studiengängen der Informatik, der Ingenieurwissenschaften und der Wirtschaftswissenschaften.

Heute wirken am Heinz Nixdorf Institut acht Professoren mit insgesamt 130 Mitarbeiterinnen und Mitarbeitern. Pro Jahr promovieren hier etwa 15 Nachwuchswissenschaftlerinnen und Nachwuchswissenschaftler.

Heinz Nixdorf Institute – Interdisciplinary Research Centre for Computer Science and Technology

The Heinz Nixdorf Institute is a research centre within the Paderborn University. It was founded in 1987 initiated and supported by Heinz Nixdorf. By doing so he wanted to create a symbiosis of computer science and engineering in order to provide critical impetus for new products and services. This includes interactions with the social environment.

Our research is aligned with the program “Dynamics, Mobility, Integration: Enroute to the technical systems of tomorrow.” In training and education the Heinz Nixdorf Institute is involved in many programs of study at the Paderborn University. The superior goal in education and training is to communicate competencies that are critical in tomorrows economy.

Today eight Professors and 130 researchers work at the Heinz Nixdorf Institute. Per year approximately 15 young researchers receive a doctorate.

Zuletzt erschienene Bände der Verlagsschriftenreihe des Heinz Nixdorf Instituts

Bd. 383 KAGE, M.: Systematik zur Positionierung in technologieinduzierten Wertschöpfungsnetzwerken. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 383, Paderborn, 2018 – ISBN 978-3-947647-02-6

Bd. 384 DÜLME, C.: Systematik zur zukunftsorientierten Konsolidierung variantenreicher Produktprogramme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 384, Paderborn, 2018 – ISBN 978-3-947647-03-3

Bd. 385 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 14. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 8. und 9. November 2018, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 385, Paderborn, 2018 – ISBN 978-3-947647-04-0

Bd. 386 SCHNEIDER, M.: Spezifikationstechnik zur Beschreibung und Analyse von Wertschöpfungssystemen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 386, Paderborn, 2018 – ISBN 978-3-947647-05-7

Bd. 387 ECHTERHOFF, B.: Methodik zur Einführung innovativer Geschäftsmodelle in etablierten Unternehmen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 387, Paderborn, 2018 – ISBN 978-3-947647-06-4

Bd. 388 KRUSE, D.: Teilautomatisierte Parameteridentifikation für die Validierung von Dynamikmodellen im modellbasierten Entwurf mechatronischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 388, Paderborn, 2019 – ISBN 978-3-947647-07-1

Bd. 389 MITTAG, T.: Systematik zur Gestaltung der Wertschöpfung für digitalisierte hybride Marktleistungen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 389, Paderborn, 2019 – ISBN 978-3-947647-08-8

Bd. 390 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 15. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 21. und 22. November 2019, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 390, Paderborn, 2019 – ISBN 978-3-947647-09-5

Bd. 391 SCHIERBAUM, A.: Systematik zur Ableitung bedarfsgerechter Systems Engineering Leitfäden im Maschinenbau. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 391, Paderborn, 2019 – ISBN 978-3-947647-10-1

Bd. 392 PAI, A.: Computationally Efficient Modelling and Precision Position and Force Control of SMA Actuators. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 392, Paderborn, 2019 – ISBN 978-3-947647-11-8

Bd. 393 ECHTERFELD, J.: Systematik zur Digitalisierung von Produktprogrammen. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 393, Paderborn, 2020 – ISBN 978-3-947647-12-5

Bd. 394 LOCHBICHLER, M.: Systematische Wahl einer Modellierungstiefe im Entwurfsprozess mechatronischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 394, Paderborn, 2020 – ISBN 978-3-947647-13-2

Bd. 395 LUKEI, M.: Systematik zur integrativen Entwicklung von mechatronischen Produkten und deren Prüfmittel. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 395, Paderborn, 2020 – ISBN 978-3-947647-14-9

Bd. 396 KOHLSTEDT, A.: Modellbasierte Synthese einer hybriden Kraft-/Positionsregelung für einen Fahrzeugachsprüfstand mit hydraulischem Hexapod. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 396, Paderborn, 2021 – ISBN 978-3-947647-15-6

Bezugsadresse:
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn

Zuletzt erschienene Bände der Verlagsschriftenreihe des Heinz Nixdorf Instituts

Bd. 397 DREWEL, M.: Systematik zum Einstieg in die Plattformökonomie. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 397, Paderborn, 2021 – ISBN 978-3-947647-16-3

Bd. 398 FRANK, M.: Systematik zur Planung des organisationalen Wandels zum Smart Service-Anbieter. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 398, Paderborn, 2021 – ISBN 978-3-947647-17-0

Bd. 399 KOLDEWEY, C.: Systematik zur Entwicklung von Smart Service-Strategien im produzierenden Gewerbe. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 399, Paderborn, 2021 – ISBN 978-3-947647-18-7

Bd. 400 GAUSEMEIER, J. (Hrsg.): Vorausschau und Technologieplanung. 16. Symposium für Vorausschau und Technologieplanung, Heinz Nixdorf Institut, 2. und 3. Dezember 2021, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 400, Paderborn, 2021 – ISBN 978-3-947647-19-4

Bd. 401 BRETZ, L.: Rahmenwerk zur Planung und Einführung von Systems Engineering und Model-Based Systems Engineering. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 401, Paderborn, 2021 – ISBN 978-3-947647-20-0

Bd. 402 WU, L.: Ultrabreitbandige Sampler in SiGe-BiCMOS-Technologie für Analog-Digital-Wandler mit zeitversetzter Abtastung. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 402, Paderborn, 2021 – ISBN 978-3-947647-21-7

Bd. 403 HILLEBRAND, M.: Entwicklungssystematik zur Integration von Eigenschaften der Selbstheilung in Intelligente Technische Systeme. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 403, Paderborn, 2021 – ISBN 978-3-947647-22-4

Bd. 404 OLMA, S.: Systemtheorie von Hardware-in-the-Loop-Simulationen mit Anwendung auf einem Fahrzeugachsprüfstand mit parallelkinematischem Lastsimulator. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 404, Paderborn, 2022 – ISBN 978-3-947647-23-1

Bd. 405 FECHTELPETER, C.: Rahmenwerk zur Gestaltung des Technologietransfers in mittelständisch geprägten Innovationsclustern. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 405, Paderborn, 2022 – ISBN 978-3-947647-24-8

Bd. 406 OLEFF, C.: Proaktives Management von Anforderungsänderungen in der Entwicklung komplexer technischer Systeme. Dissertation, Fakultät für Maschinenbau, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 406, Paderborn, 2022 – ISBN 978-3-947647-25-5

Bd. 407 JAVED, A. R.: Mixed-Signal Baseband Circuit Design for High Data Rate Wireless Communication in Bulk CMOS and SiGe BiCMOS Technologies. Dissertation, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 407, Paderborn, 2022 – ISBN 978-3-947647-26-2

Bd. 408 DUMITRESCU, R, KOLDEWEY, C.: Datengestützte Projektplanung. Fachbuch. Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Verlagsschriftenreihe des Heinz Nixdorf Instituts, Band 408, Paderborn, 2022 – ISBN 978-3-947647-27-9

Bezugsadresse:
Heinz Nixdorf Institut
Universität Paderborn
Fürstenallee 11
33102 Paderborn

In dieser Arbeit wird die Entwicklung einer dezentralen Produktionssteuerung mit digitaler Repräsentation zur Anwendung in cyber-physischen Produktionssystemen (CPPS) beschrieben. Die dezentrale Produktionssteuerung geschieht hierbei selbstorganisiert, da sie die Einlastung und Umplanung von Aufträgen sowie die Koordination zwischen den einzelnen Produktionselementen selbstständig übernimmt. Dies erlaubt die automatisierte Steuerung von CPPS. Dabei spielen die Beschäftigten in dieser Betrachtung eine wesentliche Rolle. Ihre Bedürfnisse und Wünsche werden von dem Steuerungssystem berücksichtigt. Dafür wurde das Konzept der digitalen Repräsentation von Beschäftigten entwickelt. Diese Repräsentation erlaubt den individuellen Eingriff von Beschäftigten bei Entscheidungen von dem Steuerungssystem.

Das automatisierte dezentrale Steuerungssystem wurde in einer Laborumgebung, die zu einem CPPS umgebaut wurde, implementiert. Darauf basierend wurde eine zweistufige Validierung durchgeführt. Ein Teil der Validierung findet durch eine reale Umsetzung in einem Labor statt, bei der ein Vergleich zu dem vorher im Labor eingesetzten zentralen Produktionssteuerungssystem gezogen wird. Die zweite Validierung erfolgt mit Hilfe einer Simulationsumgebung. Dabei konnte die Implementierung der Selbstorganisation und eine Steigerung der Liefertreue der dezentralen Produktionssteuerung gegenüber heutzutage üblicherweise eingesetzten Verfahren nachgewiesen werden.