Unterstützung der Wandlungsfähigkeit von Produktionsanlagen durch innovative Softwaresysteme

Der Technischen Fakultät der
Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR - INGENIEUR

vorgelegt von

Dipl.-Inf. Matthias Alexander Weber

Erlangen – 05/2007

Als Dissertation genehmigt von der Technischen Fakultät der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der Einreichung:	21.05.2007
Tag der Promotion:	26.10.2007
Dekan:	Prof. Dr.-Ing. J. Huber
Berichterstatter:	Prof. Dr.-Ing. K. Feldmann
	Prof. Dr.-Ing. A. Verl, Universität Stuttgart

Vorwort und Danksagung

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Friedrich-Alexander-Universität Erlangen-Nürnberg.

Herrn Prof. Dr.-Ing. Klaus Feldmann, dem Leiter dieses Lehrstuhls am Institut für Maschinenbau, danke ich sehr herzlich für die engagierte Förderung bei der Durchführung meiner Arbeit, die vielfältigen wissenschaftlichen Freiräume und das mir entgegengebrachte Vertrauen.

Herrn Prof. Dr.-Ing. Alexander Verl, Leiter des Instituts für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart und Leiter des Fraunhofer-Instituts für Produktionstechnik und Automatisierung danke ich für die Übernahme des Korreferats und die konstruktiven Anmerkungen.

Ferner gilt mein Dank Herrn Prof. Dr.-Ing. Albert Weckenmann für die Übernahme des Prüfungsvorsitzes und Herrn Prof. Dr.-Ing. Wolfgang Schröder-Preikschat für die Teilnahme als weiterer Prüfer.

All meinen Kollegen sei für die stets sehr gute Zusammenarbeit, die zahlreichen fachlichen Diskussionen, die gemeinsam als Team durchgeführten Kooperationsprojekte und die stets angenehme Atmosphäre gedankt. Besonders hervorheben möchte ich an dieser Stelle Herrn Dipl.-Inf. Wolfgang Wolf, dessen geduldiges Zuhören und die Bereitschaft zu intensiven Diskussionen mir eine große Hilfe waren sowie Herrn Dr.-Ing. Stefan Junker und Herrn Dr.-Ing. Stefan Lang für die Beantwortung und Diskussion unzähliger akademischer Fragen zu Themen des Maschinenbaus.

Ferner möchte ich mich bei allen Studenten bedanken, die mich während meiner Zeit am Lehrstuhl tatkräftig im Rahmen verschiedenster Projekte unterstützt haben. Namentlich erwähnen möchte ich Herrn Dipl.-Inf. Gerald Meckl und Herrn Tobias Klier.

Ein ganz besonderer Dank gebührt Frau Sabine Weinhold für die moralische Unterstützung sowie für das zeitintensive Korrekturlesen und die vielzähligen Anmerkungen, die sehr zum Gelingen der Arbeit beigetragen haben.

Einen besonderen Dank möchte ich an meine Mutter richten, die mich von Jugend an ermutigt und mich auf meinen Weg gebracht hat.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **4**
2.1 Produktion 4
2.1.1 Produktionsprozesse und -logistik 4
2.1.2 Flusssysteme 6
2.1.3 Strukturierungskonzepte 6
2.2 Softwaresysteme in der Produktion 7
2.2.1 Steuerungen 8
2.2.2 Informationsverwaltung 9
2.2.3 Informationserzeugung und -verarbeitung 10
2.2.4 Interaktion von Softwaresystemen 10
2.3 Steuerungstechnik 10
2.3.1 Rechentechnik 10
2.3.2 Kommunikationstechnik 11
2.4 Produktionsanlagen im Wandel 11
2.4.1 Wandlungstreiber 12
2.4.2 Wandlungsfähigkeit von Produktionsanlagen 13
2.4.3 Lösungsansätze in Technik und Forschung 14
2.4.4 Defizite und Ausleitung von Handlungsbedarfen 17

3 Vorüberlegungen zur Gestaltung wandelbarer Produktionsanlagen **18**
3.1 Überblick über die allgemeine Systemtheorie 18
3.1.1 Element 18
3.1.2 System 20
3.1.3 Teilsysteme 21
3.1.4 Schnittstellen und Schnittstellentypen 21
3.2 Steuerungsmotivierte Systemklassifikation 23
3.3 Gesteuerte Systeme und Steuerungsmodelle 24
3.4 Schlussfolgerungen zur Gestaltung der Wandelbarkeit 26
3.5 Systemkonzept für wandelbare Produktionsanlagen 31
3.5.1 Basiselemente 31
3.5.2 Zwillings-Basiselemente 32
3.5.3 Elemente 33
3.5.4 Zwillings-Elemente 34
3.5.5 Verbünde 34

4 Konzeption einer wandelbaren Systemlandschaft **35**

4.1 6-Ebenen$^+$-Modell . . . 35
4.1.1 Hauptebene 1 – Produktionsanlage . . . 37
4.1.2 Hauptebene 2 – Lager, Produktionslinien und -inseln, Transportsysteme . . . 37
4.1.3 Hauptebene 3 – (Sub-)Zellen . . . 38
4.1.4 Hauptebene 4 – Produktions- und Transfereinrichtungen . . . 39
4.1.5 Hauptebene 5 – (Sub-)Module . . . 39
4.1.6 Hauptebene 6 – Basismodule . . . 40
4.2 Strukturierte und unstrukturierte Bereiche . . . 40
4.3 Dezentrale Steuerungstechnik . . . 41
4.4 Informationelle Vernetzung . . . 42
4.4.1 Übergreifendes Adressierungsschema . . . 43
4.4.2 Identifikation von Modellelementen . . . 44
4.4.3 Kommunikations-Middleware – Abstraktion von Kommunikation . . . 44
4.4.4 Interaktionsformen . . . 45
4.5 Wandelbare Steuerungssoftwaresysteme . . . 45
4.5.1 Steuerung und Regelung . . . 46
4.5.2 Prozesskoordination . . . 48
4.6 Selbstadaptierende generische Informationsdienste und allgemeine Informationssysteme . . . 49
4.6.1 Betriebsdatenerfassung . . . 49
4.6.2 Rezepte- und Modellverwaltung . . . 49
4.6.3 Arbeitsplanverwaltung . . . 50
4.6.4 Auftragsverwaltung für Produktionslinien und -inseln . . . 50
4.6.5 Ausfallsicherheit . . . 50

5 Konzeption einer auf Wandelbarkeit ausgerichteten Softwarearchitektur 52
5.1 Softwareparadigma „Componentware“ als Grundlage der Architekturkonzeption . . . 52
5.2 Interaktionsformen . . . 53
5.2.1 Funktions- und Prozeduraufruf . . . 54
5.2.2 Publish & Subscribe . . . 54
5.3 Architekturelemente . . . 57
5.3.1 Knoten . . . 57
5.3.2 Facette . . . 59
5.3.3 Module . . . 60
5.4 Verhaltensformen von Facetten . . . 61
5.5 Komplexe Softwareelemente und ihre Wandlungsfähigkeit . . . 64
5.6 Grundfacetten . . . 65
5.6.1 Dateisystemverwaltung . . . 65
5.6.2 Skriptfähiger Controller . . . 66
5.6.3 Digital-Eingang und -Ausgang . . . 67
5.6.4 Analog-Eingang und -Ausgang . . . 69
5.6.5 Zeichenketten-Eingang und -Ausgang . . . 72
5.7 Entität – Abgeschlossene Einheit im Systemverbund . . . 74

6 Selbsterkennung der Entitäten und Selbstkonfiguration des Kommunikations-

netzes **76**
6.1 Isomorphie zwischen Entitäten und Kommunikationsnetzwerk . . . 76
6.2 Kommunikationstechnische Voraussetzung . . . 77
6.3 Dezentrales Verfahren zur Anwesenheitserkennung . . . 78
6.4 Dezentrales Verfahren zur Ermittlung der strukturellen Beziehungen . . . 81
6.5 Ansätze zur Reduktion der Nachrichten . . . 81
6.6 Automatische Hierarchiebildung von Entitäten . . . 83
6.6.1 Separatoren zur Hierarchiebildung . . . 84
6.6.2 Verfahren zur Ermittlung der Zonenzugehörigkeit . . . 84
6.6.3 Umsetzung des Adressierungsschemas . . . 86
6.7 Selbstkonfiguration des Kommunikationsnetzes . . . 88
6.8 Integration in das übergeordnete Kommunikationsnetz der Fabrik . . . 90

7 Selbstorganisation von Stoffflüssen **94**
7.1 Klassifikation von Transfersystemen . . . 94
7.2 Funktionsprinzip der Wegfindung . . . 95
7.3 Funktionsprinzip von Verzweigungen . . . 97
7.4 Generisches Steuerungsmodell für Verzweigungen . . . 99
7.4.1 Aufbau der spezifischen Facetten . . . 100
7.4.2 Informationelle Verknüpfung und Steuerungsablauf als Kreuzung . . . 102
7.4.3 Informationelle Verknüpfung und Steuerungsablauf als Weiche . . . 105
7.4.4 Applikation auf freinavigierende Transportsysteme . . . 105
7.4.5 Ansatzpunkte zur effizienten Steuerung der Stoffflüsse . . . 105
7.5 Dezentrale Strategien zur Selbstorganisation von Stoffflüssen . . . 106
7.5.1 Stationsorientiertes Routing . . . 107
7.5.2 Tätigkeitsorientiertes Routing . . . 108
7.5.3 Tätigkeitsorientiertes Routing mit Lastausgleich . . . 108
7.5.4 Werkstückzentriertes, angebotsorientiertes Routing . . . 109
7.6 Simulationsstudie zur Bewertung der Flussstrategien . . . 110
7.6.1 Simulationsmodelle . . . 110
7.6.2 Simulationsexperimente . . . 112
7.6.3 Simulationsergebnisse . . . 112
7.7 Integration der besten Strategie . . . 113

8 Selbstorganisation wandlungsfähiger Produktionsanlagen **116**
8.1 Transferhilfsmittel . . . 116
8.2 Generisches Lager . . . 118
8.2.1 Modell zur Ablaufsteuerung . . . 119
8.2.2 Eingriffspunkte zur effizienten Steuerung . . . 122
8.2.3 Unterstützung der Wandelbarkeit . . . 123
8.3 Produktionslinien und -inseln . . . 123
8.3.1 Arbeitsstationen . . . 125
8.3.2 Auftragsverwaltung . . . 126
8.3.3 Informationelle Verknüpfung und Steuerungsablauf . . . 127
8.3.4 Identitäten initialer Gegenstände . . . 128

8.3.5 Eingriffspunkte zur effizienten Steuerung . . . 128
8.3.6 Unterstützung der Wandelbarkeit . . . 129
8.4 Selbstorganisation der Produktionsanlage . . . 131
8.4.1 Kopplung mehrerer Systeme . . . 131
8.4.2 Ankopplung an die Produktionsplanung und -steuerung . . . 131
8.4.3 Unterstützung der Wandelbarkeit . . . 133

9 Selbstadaptierende generische Informationsdienste und allgemeine Softwaresysteme 134
9.1 Generische Informationsdienste . . . 134
9.1.1 Grundaufbau der Informationsdienste . . . 135
9.1.2 Redundanz zur Absicherung von Ausfällen . . . 135
9.1.3 Partitionierung und Verschmelzung von Dienstgruppen durch Wandlung . . . 138
9.2 Katalogdienst . . . 140
9.3 Arbeitsplanverwaltung . . . 143
9.4 Rezepteverwaltung . . . 143
9.5 Modellverwaltung . . . 144
9.6 Betriebsdatenverwaltung . . . 145
9.7 Allgemeine Softwaresysteme . . . 147

10 Referenzimplementierung der Softwarearchitektur und Evaluation der Konzepte 149
10.1 Wahl der Steuerungstechnik . . . 149
10.2 Aufbau des Labors für verteilte Steuerungssysteme . . . 151
10.3 Softwaresysteme für Entitäten . . . 153
10.4 Entwicklungswerkzeuge . . . 158
10.5 Erweiterung von Python zur Skriptsprache für Rezepte . . . 164
10.6 Materialfluss für verzweigte Transportsysteme . . . 165
10.7 Prototypischer Leitstand . . . 166
10.8 Das Labor im Betrieb . . . 169

11 Zusammenfassung 172

Literaturverzeichnis 178

Kapitel 1

Einleitung

Mit Blick auf die voranschreitende Globalisierung der Wirtschaftsräume und der einhergehenden Öffnung heimischer Märkte für Produkte aus Niedrig- und Niedrigstlohnländern stehen insbesondere Produzenten aus Hochlohnländern vor großen Herausforderungen. Sie begegnen diesen häufig mit der Erschließung neuer Märkte, deren Merkmal regionale Eigenheiten und Kundenbedürfnisse sind, sowie mit einer verstärkten Kundenorientierung durch individualisierte Produkte. Dadurch entsteht eine zunehmende Produktdiversifikation mit steigender Variantenvielfalt und sinkenden Stückzahlen. Die Konsequenzen hieraus sind heterogene Produktmixe und Produktionsabläufe, die vielerseits als turbulent charakterisiert werden [3, 11, 21, 49, 68, 115, 127].

Eine weitere Strategie – die fortwährende Generierung von Produktinnovationen bei abnehmenden Entwicklungszeiten – bewirkt sinkende Produktlebenszyklen und eine ansteigende Zahl von Veränderungen in den Produktspektren. Mit ihnen ändern sich auch Produktionsanlagen, da sie strukturell, technologisch, kapazitiv, im Automatisierungsgrad oder zumindest in der Ablauforganisation an die veränderten Situationen angepasst werden müssen. Die Turbulenz, der eine Produktion dadurch ausgesetzt ist, nimmt so weiter zu.

Aufgrund steigender Veränderungsfrequenzen gewinnen zeitliche Aufwendungen und Investitionen für Anpassungen vermehrt an Bedeutung. Die Fähigkeit einer Produktionsanlage zum schnellen und effektiven Umbau mit geringen Stillstandszeiten erlangt somit einen erheblichen Stellenwert. Kurze Planungs-, Entwicklungs- und Inbetriebnahmezeiten sind daher angestrebte Ziele. In diesem Zusammenhang steht der Begriff der *Wandlungsfähigkeit*, der nach [114, 115] „das Vermögen einer Fabrik, ausgehend von internen und externen Auslösern, aktiv ihren Aufbau auf allen Ebenen“ „bei geringen Kosten, Sach- und Personalaufwand“ „verändern zu können“ umschreibt. Nach [112] wird ein System als wandlungsfähig bezeichnet, „wenn es aus sich selbst heraus über gezielt einsetzbare Prozess- und Strukturvariabilität sowie Verhaltensvariabilität verfügt“. Darüber hinaus sind wandlungsfähige Systeme „in der Lage, neben reaktiven Anpassungen auch antizipative Eingriffe vorzunehmen“ [112].

Als Kern der wandlungsfähigen Gestaltung wird häufig auf technischer Seite die Modularisierung von Produktionsanlagen in verschiedenen Ebenen angesehen [41, 42, 91, 115]. Systemmodule, mit klar definierten Schnittstellen und wohl definierter Funktionalität ausgestattet, erlauben eine effiziente Austauschbarkeit. Allerdings fokussieren bekannte Konzepte weitestgehend auf mechanische Aspekte – Steuerungstechnik und Softwaresysteme, die in modernen Produktionsanlagen eine wichtige Position einnehmen, werden in der Regel nicht explizit berücksichtigt.

Jedoch erfüllen Steuerungstechnik und Softwaresysteme im Zusammenspiel – horizontal in und vertikal über alle Ebenen einer Produktionsanlage – wesentliche Aufgaben zur Durchführung der Produktionsabläufe. Sie koordinieren, steuern und regeln Prozesse, verwalten Produktionsaufträge, steuern

Stoff- und Energieflüsse, erfassen, verarbeiten und aggregieren Betriebsdaten, dienen der Fortschrittsermittlung, implementieren Mensch-Maschine-Schnittstellen, sind Bestandteil der Qualitätssicherung und des Produkt-Trackings, u.v.m. Sie tragen zum Erreichen von Produktionszielen, wie z.B. Ausbringungsmengen, Termintreue oder Servicegrad, ebenso bei, wie zur aktiven Kompensierung von Störungen oder der Adaption an kurzfristige Änderungen von Produktmixen bzw. Losgrößen.

Steuerungstechnik und Softwaresysteme sind damit, insbesondere im turbulenten Umfeld, maßgeblich an der Durchführung von Produktionsaufgaben beteiligt. Sie sind deshalb bei der wandlungsfähigen Gestaltung von Produktionsanlagen zu berücksichtigen.

Zielsetzung und Aufbau der Arbeit

Das Ziel der vorliegenden Arbeit besteht darin, die Wandlungsfähigkeit und -effizienz von Produktionsanlagen durch eine ganzheitliche Betrachtung der Modularisierung zu erhöhen. Dazu wird [8] folgend Produktion als ein geschachtelter Komplex aus verschiedenartigen Flusssystemen aufgefasst, in dem Stoffe, Energien, Signale und Informationen fließen und transformiert werden.

Ausgehend von der allgemeinen Systemtheorie werden in einer theoretischen Analyse mit Schwerpunkt auf Steuerungstechnik und Softwaresystemen Schlussfolgerungen für die Gestaltung der Wandelbarkeit erarbeitet und ein theoretisches Systemkonzept für durchgängig wandelbare Produktionsanlagen aufgestellt (Kap. 3). Der theoretische Ansatz wird anschließend in das Konzept einer wandelbaren Systemlandschaft übergeführt, das Produktionsanlagen als mehrstufige, hierarchische Systeme begreift. Wandelbare Steuerungstechnik und Softwaresysteme finden in diesem Konzept auf allen Ebenen als horizontal und vertikal vernetzte, interagierende und kooperierende Einheiten Anwendung (Kap. 4).

Für die Entwicklung dieser Softwaresysteme wird eine auf Wandelbarkeit ausgerichtete Softwarearchitektur vorgestellt, die strikt auf den Ergebnissen der theoretischen Analyse aufbaut und dabei Aspekte der Systemlandschaft berücksichtigt. Sie beruht auf den Paradigmen der Softwarekomponenten und objektorientierten Programmierung. Die Architektur zielt darauf ab, Software durch Kopplung von Modulen aus einem Baukastensystem aufzubauen. Grundbausteine eines solchen Baukastens wurden exemplarisch entwickelt (Kap. 5).

Wesentlichen Anteil zur Reduzierung von Planungs-, Entwicklungs- und Inbetriebnahmezeiten, bei gleichzeitiger Sicherstellung effektiver Produktionsabläufe, haben dezentrale Verfahren zu Selbsterkennung, Selbstkonfiguration und Selbstorganisation. Hierfür werden erarbeitete Lösungen zur Erkennung der Anwesenheit von Systemmodulen, ihren strukturellen Verknüpfungen sowie ihren hierarchischen Beziehungen aufgezeigt und erläutert. Basierend auf Strukturinformationen erfolgt die Vorstellung eines ebenfalls dezentralen Verfahrens zur automatischen Konfiguration der Anlagenkommunikationsnetze, das physisch gewandelte Produktionsanlagen ohne weitere manuelle Tätigkeiten in einen kommunikationsbereiten Zustand versetzt (Kap. 6).

Mit diesen Grundlagen ist es möglich, Stoffflüsse selbstorganisierend zu gestalten. Hierzu wird ein Ansatz präsentiert, der einerseits aus einer generischen Ablaufsteuerung besteht, die auf Kreuzungen, Weichen und autonome Transportfahrzeuge anwendbar ist, und andererseits eine dezentrale Strategie zur effektiven Stoffweiterleitung an Verzweigungen der Transfernetze beinhaltet. Diese verfügt über einen inhärenten Mechanismus zur Kompensation von Stationsausfällen und wurde aus vier einfachen

Strategien ausgesucht, die konzipiert und in einer Simulationsstudie gegenübergestellt wurden (Kap. 7).

Eine Ausdehnung der Selbstorganisation auf einzelne Produktionssysteme und darüber hinaus auf die gesamte Produktionsanlage, unter der Einbeziehung von Lagern, Auftragsverwaltungen und Arbeitsstationen, erfolgt im Anschluss. Wesentliche Ziele bestehen in der Generierung eines stetigen Werkstückflusses, bei automatischem Ausgleich von Störungen sowie in der Anpassung der Systeme an geänderte Randbedingungen durch Wandlung (Kap. 8).

Zum Betrieb von Produktionsanlagen werden in der Regel auch Softwaresysteme zur Informationsverwaltung benötigt, die sich in *generische Informationsdienste* und *allgemeine Softwaresysteme* einteilen lassen. Sie interagieren mit anderen Systemen und können daher auch von Wandlungsvorgängen betroffen sein. Es werden selbstadaptierende Informationsdienste vorgestellt, die eine wandlungsfähige Produktion unterstützen, und für allgemeine Softwaresysteme Wege aufgezeigt, wie sie in die vorgestellten Konzepte integriert werden können (Kap. 9).

Im Rahmen der Arbeit wurden für die auf Wandelbarkeit ausgerichtete Softwarearchitektur eine Referenzimplementierung realisiert und zur aufwandsarmen Entwicklung von Softwaresystemen geeignete Werkzeuge entworfen. Die entwickelten dezentralen Strategien wurden in einem eigens eingerichteten Labor für verteilte Steuerungssysteme evaluiert, dessen Fähigkeiten anhand von Beispielen illustriert werden (Kap. 10).

Als erster Schritt gibt Kapitel 2 zunächst einen Kurzüberblick über die Produktion, das Konzept der Flusssysteme sowie die Strukturierung von Produktionssystemen und beleuchtet Softwaresysteme und Steuerungstechnik. Die Thematisierung von Produktionsanlagen im Wandel führt zu einer Ausleitung von Handlungsbedarfen und motiviert die weiteren Bestrebungen.

Kapitel 2

Grundlagen

Die Primäraufgabe einer Fabrik ist die Herstellung von Erzeugnissen, die durch logistisch gekoppelte Produktionsprozesse erfüllt wird [18, 36, 38, 91]. Produktionsanlagen und ihre Abläufe lassen sich als Flusssysteme beschreiben, da stoffliche Objekte, Energien, Signale und Informationen durch sie hindurch „fließen“ [8]. In Abhängigkeit der Produkt- und Variantenvielfalt sowie der zu erreichenden Zielgrößen (z.B. Ausbringung und Liefertreue) eignen sich unterschiedliche Konzepte für die Strukturierung einer Produktionsanlage [91]. Die Koordinierung und Überwachung der logistischen Abläufe und Prozesse sowie die Verwaltung und Bereitstellung von Informationen wird in modernen Produktionsanlagen durch rechnergestützte Softwaresysteme erbracht, die eine Symbiose mit Rechentechnik bilden. Aufgrund der Vielfältigkeit der Aufgaben und einer steigenden Komplexität erfolgt eine zunehmende Dezentralisierung dieser Systeme, unter Verknüpfung der eingesetzten Rechentechnik zu Verbünden mittels Kommunikationstechnik. Rechentechnik und Kommunikationstechnik werden im Folgenden unter dem Überbegriff *Steuerungstechnik* zusammengefasst.

Externe und interne Treiber sind für den Wandlungsbedarf von Produktionsanlagen verantwortlich [115]. Ändern sich die Anforderungen an die Produktion derart, dass diesen durch elastische Flexibilität allein nicht entsprochen werden kann, muss ein struktureller Umbau erfolgen [91, 115]. Hieraus folgen auch Konsequenzen für die Softwaresysteme, die an die neuen Produktionsstrukturen anzupassen sind. Um Produktionsanlagen aus technischer Sicht effizient wandeln zu können, ist eine entsprechende Gestaltung unabdingbar. Hierbei müssen Aspekte des mechanischen Aufbaus, der Energieversorgung, der Informationsverarbeitung, der Automatisierung sowie der Organisation berücksichtigt werden [21, 41, 68, 91, 115].

2.1 Produktion

2.1.1 Produktionsprozesse und -logistik

Die Erzeugung von Gütern geschieht durch Produktionsprozesse, die durch Produktionslogistik miteinander gekoppelt sind. Ihre Aufgabe ist es, durch ein zielgerichtetes Zusammenwirken Erzeugnisse in gewünschter Menge und Qualität, zeitlich determiniert, herzustellen.

Zur Erfüllung dieser Aufgabe werden Betriebsmittel, Werkzeuge oder Verfahren eingesetzt [91]. Betriebsmittel sind im Allgemeinen Anlagen, Geräte oder Einrichtungen, die zur betrieblichen Leistungserstellung dienen [109]. Sie lassen sich nach ihrem Zweck in Fertigungs-, Montage-, Logistik-, Informations-, Kommunikations-, Sicherheits-, Qualitätssicherungs- sowie Ver- und Entsorgungseinrichtungen kategorisieren [70, 115]. Neben dieser Einordnung ist auch eine Klassifikation in manuelle,

teilautomatisierte und vollautomatisierte Betriebsmittel möglich, wobei unabhängig des Automatisierungsgrades eine Rechnerunterstützung vorhanden sein kann.

Einzelne bzw. eine Gruppe von Fertigungs- bzw. Montagebetriebsmitteln bilden Arbeitsstationen und agieren ggf. kooperierend in einem gemeinsamen Arbeitsraum. Mehrere Arbeitsstationen können aus logistischen Gründen oder zur kooperativen Durchführung von Prozessschritten in gemeinsamen Arbeitsräumen zu Zellen zusammengefasst sein.

Zur Durchführung ihrer Aufgaben müssen Betriebsmittel mit Stoffen, Energien und Informationen versorgt werden. Zu den Stoffen zählen bei der Fertigung Roh- und Fertigteile, bei der Montage Kauf- und Normteile sowie Werkstücke und allgemein alle zur Herstellung erforderlichen Werkzeuge, Vorrichtungen sowie notwendige Hilfsstoffe. Hinzukommen bei der Durchführung von Prozessen entstehende Abfälle. Energien umfassen Elektrizität, Pneumatik, Vakuum und Hydraulik. Die Durchführung der Prozesse ist produkt- und variantenspezifisch. Die Bereitstellung aller Informationen zur adäquaten Steuerung von Geräten und Maschinen bzw. zur Anleitung von Menschen sind daher erforderlich [8, 91].

Die Versorgung der Fertigungs- bzw. Montagebetriebsmittel übernehmen Betriebsmittel der Logistik, die ihrerseits durch eine vorgelagerte Logistik mit Energien und Informationen versorgt werden müssen, um das erforderliche Objekt zur richtigen Zeit, in der richtigen Menge und Qualität an den richtigen Ort zu befördern.

Neben dem Transfer erfüllt die Stofflogistik Aufgaben des Umschlagens und Lagerns sowie des Handhabens, Kommissionierens und Verpackens [91], wobei häufig Transferhilfsmittel wie Paletten oder Werkstückträger Verwendung finden. In einer Produktionsanlage können eine Vielzahl stetiger und unstetiger Transfertechniken zum Einsatz gelangen, die in Förder- und Transporttechnologie eingeteilt werden und durch spezifische Transfermittel realisiert sind [48]. Fördern erfolgt kontinuierlich oder diskontinuierlich und ist zumeist derart autonom, dass der Mensch zur Ortsveränderung von Gegenständen nicht erforderlich ist. Transport unterscheidet sich insofern, als dass er immer diskontinuierlich stattfindet und in unterschiedlichen Automatisierungsgraden vorkommen kann. So werden Transporte sowohl von Menschen allein als auch von autonomen bzw. von Menschen gesteuerten oder von ihnen angetriebenen Transportfahrzeugen durchgeführt.

Realisierbar sind innerzelluläre bzw. -stationäre Stoffflüsse, vornehmlich zur Bereitstellung von Teilen, Material und Hilfsstoffen sowie interzelluläre bzw. -stationäre Stoffflüsse. Insbesondere bei Inter-Stoffflüssen werden Fördermittel häufig zur Überbrückung größerer Distanzen zu Förderketten verschaltet, wobei gleichzeitig unterschiedliche Techniken eingesetzt werden können. Zur Gestaltung flexibler Flüsse sind mehrere Förderketten zu komplexen Fördersystemen kombinierbar [48].

Die Bereitstellung von Energien erfolgt in der Regel über Leitungsnetze, die sich in ihrer strukturellen Komplexität unterscheiden, da der Ort der Energieerzeugung variieren kann. So stammt Elektrizität in der Regel aus dem allgemeinen Stromnetz der Fabrik, während Vakuum oder Druckluft zentral für die gesamte Anlage oder dezentral vor Ort in Zellen oder Arbeitsstationen erzeugt werden kann. Hydrauliksysteme sind dagegen in der Regel abgeschlossen und örtlich begrenzt.

2.1.2 Flusssysteme

Die Beschreibung von Produktionsanlagen als Flusssysteme basiert auf einer Theorie von Wirth [8, 91], wonach eine Produktionsaufgabe durch Aneinanderreihung sich qualitativ und quantitativ unterscheidender Einzelfunktionen zu Funktionsketten gelöst wird. Diese werden von stofflichen, energetischen und informationellen Flussgegenständen sowie Signalen durchlaufen. Die Einzelfunktionen lassen sich hierbei unabhängig von der Art des Flussgegenstands in die drei Grundfunktionen *Transformation*, *Transfer* und *Speichern* untergliedern, wobei Transformation während Transfer oder Speicherung als kombinierte Funktionen möglich sind. Unter Transformation wird hierbei allgemein das Verändern von Eigenschaften und Zuständen eines Flussgegenstands verstanden. Das Speichern dient der Bevorratung und Bereitstellung, der Transfer dem Überwechseln und Befördern zwischen zwei aufeinander folgenden Einrichtungen des Transformierens, Speicherns oder Transferierens. Die Grundfunktionen werden entweder durch technische Einrichtungen oder Menschen erbracht.

Die Theorie gründet sich auf nachfolgenden Definitionen. Unter einem *Fluss* wird die gerichtete zeitliche und räumliche Abfolge von Eigenschafts- und Zustandsänderungen eines bestimmten beweglichen *Flussgegenstands* verstanden, von dem mehrere zu *Flusseinheiten* zum Zwecke eines gesammelten Transfers oder Transformation gebunden werden können. Auf beide beweglichen Objekte wirken elementare oder komplexe *Flussfunktionen*, die eine Objektänderung bewirken. Insgesamt übt ein Fluss eine *Flussaufgabe* aus.

Flüsse und ihre Flussfunktionen werden von *Flusssystemen*, die aus gekoppelten *Flusssystem-Elementen* bestehen, realisiert. Diese setzen Flussfunktionen um, wobei Elemente des Transformierens und Transfers häufig (im Falle der Automatisierung immer) weitere Funktionen anderer Flüsse, etwa zur Energieversorgung, beinhalten. Elemente können strukturell zu *Flusssystem-Elemente-Komplexen* gekoppelt werden, wobei Flusselemente aller beteiligten Flusssysteme einer technischen Einrichtung zusammengefasst werden. Die so aufgebauten Flusssysteme lassen sich mit anderen Flusssystemen zu *Flusssystem-Komplexen* ebenfalls koppeln, wie sie Produktionsanlagen mit ihren verschiedenen Stoff-, Energie- und Informationsflüssen darstellen. *Kopplungen* von Flusssystemen und Flusssystemelementen erfolgen über interne und externe *Flusssystemschnittstellen*. Subelemente und -flusssysteme werden über die internen, Elemente und Flusssysteme über die externen Schnittstellen miteinander verknüpft.

Eine Produktionsanlage durchziehen vielzählige Flusssysteme unterschiedlicher Objektart, die sich als *Haupt-* und *Nebenflusssysteme* charakterisieren lassen. Hauptflusssysteme üben direkt Funktionen auf Erzeugnisse bzw. vorgelagerte Baugruppen aus, während Nebenflusssysteme alle dafür erforderlichen sekundären Leistungen erbringen. Veränderungen in den primären Flusssystemen ziehen unweigerlich Anpassungen in den Sekundären nach sich.

2.1.3 Strukturierungskonzepte

Angepasst an Erzeugnisvielfalt und geforderte Stückzahlen werden Produktflusssysteme nach unterschiedlichen Strukturierungskonzepten umgesetzt, die sich bei der Teilefertigung in die Grundstrukturen *Werkstatt-*, *Insel-*, *Reihen-* und *Fließfertigung* und bei der Montage in *Baustellen-*, *Gruppen-*, *Reihen-* und *Fließmontage* gliedern [23, 25, 61, 113]. Eine Kombination aus Teilefertigung und Montage bzw. verschiedener Strukturen ist möglich.

Die *Werkstattmontage* zeichnet sich durch eine Gruppierung gleichgearteter Fertigungseinrichtungen mit ungerichtetem Materialfluss aus. Die *Fertigungsinsel* fasst stattdessen die für die Bearbeitung konkreter Teile erforderlichen Fertigungseinrichtungen für unterschiedliche Produktionsprozesse zusammen, wobei ungerichtete und gerichtete Materialflüsse möglich sind. Die *Reihen-* und *Fließfertigung* sequenzialisieren schließlich, bei gerichtetem Materialfluss, die notwendigen Arbeitsvorgänge eines Erzeugnisses. Während bei der Reihenfertigung ein Überspringen einzelner Fertigungseinrichtungen zum Zwecke der Variantenerzeugung möglich ist und keine Abtaktung besteht, erfolgt bei der Fließfertigung eine streng abgetaktete, variantenlose Teileproduktion. Die *Baustellenmontage* zeichnet sich dadurch aus, dass das Erzeugnis ortsfest komplett hergestellt wird. Bei der *Gruppenmontage* werden dagegen einzelne Montageaufgaben von einer Gruppe von Montageeinrichtungen bzw. von Menschen kooperativ erbracht. Ist eine Teilaufgabe abgeschlossen, bewegen sich entweder die Akteure oder die Erzeugnisse weiter. Die *Reihen-* und *Fließmontage* erfolgen analog zur Reihen- bzw. Fließfertigung, also getaktet oder ungetaktet in entsprechender Reihenfolge der Arbeitsvorgänge.

Sowohl für Fertigung als auch Montage gilt, dass bei Reihen- oder Fließfertigung die Flexibilität tendenziell sinkt, dafür jedoch die Ausbringung steigt. Dies gilt insbesondere bei Abtaktung der Arbeitsstationen. Allerdings erhöht sich durch die sequentielle Anordnung auch die Störanfälligkeit, da im Falle des Ausfalls einer Montageeinrichtung ebenfalls alle anderen Einrichtungen betroffen sind [91].

Weiterführende Strukturierungskonzepte, die auf dem Reihenansatz basieren, sind Linien mit Nebenschlüssen und Ringstrukturen. Nebenschlüsse werden über Abzweigungen an Linien gekoppelt und dienen primär zur Verlagerung produkt- und variantenspezifischer Produktionseinrichtungen in Nebenlinien, so dass sich keine oder nur noch für alle Erzeugnisse gleichermaßen erforderliche Einrichtungen in der Hauptlinie befinden. Ringstrukturen erweitern diesen Ansatz. Sie dienen meist zur aufwandsarmen Rückführung von Transferhilfsmitteln, als Puffer im Falle temporär besetzter Nebenschlüsse und als „Schnellstraße“ zum direkten Transfer von Werkstücken zu benötigten Arbeitsstationen. Linien und Ringe können beliebig zu Strukturen höherer Komplexität kombiniert werden, wobei Koppelpunkte entweder Verzweigungen oder Transfereinrichtungen zum Umschlagen sind.

Insbesondere bei Montage aufwändiger Erzeugnisse, bei der vorgelagert erzeugte Teile in Baugruppen höherer Ordnung verbaut werden, finden sich derartig komplexe Strukturen. Sie ordnen sich dabei zu Baumstrukturen, wobei Zweige einzelnen Linien entsprechen. Häufig werden bei der Anbindung mehrere Linien durch Puffer entkoppelt, da eine passgenaue Abtaktung bei komplexen Strukturen unmöglich ist. Eine weitere Möglichkeit besteht in der räumlichen Trennung mehrerer Linien, die dann mit Hilfe übergeordneter Materialflüsse gekoppelt werden. Dabei kommen häufig ortsvariable Transporteinrichtungen zum Einsatz, die dann zumeist auch den Transfer anderer Stoffe übernehmen.

2.2 Softwaresysteme in der Produktion

Zur Durchführung von Produktionsaufgaben gelangen in rechnergestützten Produktionsanlagen vielfältige Softwaresysteme zum Einsatz. Jedes System ordnet sich nach seiner primären Funktion in einen der vier Bereiche *Steuerung*, *Informationsverwaltung*, *Informationserzeugung* und *Informationsverarbeitung* ein, wobei aber auch mehrere Bereiche abgedeckt werden können. Die zur Ausfüh-

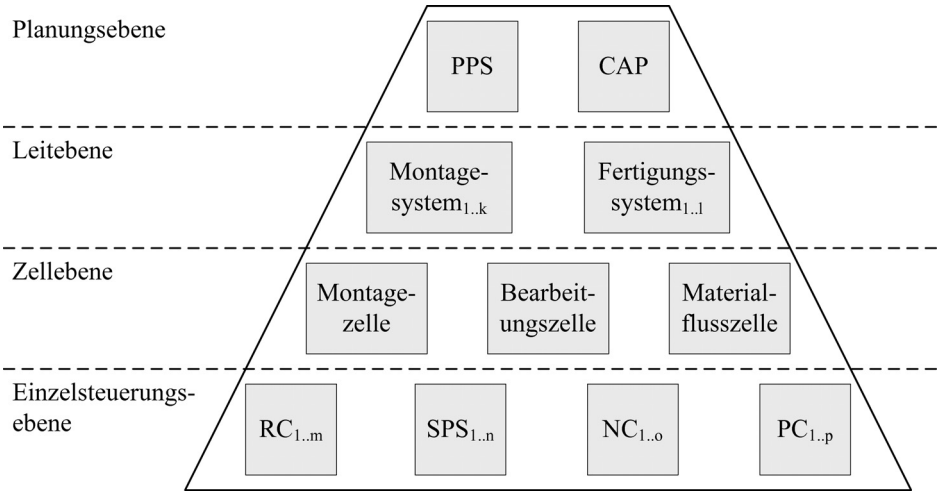


Abbildung 2.1: Hierarchieebenen einer Produktionsanlage nach [79]. Unmittelbare maschinennahe Steuerungen befinden sich auf der untersten Ebene. CAP = Rechnergestützte Planung; NC_o = Numerische Steuerung; PC_p = PC-basierte Steuerung; PPS = Produktionsplanung und Steuerung; RC_m = Robotersteuerung; SPS_n = Speicherprogrammierbare Steuerung.

rung notwendige Hardware und Kommunikationstechnik wird dabei explizit nicht als Bestandteil der Softwaresysteme erachtet, sondern getrennt betrachtet (Kap. 2.3.1).

2.2.1 Steuerungen

Im Rahmen der vorliegenden Arbeit werden als *Steuerungen* alle Softwaresysteme zum Zwecke des direkten oder indirekten *Steuerns*, *Regelns* und zeitlichen *Koordinierens*, von sowohl technischen Einrichtungen als auch Menschen, zusammengefasst, die der kontrollierten Ausführung produktionstechnischer bzw. produktionslogistischer Prozesse dienen.

Nach [79] befinden sich Steuerungen auf allen Ebenen einer Produktionsanlage, deren Hierarchie als Pyramide in Abbildung 2.1 dargestellt ist. Steuerungen können nach „maschinennah“ (unmittelbar) und „maschinenfern“ (mittelbar) unterschieden werden. Maschinennahe Steuerungen bilden die unterste Hierarchieebene, maschinenferne alle anderen Ebenen darüber.

Nach [62] lässt sich maschinennahe Steuerungssoftware bei Fertigungseinrichtungen ebenfalls in einem Ebenenmodell anordnen (Abb. 2.2), dass dem funktionellen Aufbau von Maschinen folgt. Demnach erbringen Maschinen ihre Funktionalität durch hierarchische Aggregation von Elementarfunktionen, die in Abhängigkeit der Komplexität einer Maschine zu Funktionsgruppen zusammengefasst werden. Diese bilden Zwischenebenen im Ebenenmodell. Nach unten schließen Sensoren und Aktoren das Ebenenmodell ab.

Auf Ebene der Maschinen erfolgt die Koordination der in einer Maschine vorhandenen Funktionsgruppen und Elementarfunktionen. Im Gegensatz dazu erfolgt auf Zellebene die ereignisdiskrete oder zeitkontinuierliche Koordination mehrerer Maschinen.

Ab der Zellebene aufwärts erfolgt in einer Produktionsanlage maschinenfern die logistische Kontrolle der Stoffflüsse. Dazu wird im Rahmen dieser Arbeit Auftragsverwaltung, Auftragseinlastung und das

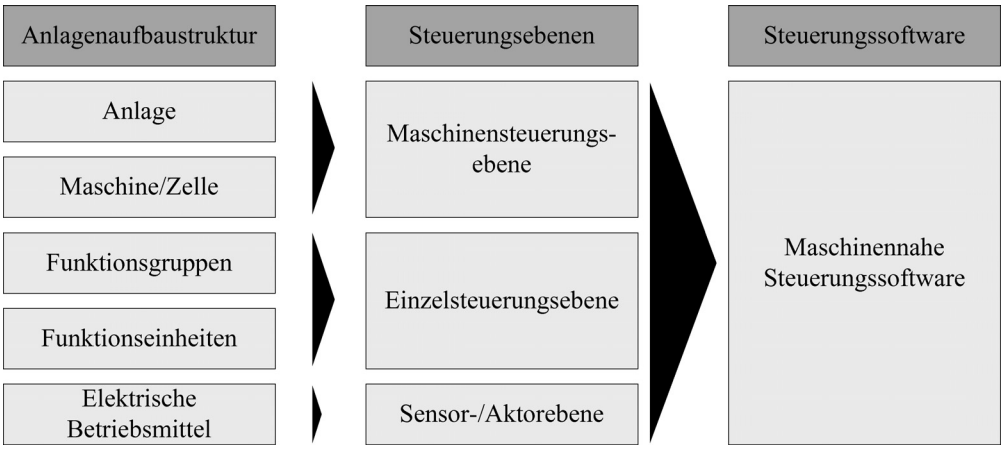


Abbildung 2.2: Ebenenmodell für maschinennahe Steuerungen bei Fertigungseinrichtungen nach [62].

Routing von Stoffen auf den Ebenen Produktionsanlage, Produktionslinien und -inseln sowie Zellen verstanden.

2.2.2 Informationsverwaltung

Jegliche Softwaresysteme innerhalb einer Produktionsanlage, die zur Bereitstellung und Verwaltung von Informationen dienen und direkt oder indirekt zur Steuerung der Produktionsabläufe notwendig sind, werden im Rahmen dieser Arbeit als *Informationsverwaltungssysteme* aufgefasst. Sie stehen mit einer oder mehreren Informationsquellen bzw. -senken in Verbindung, so dass Informationsflüsse entstehen. Die Verwaltung der Informationen kann dabei zentral oder dezentral realisiert sein. Zu den möglichen Systemen zählen *Auftrags-, Arbeitsplan-, Rezept-, Material-, Hilfsmittel- und Betriebsdatenverwaltung* sowie Systeme zur *Qualitätssicherung* und *Produktverfolgung*.

Alle zur Produktion vorgesehenen Aufträge werden zunächst in der *Auftragsverwaltung* gesammelt und sukzessive zur Herstellung freigegeben, worauf sie die Produktionsanlage durchlaufen. Der Weg ist von der Reihenfolge der erforderlichen Arbeitsschritte bestimmt, die in erzeugnisspezifischen Arbeitsplänen festgehalten und von der *Arbeitsplanverwaltung* zur Verfügung gestellt werden. Um einzelne Arbeitsschritte durchführen zu können, benötigen Produktionsprozesse entsprechende Arbeitsanweisungen (= Rezepte) für Werker oder technische Produktionseinrichtungen. Diese werden von der *Rezeptverwaltung* bereitgestellt. Die zur Durchführung benötigten Teile, Materialien oder Hilfsmittel organisieren oben genannte Verwaltungen. Die *Betriebsdatenverwaltung* schließlich sammelt alle organisatorischen und technischen Betriebsdaten zu Auftragszuständen, Personal, Maschinenzuständen und Prozessdurchführung [117]. Sie stellt damit den Erinnerungsspeicher der Produktion dar und dient zur Informationsbereitstellung für höhere Steuerungsaufgaben, wie *Qualitätssicherung* oder *Produktionsplanung und -steuerung* (PPS), einerseits und weiteren Informationsverarbeitungssystemen, wie der *Produktverfolgung*, andererseits.

2.2.3 Informationserzeugung und -verarbeitung

Elektronisch verarbeitbare Informationen (Daten) entstehen an unterschiedlichen Stellen einer Produktionsanlage. Sie werden entweder durch Sensoren erzeugt, aus der Produktionsplanung und -steuerung eingeschleust, entstehen durch die Verarbeitung anderer Informationen, werden von Softwaresystemen an andere übermittelt oder von Menschen eingegeben. Als *Softwareapplikationen* werden im Rahmen dieser Arbeit alle Softwaresysteme aufgefasst, die entweder zur automatischen Datenverarbeitung und -erzeugung dienen und dabei keine Steuerungen sind, oder Schnittstellen zur menscherfassbaren Informationsdarstellung und manuellen Datenerzeugung realisieren.

Die Interaktion mit dem Mensch kann prinzipiell auf allen Anlagenebenen stattfinden, von Leitstandsebene bis hinab zur Sensor-/Aktor-Ebene. Die Aufgaben der Softwareapplikationen sind hierbei Informationsanzeige, Parametrierung, Steuerung oder Regelung.

2.2.4 Interaktion von Softwaresystemen

Softwaresysteme stehen in *direkter* und *indirekter* Verbindung zueinander. Eine direkte Verbindung besteht bei einer Software-Software-Kopplung, wobei die Systeme lokal auf einem oder entfernt auf unterschiedlichen Rechnern ausgeführt werden können. Im zweiten Fall gelangen zur Realisierung der Interaktion Kommunikationstechnik und -protokolle zum Einsatz. Eine indirekte Interaktion erfolgt durch eine zwischengeschaltete Aktor-Sensor-Kopplung. Hierbei bewirkt ein Aktor die „Reizung“ eines Sensors, wobei dies mit zeitlicher Verzögerung geschehen kann. So stehen letztlich alle Softwaresysteme einer Produktionsanlage miteinander in Verbindung, da Aktionen eines Softwaresystems über mehrere zwischengeschaltete Systeme auf mindestens ein anderes wirken. Alle dazwischen liegenden Systeme werden dabei ebenfalls beeinflusst.

2.3 Steuerungstechnik

Die *Steuerungstechnik* repräsentiert als Überbegriff im Rahmen dieser Arbeit die zur Ausführung jeglicher Softwaresysteme erforderliche *Rechen-* und *Kommunikationstechnik*.

2.3.1 Rechentechnik

Zur Ausführung von Softwaresystemen werden *SPSen, PCs* und *Mikrocontroller* eingesetzt. Deren Implementierung erfolgt dabei unter Zuhilfenahme spezifischer, von der Rechentechnik abhängiger, Programmiersprachen. Obwohl die SPS (und auch SPS-Einsteckkarten bzw. Soft-SPS-Lösungen für PCs) primär für Steuerung und Regelung auf Linien-, Zellen- und Geräteebene konzipiert ist, besteht die Möglichkeit zur Benutzerinteraktion, die entweder unter Zuhilfenahme von Sensorik (Taster, Drehknöpfe, etc.) oder durch Ankopplung graphischer Anzeigeeinheiten erfolgt. Nicht geeignet sind sie dagegen für Informationsverwaltung und -verarbeitung. Im Gegensatz dazu eignet sich der PC aufgrund seiner offenen Architektur zur Entwicklung von Softwaresystemen für alle Bereiche, also zum Steuern/Regeln, Informationsverwaltung und -verarbeitung sowie Benutzerinteraktion auf allen

Ebenen einer Produktionsanlage. Der Mikrocontroller wird vorwiegend zum Steuern und Regeln auf den unteren Ebenen eingesetzt. Über koppelbare Anzeigeeinheiten und mit Hilfe von Sensorik gelingt ebenfalls Benutzerinteraktion. Mikrocontroller werden häufig für Speziallösungen eingesetzt, wobei in der Regel fallspezifisch eigens entwickelte elektronische Baugruppen eingesetzt werden. Sie befinden sich auch auf Steuer-/Reglerkarten für PCs oder Peripheriebausteinen für SPSen.

2.3.2 Kommunikationstechnik

Als Grundlage zum Datenaustausch finden speziell für die Produktion entwickelte *Feldbusse* sowie universellere Techniken Anwendung. Hier ist allen voran *Ethernet* zu nennen, aber auch serielle Punkt-zu-Punkt-Kommunikation wie RS232 und RS485 oder serielle Busse, wie USB. Feldbusse, wie bspw. *AS-Interface, ControlNet, CANopen, DeviceNet, Interbus, Profibus, SafetyBUS* oder *SERCOS* werden vor allen Dingen in der Feld- und Prozessebene der klassischen Kommunikationspyramide eingesetzt und dienen zur Interaktion verteilter Steuerungen untereinander sowie mit peripherer Aktorik und Sensorik. Häufig werden sie gemischt verwendet, da die einzelnen Feldbusse für unterschiedliche Einsatzschwerpunkte konzipiert wurden [43]. Zunehmend erhalten Feldbusse Konkurrenz von industriellen Ethernet-Lösungen, wie *EtherCAT, Ethernet-Powerlink, SERCOS III, ProfiNet, Varan oder SafetyNET*. Diese bieten, gegenüber den klassischen Feldbussen, neben einer häufig höheren Leistungsfähigkeit und größerer Universalität, den Vorteil einer direkten Kopplungsfähigkeit an höhere Ebenen der Fabrik, da deren Kommunikationsnetze ebenfalls auf Ethernet basieren.

Jede Kommunikationstechnik verfügt über eigene, proprietäre Protokolle. Allerdings haben die meisten Feldbussanbieter mittlerweile TCP/IP integriert, so dass darauf aufbauende Anwendungsprotokolle wie UDP, TCP, HTTP, FTP, SSH [102] etc. genutzt werden können. Dies vereinfacht einerseits die Realisierung einer Interoperabilität zwischen Geräten – zumindest bei Nichtechtzeitanforderungen – und andererseits die Ankopplung der Prozessebene an höhere Ebenen der Kommunikationspyramide erheblich.

2.4 Produktionsanlagen im Wandel

Eine Vielzahl von Wandlungstreibern können auf einen Produktionsbetrieb einwirken und eine Anpassung der Produktionsanlagen an veränderte Situationen erzwingen. Die Reaktion reicht auf technischer Seite von der Umparametrierung bis hin zum Umbau der Anlage. Um diese Schritte effizient und wirtschaftlich durchführen zu können, ist die Ausstattung einer Produktionsanlage mit Wandlungspotenzial erforderlich. Alle Konzepte zur entsprechenden Planung und Gestaltung basieren auf dem Prinzip der Modularisierung durch Standardisierung von Funktionen und Schnittstellen. Es ist nicht bekannt, ob bisherige Ansätze explizit Steuerungstechnik mit einbeziehen, weshalb an dieser Stelle entsprechende Defizite angenommen werden.

2.4.1 Wandlungstreiber

Nach [115] lassen sich *externe* und *interne* Wandlungstreiber unterscheiden. Die externen Treiber gliedern sich in *technologie-, markt-* und *umweltinduzierte*, während sich die internen in *reaktive* und *proaktive* Treiber unterteilen.

Im technologischen Bereich führt der Einsatz neuer Verfahren und Prozesse, zur Steigerung der Effizienz oder zur Verbesserung der Produkteigenschaften, zu Veränderungen an Produktionsanlagen, da ein Austausch bzw. eine Anpassung von Produktionseinrichtungen erforderlich wird. Der Einsatz neuer Werkstoffe bewirkt ähnliches, wenn Prozesse modifiziert oder ersetzt werden müssen bzw. neu hinzukommen.

Seitens der Märkte führen steigende Anforderungen an Funktionalität, Qualität, Design und Kostenreduzierung zu stetigen Produktverbesserungen. Die Individualisierung von Kundenwünschen, aber auch regionale Unterschiede in den globalen Märkten, führen zu höherer Produktdiversifikation mit kleineren Losgrößen und steigenden Produktmixen, die sich durch schwankende Nachfragen und Materialverfügbarkeit stetig verändern. Hieraus resultieren kürzere Produktlebenszyklen sowie Produktionslebenszyklen – zumindest unter der Sichtweise, dass aus Veränderungen an einem Produkt oder einer Anlage ein neues Produkt oder eine neue Anlage entsteht. Die Verkürzung von Lieferzeiten und die Verbesserung der Liefertreue führen neben einer strafferen Organisation der Produktionsabläufe und besser Qualitätssicherung unter Umständen zu Modifikationen von Produktionsanlagen, etwa durch Veränderungen von Automatisierungsgraden oder der Kapazität von Produktionseinrichtungen.

Umwelteinflüsse, wie Umweltschutzauflagen oder Sicherheitsbestimmungen für Arbeitskräfte – letztlich auch die Verfügbarkeit und Qualifikation der Arbeitskräfte selbst – haben Auswirkungen auf die Gestaltung einer Produktionsanlage und können bei Änderungen Anlagenmodifikation erzwingen.

Reaktive Wandlungstreiber resultieren aus Schwachstellen oder Verbesserungspotenzialen in den Produktionsabläufen bzw. -prozessen, die erst während der Betriebsphase entstehen oder erkannt werden. Sie betreffen die Produktqualität, Lieferzeiten, Liefertreue, Bestände, Prozesssicherheit, Produktionskosten und die Übersichtlichkeit der Materialflüsse.

Zu den proaktiven Wandlungstreibern zählen im Wesentlichen firmenstrategische Entscheidungen zur Senkung der Kosten, zur Erhöhung der Marktanteile oder zur Erschließung neuer Märkte. Reaktive wie proaktive Treiber können zu organisatorischen und technischen Veränderungen der Produktionsanlage führen.

Alle Wandlungstreiber können Auswirkungen in grundsätzlich sechs Wirkdimensionen induzieren, wie Abbildung 2.3 verdeutlicht. Diese sind:

- Qualitative und quantitative Änderungen in Produktmixen
- Änderungen im Produktspektrum
- Änderungen im Funktionsspektrum
- Änderungen von Kapazitäten
- Änderungen im Automatisierungsgrad

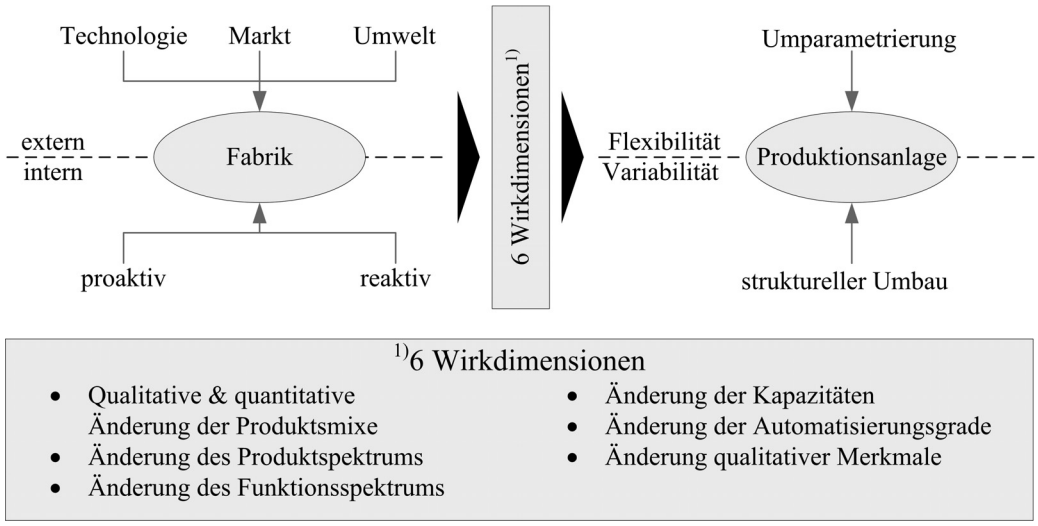


Abbildung 2.3: Wandlungstreiber induzieren Veränderungen in den Anforderungen an die Produktionsanlage, die unter Nutzung von Flexibilität oder Variabilität an neue Situationen angepasst wird. Teilweise nach [115].

- Änderungen qualitativer Merkmale, z.B. zum Schutz der Umwelt oder Mitarbeitern

Um den Druck der Wandlungstreiber aufzufangen, sind zudem organisatorische Anpassungen der Produktionsabläufe oder strukturelle Anpassungen an der Produktionsanlage erforderlich.

2.4.2 Wandlungsfähigkeit von Produktionsanlagen

Nach [115] und in Anlehnung an [114], beschreibt Wandlungsfähigkeit „das Vermögen einer Fabrik, ausgehend von internen und externen Auslösern, aktiv ihren Aufbau auf allen Ebenen“ „bei geringen Kosten, Sach- und Personalaufwand“ „verändern zu können“. Zwei Aspekte kennzeichnen dabei die Wandlungsfähigkeit, die *Entwicklungsfähigkeit* und die *Anpassungsfähigkeit* [90, 91, 93, 111].

Die Entwicklungsfähigkeit einer Fabrik ist durch ihre *Agilität, Vitalität* und *Reaktionsschnelligkeit* geprägt. Ersteres bedeutet den Grad der Eigenständigkeit, Veränderungen selbstständig anzustoßen und durchzuführen, während Zweiteres die Sensibilität für die Erkennung von Änderungsbedarfen bedeutet. Die Entwicklungsfähigkeit ist auf organisatorischer Ebene angesiedelt und wird daher nicht weiter vertieft.

Die *Anpassungsfähigkeit* betrifft die Produktionsebene und wird im Wesentlichen durch die beiden Faktoren *Flexibilität* und *Variabilität* bestimmt.

Flexibilität beschreibt das einer Produktionsanlage innewohnende Potenzial zur aufwandsarmen Anpassung an sich ändernde Anforderungen, ohne Umbauten vorzunehmen. Nach [91] können die drei verschiedenen Dimensionen *funktionale, strukturelle* und *kapazitive Flexibilität* unterschieden werden.

Mit einer hohen *funktionalen Flexibilität* ist in einer Produktionsanlage ein Übermaß an Prozessen implementiert, mit denen ein Produkt- und Variantenspektrum abgedeckt wird, das nicht gleichzeitig

erzeugt wird. Stattdessen ändern sich die zu produzierenden Produktmixe qualitativ, entsprechend der Nachfrage.

Strukturelle Flexibilität bezieht sich sowohl auf Strukturen der Stoffflüsse als auch auf die (teil-)redundante Verfügbarkeit von Prozessen. Im Falle von Systemstörungen, Wartungsarbeiten oder Pausen können alternative Stoffflusspfade und Produktionseinrichtungen genutzt werden, so dass die Leistungsfähigkeit der Produktionsanlage erhalten bleibt.

Kapazitive Flexibilität bedeutet schließlich, die Produktionsanlage mit Leistungsreserven auszustatten, so dass ein Kapazitätsband entsteht, innerhalb dessen operiert wird. So können in einem gewissen Rahmen quantitative Änderungen von Produktmixen, also schwankende Mengennachfragen verschiedener Produkte und Varianten, abgefangen werden.

Die Entscheidung über den Grad an Flexibilität einer Produktionsanlage hängt von betriebswirtschaftlichen Gesichtspunkten ab, da sowohl ein Übermaß an Prozessen als auch strukturelle und Kapazitätsreserven einen geringeren Nutzungsgrad von Produktionseinrichtungen zur Folge haben. Andererseits kann ein Mangel an Flexibilität schnell zu niedrigeren Anlagenleistungen führen. Flexibilität ist im Sinne einer schnellen Reaktionsfähigkeit auf Änderungsanforderungen mit der Umkehrbarkeit zur Ausgangskonfiguration elastisch.

Im Gegensatz zur Flexibilität steht *Variabilität* für den strukturellen Umbau der Produktionsanlage. Hierbei erfolgt die Anpassung an veränderte Randbedingungen durch Entfernen, Austauschen oder Hinzufügen neuer Teile. Für eine aufwandsarme und effiziente Durchführung ist eine konsequente Modularisierung der Produktionsanlage auf verschiedenen Ebenen unabdingbar. Dies setzt neben einer Abgrenzung von Funktionalitäten in Module auch den Entwurf konsistenter Schnittstellen voraus [41, 42]. Variabilität kann so gleichgesetzt werden mit *Modularisierung durch Standardisierung von Funktionen und Schnittstellen*.

2.4.3 Lösungsansätze in Technik und Forschung

Die Analyse der in der Literatur benannten Konzepte zur Befähigung von Produktionsanlagen zum technisch effizienten Wandel zeigt eine Präferenz für eine modulare Gestaltung auf mehreren Hierarchieebenen, wobei Module Elemente eines Baukastensystems sind. Wandlung wird durch Entfernen, Austauschen oder Hinzufügen von Modulen erreicht. Als wesentliche Elementeigenschaften gelten dabei *Universalität, Autarkie, Modularität, Kompatibilität, Mobilität* und *Skalierbarkeit* [41, 68, 115].

Universalität bedeutet, dass Elemente eines Baukastens in unterschiedlichen Kontexten nutzbar sind. In Bezug auf andere Elemente weisen sie eine möglichst hohe Unabhängigkeit auf, um Universalität zu wahren. Sie verfügen als Module über eine abgeschlossene Funktionalität sowie über Schnittstellen zur Kopplung mit anderen Elementen, wobei ein hohes Maß an Standardisierung der Schnittstellen Kompatibilität ermöglicht. Da ein Wandel auf technischer Seite mit einem Umbau verbunden ist, den es aufwandsarm umzusetzen gilt, sind Elemente handhabbar als kompakte, bewegliche bzw. einfach bewegbare Einheiten zu gestalten. Neben diesen Aspekten wird durch eine skalierbare Auslegung erreicht, dass kleine bis große Systeme realisiert werden können, die – im Sinne einer Atmung – über die Zeit in ihrer Größe veränderbar sind. [41, 68, 115]

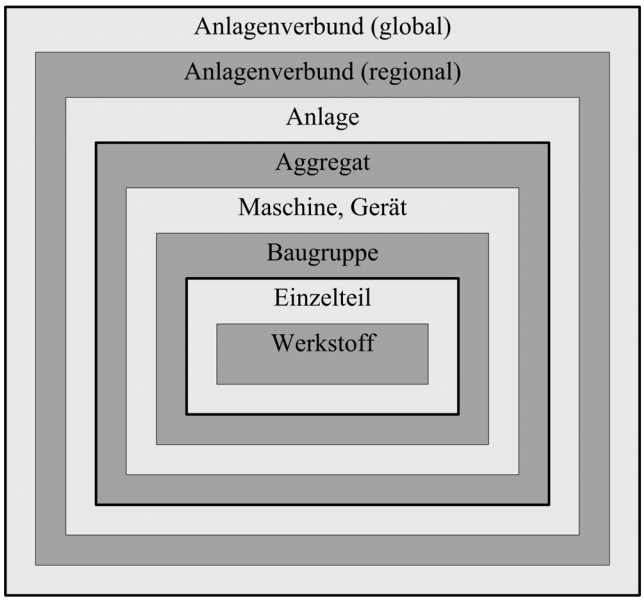


Abbildung 2.4: Ropohl'sche 8-Ebenen-Hierarchie [86].

Produktionsanlagen verfügen über einen hierarchischen Aufbau, wobei in der Literatur verschiedene Anzahlen an Hierarchieebenen zu finden sind. Geläufig scheinen sieben bis neun Ebenen zu sein, wobei an dieser Stelle Bezug auf die Ropohl'sche 8-Ebenen-Hierarchie genommen wird [86]. Wesentlich ist, dass Lösungskonzepte und kommerzielle Lösungen für unterschiedliche Ebenen existieren, jedoch keiner der bekannten Ansätze alle Ebenen umfasst.

Im Bereich der modularen Montagesysteme existieren Lösungen auf den Ebenen der Aggregate, Maschinen und Baugruppen (Bezeichnung der Ebenen nach [86]), wie eine Marktstudie aus dem Jahr 2000 zeigt, in der 80 europäische Hersteller aus dem Bereich der Montagesysteme und -komponenten befragt wurden [28]. Modulare Montagesysteme lassen sich in *Modulare Baukastensysteme, Modulsysteme* und *Modulare Komplettsysteme* kategorisieren [96]. Ein Beispiel für ein Modulsystem mit teilweise automatisierter Berücksichtigung von Materialflüssen ist das MAMOS-Montagesystem, das über eine zentrale Simulatorankopplung die Struktur und Strukturänderungen eines Produktionssystems erkennt und den Materialfluss automatisch adaptiert [96].

Ein Vertreter für wandelbare Geräte stellt der PowerCube von Amtec Robotics dar. Hier können Roboter aufgabenspezifisch aus einzelnen Grundmodulen zusammengesetzt werden, wobei Komponenten für Schwenken, Antrieb, Verfahren und Greifen zur Verfügung stehen. [1]

Standard im Bereich der Dreh- und Fräsmaschinen sowie bei Robotern sind Werkzeugwechselsysteme zur effizienten Maschinenrüstung. Auch hierbei besteht ein modularer Aufbau dieser Systeme, die dadurch wandlungsfähig sind. Mit einem an der Universität Erlangen-Nürnberg entwickelten flexiblen Drei-Finger-Greifer existiert ein Beispiel für ein Werkzeug mit integrierter Intelligenz, das aufwandsarm an einen Roboter gekoppelt werden und mit diesem interaktiv Handhabungsaufgaben bewältigen kann [26].

Mit dem Konzept PLUG+PRODUCE, das neben Produktionssystemen auch auf Gebäudesysteme fokussiert, liegen theoretische Grundlagen für die Schaffung wandlungsfähiger Produktionsanlagen vor.

Es beschränkt sich auf den Aufbau von Fabriken aus Fabrikmodulen und -komponenten und definiert daher für die eigentlichen Produktionssysteme Elemente auf den Ebenen der Aggregate und Maschinen. Das Konzept unterscheidet zwei wesentliche Fabrikstrukturtypen. Typ 1 stattet eine Fabrik mit dem Potenzial aus, effizient an neue Situationen angepasst werden zu können. Typ 2 ermöglicht den schnellen Umzug der gesamten Produktion an einen neuen Standort. [41, 42]

Die Ausgestaltung von Produktionseinrichtungen und ihre Ausstattung mit Mobilität als Mittel zur schnellen Anpassbarkeit von Produktionsanlagen, aber auch die Mobilität ganzer Produktionssysteme sind Themen verschiedener Forschungsarbeiten. [46] betrachtet Mobilität von Produktionseinrichtungen als Mittel für die flexible Layoutgestaltung von Fabriken. [54] entwickelte eine Methodik zur Gestaltung und Bewertung mobiler Produktionssysteme und [94] eine Bewertungssystematik zur Gestaltung von struktur- und betriebsvariablen Produktionssystemen.

Auf Seite der Planung wandlungsfähiger Fabriken befasste sich [21] mit der Planung und dem Betrieb wandlungsfähiger Logistiksysteme in der variantenreichen Serienproduktion, entwarf [68] eine Systematik der Wandlungsfähigkeit in der Fabrikplanung und behandelte [3] allgemein die Fabrikplanung im turbulenten Umfeld. [83] beleuchtete die Ausstattung von Simulationsmodellen mit Wandlungspotenzial für die Ablaufsimulation.

Im Bereich wandlungsfähiger Steuerungstechnik und Softwaresysteme sind mit Ausnahme des Konzepts der *rekonfigurierbaren Produktionssysteme (RMS – Reconfigurable Manufacturing Systems)* keine Entwicklungen bekannt. Dieses berücksichtigt in seinem Ansatz explizit Steuerungssoftware und verdeutlicht die Notwendigkeit rekonfigurierbarer Software. Es sieht dazu Softwaremodule vor, die anwendungsspezifisch verknüpft werden und auf einer allgemeinen Hardwareplattform laufen. Umbauten am physischen System gehen mit Umbaumaßnahmen der Software einher, so dass Physis und Steuerung eine passende Einheit bilden. Mehrere Steuerungen können über Rechnergrenzen hinweg untereinander kommunizieren, wozu eine Kommunikations-Middleware vorgesehen ist. [50]

Andere Arbeiten konzentrieren sich auf die Konfektionierung von Steuerungssoftware in jeweils verschiedenen Ebenen der Produktionsanlage zur Realisierung kurzer Entwicklungszeiten. Wesentlicher Ansatzpunkt ist hierbei die Wiederverwendung bereits entwickelter Softwarekomponenten. So entwarf [9] ein adaptierbares Leitsteuerungssystem für flexible Produktionssysteme, befasst sich [35] mit der durchgängigen Programmierung von Fertigungszellen und [19] mit dem dynamischen Konfigurieren von Steuerungssoftware für offene Systeme. [62] behandelt Softwaretechnik für maschinennahe Steuerungsfunktionen bei Fertigungseinrichtungen und [100] konzipierte modulare Systemplattformen für offene Steuerungssysteme. Alle genannten Arbeiten zur Softwareentwicklung konzentrieren sich auf Steuerungen von Fertigungseinrichtungen. Softwaresysteme auf höheren Ebenen betrachten sie nicht.

Im Bereich logistischer Steuerungen sind dezentrale Ansätze bekannt, die eine Selbstorganisation der Abläufe zum Ziel haben. [78] konzipierte ein Multiagentensystem für die Ablaufsteuerung in der flexibel automatisierten Fertigung. [16] stellen eine agentenbasierte Steuerungsplattform für holonische Produktionssysteme und [57] mit *ADACOR* eine Automatisierungs- und Steuerungsarchitektur für kooperierende Systeme vor. [59] zeigt eine dezentrale Steuerung für Materialflusssysteme am Beispiel von Stückgutförder- und sortieranlagen. Alle genannten Arbeiten vernachlässigen Softwaresysteme auf niedrigeren Ebenen.

2.4.4 Defizite und Ausleitung von Handlungsbedarfen

Die in der Literatur auffindbaren Konzepte zur technischen Wandlungsfähigkeit zeichnen sich überwiegend durch einen einseitigen Schwerpunkt auf physische Modularisierung aus und konzentrieren sich auf den effizienten Umbau in den jeweils favorisierten Hierarchieebenen. Ein ganzheitlicher, homogener Ansatz ist bisher nicht bekannt. Mit einer Ausnahme benennt und berücksichtigt keines der genannten Konzepte die geeignete Wandlungsfähigkeit der Steuerungstechnik und der Softwaresysteme. Dies erscheint insofern als ein Defizit, als dass jede Veränderung auf physischer Seite häufig eine Anpassung der Steuerungstechnik und immer eine Anpassung der Softwaresysteme erfordert. Bisher erfolgt dies entweder in voraussehender Weise durch den Versuch, alle vorhersehbaren Möglichkeiten a priori zu ermitteln und zu implementieren oder a posteriori, indem die Systeme nachträglich angepasst werden. Beide Vorgehensweisen sind suboptimal, da einerseits zukünftige Anforderungen vorher nie vollständig erfassbar sind und andererseits ein Anpassen ohne ein entsprechendes Softwarekonzept hohe Entwicklungsaufwendungen verursacht und sich dadurch die Wiederanlaufzeiten erhöhen.

Zwar sieht das Konzept der rekonfigurierbaren Produktionssysteme [50] modulare Steuerungen und Softwaresysteme vor, jedoch bleiben eine analytische Vorgehensweise, die Konzeption einer entsprechenden Softwarearchitektur und eine konkrete Umsetzung im Unklaren.

Deshalb erfolgt im Weiteren zunächst eine systemtheoretische Analyse, die die Problemstellung der wandelbaren Produktionsanlagen ganzheitlich aufgreift und insbesondere die theoretischen Zusammenhänge zwischen physischen Geräten und ihren Steuerungen aufzeigt. Darüber hinaus erfolgt die Entwicklung eines Systemkonzepts für wandelbare Produktionsanlagen. Ausgehend von diesem Systemkonzept wird dann das Bild einer wandelbaren Systemlandschaft gezeichnet, das auf einem eigenen Hierarchiemodell fußt. Da modulare physische Systeme hinreichend bekannt sind, fokussiert die Landschaft auf wandelbare, rechnerübergreifende Softwaresysteme und Steuerungstechnik, die für den Aufbau wandelbarer Produktionsanlagen erforderlich sind. Zur Realisierung kooperativer, rechnerübergreifender Softwaresysteme wird anschließend das Konzept einer auf Wandelbarkeit ausgerichteten Softwarearchitektur vorgestellt. Produktionsanlagen mit einheitlichen Softwaresystemen vorausgesetzt, werden Verfahren zur logistischen Selbstkonfiguration und Selbstorganisation vorgestellt, die die Machbarkeit von selbstadaptierenden Produktionssystemen im Wandel belegen. Es wird gezeigt, wie sich Materialflüsse von Produktionslinien und -inseln sowie von komplexen Produktionsanlagen mit minimalen Aufwendungen weitgehend selbstständig an neue Situationen anpassen können. Des Weiteren werden selbstadaptierende generische Informationsdienste und Softwareapplikationen vorgestellt, die für die Realisierung wandelbarer Produktionsanlagen unabdingbar sind. Um zu den Konzepten auch den praktischen Nachweis zu erbringen, werden abschließend eine Referenzimplementierung der Softwarearchitektur und die Evaluation der Konzepte vorgestellt.

Kapitel 3

Vorüberlegungen zur Gestaltung wandelbarer Produktionsanlagen

Um Lösungen für die Problemstellung wandelbar gestalteter Produktionsanlagen zu entwickeln, ist ein ganzheitliches, konsistentes theoretisches Systemkonzept erforderlich, das vor allem auch Aspekte von Steuerungen berücksichtigt.

Hierfür wird zunächst ein Überblick über die allgemeine Systemtheorie gegeben, aus der im Anschluss eine steuerungsmotivierte Systemklassifikation abgeleitet wird, in die sich Produktionsanlagen aus Sicht der Steuerungen einordnen lassen. Teil dieser Systemklassifikation ist die Klasse der gesteuerten Systeme, die als übergeordnete Klasse von Produktionsanlagen näher beleuchtet wird. Aus diesen Betrachtungen werden Schlussfolgerungen für die Gestaltung der Wandelbarkeit gezogen, die abschließend zum bereits genannten theoretischen Systemkonzept führen. [29]

3.1 Überblick über die allgemeine Systemtheorie

Auf Grundlage der allgemeinen Systemtheorie lassen sich alle Systeme – somit auch Produktionsanlagen – beschreiben. Zunächst werden die Bausteine *Element, System, Elementesystem*, *Subsystem* und *Teilsystem* in Anlehnung an Ropohl definiert [85, 86]. Im Gegensatz zu dem deskriptiven Charakter der dort genannten Definitionen zielen die hier Vorgestellten auf die Synthese von Systemen ab.

3.1.1 Element

Ein *Element* E ist die kleinste Einheit eines Systems und wird als unzerlegbar, also atomar angesehen. Dabei ist nicht die faktische sondern die kontextabhängige Unzerlegbarkeit von Bedeutung. So ist bspw. die Zerlegbarkeit eines Elektromotors im Kontext eines Transfersegments unwichtig. Das Element verfügt über *Schnittstellen*, die durch *Konnektoren* realisiert sind, mit deren Hilfe *Verbindungen* zu anderen Elementen erstellt werden können. Es sind die zwei Gruppen *bidirektional strukturelle* und *unidirektional funktionelle* Schnittstellen unterscheidbar. Erstere dienen dazu, zwei Elemente „physisch“ miteinander zu verbinden, Letztere zur Übertragung physischer und nicht physischer Objekte (Abb. 3.1).

Objekte, die über einen Eingang X' in ein Element gelangen, werden als Eingangsvariable x, Objekte, die ein Element über einen Ausgang Y' verlassen, als Ausgangsvariable y der Elementfunktionalität φ beschrieben. Die Funktionalität eines Elements ergibt sich aus einer Menge von Funktionen, die in ihrer Gesamtheit die Menge der Eingangsvariablen $\{x\}$ auf die Menge der Ausgangsvariablen

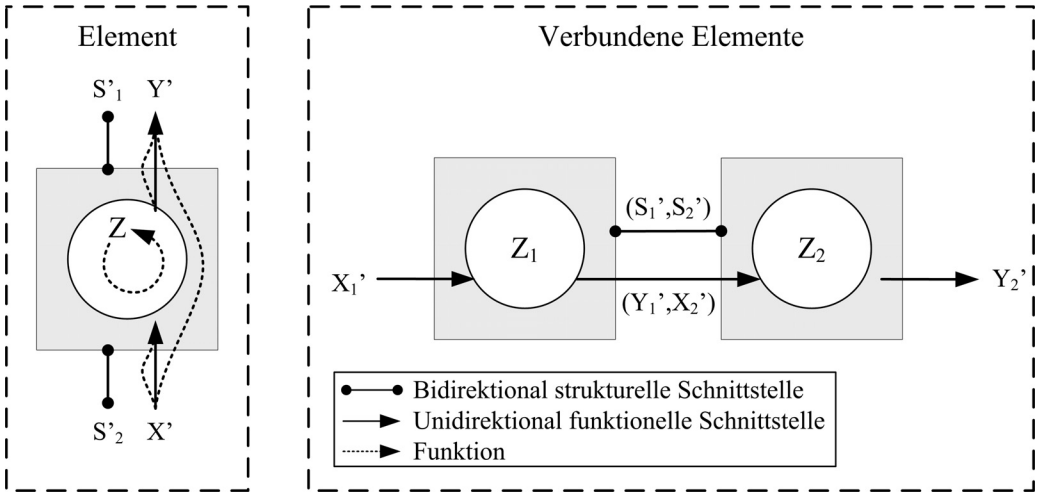


Abbildung 3.1: Links – Element. Rechts – Strukturell und funktionell verknüpfte Elemente. S_l = Strukturschnittstelle; X'_m = Eingang; Y'_n = Ausgang; Z = Zustand.

$\{y\}$ abbilden. In diesem Zusammenhang wird auch vom „Verhalten“ eines Elements gesprochen. Die Abbildung kann direkt oder in Abhängigkeit einer dritten Komponente, dem Zustand Z, erfolgen. Z besteht aus einer Menge von *Zustandsvariablen*. Der Wert einer Zustandsvariable z ist abhängig von den Zuständen einer oder mehrerer Eingangsvariablen bzw. anderen Zustandsvariablen. *Zustandsänderungen* liegen somit – über funktionale Abhängigkeiten – einerseits Wechsel an Eingängen zu Grunde. Andererseits können sich diese auch spontan, ohne äußere Einwirkung, verändern.

Elemente agieren in einem System entweder als *Objektquellen*, die Objekte in ein System einschleusen, als *Objekttransformer*, die den Zustand eines Objekts wandeln, oder als *Objektsenken*, die Objekte aus dem System entfernen.

Formal kann das Element wie folgt beschrieben werden:

Element:
$E = (S, X, Y, Z, \varphi)\ mit\ S \cup X \cup Y \neq \emptyset$

Strukturelle Schnittstellen:
$S = \emptyset \cup \{S'_i\}\ ;\ mit\ i = 1, , I \wedge I \in \mathbb{N}$

Eingänge:
$X = \emptyset \cup \{X'_i\}\ ;\ mit\ i = 1, , I \wedge I \in \mathbb{N}$

Eingang:
$X' = (S_x, x)$

Ausgänge:
$Y = \emptyset \cup \{Y'_i\}\ ;\ mit\ i = 1, , I \wedge I \in \mathbb{N}$

Ausgang:
$Y' = (S_y, y)$

Zustand:
$Z = \emptyset \cup \{Z_i^{'}\}\ ;\ mit\ i = 1, , I \wedge I \in \mathbb{N}$

Funktionalität:
$\varphi = \{\prod(X \cup Z) \rightarrow Z\} \cup \{\prod(Z) \rightarrow Z\} \cup \{\prod(Z) \rightarrow (Y)\}$

3.1.2 System

Ein *System* Σ besteht aus einer Menge von Elementen und Subsystemen, die ihrerseits Systeme darstellen (Abb. 3.2). Die Komponenten eines Systems sind über strukturelle und funktionelle Konnektoren miteinander gekoppelt, wobei zwischen zwei Komponenten mehrere Verbindungen bestehen können. Gemeinsam bilden alle Verbindungen die *Systemstruktur*. Analog zum Element kann ein System das Verhalten $\varphi\varphi$ aufweisen, das sich allerdings aus dem individuellen Verhalten seiner Komponenten und deren funktionellen Kopplungen ergibt. Jedes System kann in die beiden Teile *Strukturteil* ST und *Funktionsteil* FT zerlegt werden [85, 86], wodurch die relevanten Eigenschaften der Systembestandteile getrennt beschreibbar sind.

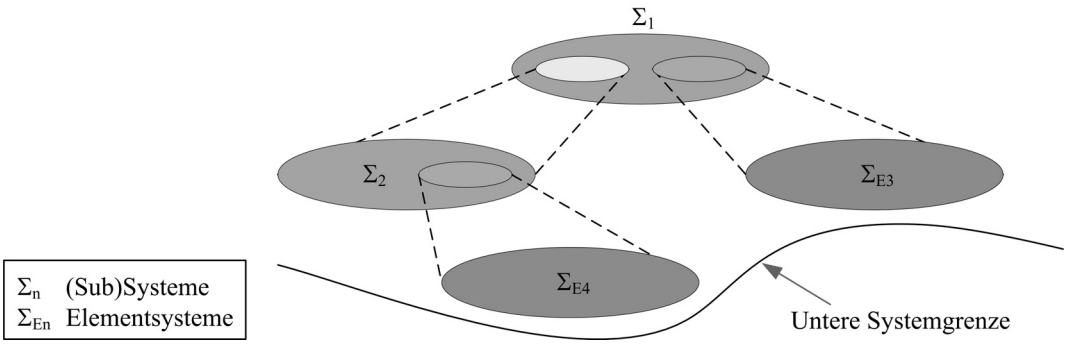


Abbildung 3.2: Systemhierarchie mit Elementesystemen Σ_{En} als untere Grenze.

Systeme können ineinander geschachtelt werden, wodurch die Bildung komplexer Systemhierarchien möglich ist. Für Systeme, die nur aus Elementen bestehen, wird der Begriff *Elementesystem* eingeführt. Diese Systeme bilden stets das untere Ende einer Systemhierarchie. Ein System, das Bestandteil eines übergeordneten Systems ist, wird dessen *Subsystem* genannt. Umgekehrt wird das übergeordnete System als *Supersystem* bezeichnet. Während ein System aus beliebig vielen Subsystemen aufgebaut sein kann, kann es im Allgemeinen nur Teil eines einzigen Supersystems sein. Die Systemhierarchie weist somit immer eine (entartete) Baumstruktur auf. An dieser Stelle vernachlässigte Ausnahmen bilden entsprechend gestaltete Softwaresysteme, da es bei Software prinzipiell möglich ist, ein Subsystem mehreren Supersystemen unterzuordnen.

Formal kann das System wie folgt beschrieben werden:

System:
$\Sigma = (B, FT, ST)$

Bestandteile:
$B = \{E\} \cup \{B^-\}$

Hierarchie:
$\Sigma \subset \Sigma^+ = (B^+, F^+, S^+)$
$\Sigma \supset \Sigma^- = (B^-, F^-, S^-)$
$\Rightarrow \Sigma_H = (.., \Sigma^-, \Sigma, \Sigma^+, ..)$

Funktionsteil:
$FT = (XX, YY, ZZ, \varphi\varphi)$
$XX = \{x \mid x \in X' \wedge X' \in X \wedge X \in E \wedge E \in B\}$
$YY = \{y \mid y \in Y' \wedge Y' \in Y \wedge Y \in E \wedge E \in B\}$
$ZZ = \{Z \mid Z \in E \wedge E \in B\}$
$\varphi\varphi = \{\varphi \mid \varphi \in E \wedge E \in B\}$

Strukturteil:
$ST = (SS, SX, SY, \lambda)$
$SS = \{S \mid S \in E \wedge E \in B\}$
$SX = \{S_x \mid S_x \in E \wedge E \in B\}$
$SY = \{S_y \mid S_y \in E \wedge E \in B\}$
$\lambda = \{(S_1, S_2)\} \cup \{(S_y, S_x)\}\ ; mit$
$S_1 \in E_1, S_2 \in E_2\ und\ S_y \in E_3, S_x \in E_4$

3.1.3 Teilsysteme

Ein *Teilsystem* Σ_T ist ein beliebiger Ausschnitt der Menge der Komponenten eines Systems und seiner Subsysteme sowie deren Kopplungen. Es ist ein System nach obiger Definition. Seine Bestandteile sind nicht auf eine bestimmte Ebene der Systemhierarchie begrenzt. Bausteine unterschiedlicher Ebenen können ein Teilsystem bilden. Da die Elemente eines Systems und seiner Subsysteme Teil des Systems selbst sind, ergibt sich die Menge Ω_{Σ_T} aller potenzieller Teilsysteme als:

$\Omega_{\Sigma_T} = \wp(\bigcup(E_k \in B \in \Sigma))$
$\Sigma_T \in \Omega_{\Sigma_T}$

Zwar ist die willkürliche Bildung von Teilsystemen nicht sinnvoll, jedoch sind Gesamtsysteme häufig erst durch spezifische Teilsysteme, mit aufeinander abgestimmten Komponenten, funktionsfähig. Ein Beispiel hierfür sind logistische Softwarekomponenten, die auf verschiedene Einzelgeräte verteilt sind, jedoch ein gemeinsames Ganzes bilden müssen, um eine Produktion zu ermöglichen.

3.1.4 Schnittstellen und Schnittstellentypen

Schnittstellen übernehmen wichtige Aufgaben für Elemente und Systeme, da sie einen wesentlichen Beitrag zur Realisierung der Gesamtfunktionalität eines Elements oder Systems leisten.

Eine wesentliche Eigenschaft von Schnittstellen ist ihre Typisierung t. Zwei Komponenten sind nur koppelbar, wenn beide Konnektoren vom identischen Schnittstellentyp sind. Deshalb sind die erfolgten Definitionen der Struktur- und Funktionsschnittstellen zu präzisieren:

Schnittstellentypen:
$t : Allgemeiner\ Typ$
$t^{'} : Strukturtyp$
$t_x : Funktionstyp\ mit\ Ausprägung\ Eingang$
$t_y : Funktionstyp\ mit\ Ausprägung\ Ausgang$

Schnittstellen:
$S^{'} = (t^{'},\ )$

Eingang:
$S_x = (t_x,\ )$

Ausgang:
$S_y = (t_y,\ )$

Implizite Kopplungsregel:
$\Omega_K = \{(S_y, S_x) \mid t_1 \in S_y \wedge t_2 \in S_x \wedge t_1 = t_2\}$

Diese einfachen Schnittstellentypen lassen sich durch *Komposition* zu untrennbaren Typelementen vereinen sowie einfache Typen und Typelemente zu trennbaren Typsystemen *aggregieren*. Im Gegensatz zu den oben beschriebenen Elementen und Systemen verfügen Typelemente und Typsysteme nur über strukturelle und keine funktionalen Merkmale. Ihr Zweck dient der reinen Verbindung zweier Elemente oder Systeme.

Formal können komplexe Schnittstellen wie folgt beschrieben werden:

Typelement:
$t_E = T^{'} \cup T_x \cup T_y \cup T_E^-$
$T^{'} = \emptyset \cup \{t^{'}\}$
$T_x = \emptyset \cup \{t_x\}$
$T_y = \emptyset \cup \{t_y\}$
$T_E^- = \emptyset \cup \{t_E^-\}$
$T^{'} \cap T_x \cap T_y \cap T_E^- = \emptyset$
$|T_E| \geq 2$

Typelementschnittstelle:
$S^E = (t_E,\ )$

Typsystem:
$t_S = T^{'} \cup T_x \cup T_y \cup T_E \cup T_S^-$
$T_S^- = \emptyset \cup \{t_S^-\}$
$T^{'} \cap T_x \cap T_y \cap T_E \cap T_S^- = \emptyset$
$|T_S| \geq 2$

Typsystemschnittstelle:
$S^S = (t_S, \ )$

3.2 Steuerungsmotivierte Systemklassifikation

Alle Systeme lassen sich je nach Betrachtungsweise unterschiedlich klassifizieren. Der Schwerpunkt dieser Arbeit liegt auf Steuerungen, weshalb im Folgenden eine entsprechende Systemklassifikation aufgestellt wird (Abb. 3.3).

Die Menge Ω_Σ aller erzeugbaren Systeme ist in zwei Teilmengen zerlegbar. $\Omega_{\Sigma_{STAT}}$ beschreibt die Menge aller *statischen* Systeme, die dadurch gekennzeichnet sind, dass sie einen nicht änderbaren Zustand aufweisen und daher keine Zustandsvariablen zur Beschreibung erforderlich sind. $\Omega_{\Sigma_{DYN}}$ enthält die Menge aller *dynamischen* Systeme, deren Zustände variabel sind:

$$\Omega_\Sigma = \Omega_{\Sigma_{STAT}} \cup \Omega_{\Sigma_{DYN}}$$
$$\Omega_{\Sigma_{STAT}} \cap \Omega_{\Sigma_{DYN}} = \emptyset$$
$$\Omega_{\Sigma_{STAT}} = \{\Sigma \mid Z = \emptyset\}$$
$$\Omega_{\Sigma_{DYN}} = \{\Sigma \mid Z \neq \emptyset\}$$

Dynamische Systeme lassen sich weiterhin in die Menge $\Omega_{\Sigma_{STB}}$ aller *steuerbaren*, die Menge $\Omega_{\Sigma_{ST}}$ aller *steuernden* und die Menge $\overline{\Omega_{\Sigma_{STB}}}$ aller *nicht steuerbaren* Systeme untergliedern. Ein steuerbares System Σ_{STB} empfängt Steuerinstruktionen I eines steuernden Systems Σ_{ST} und passt seinen Zustand entsprechend an. Dazu besteht zwischen beiden Systemen eine Kopplung $(Y_I^{'}, X_I^{'})$. Die Kombination aus beiden Systemen ergibt ein *gesteuertes* System Σ_{STSTB}.

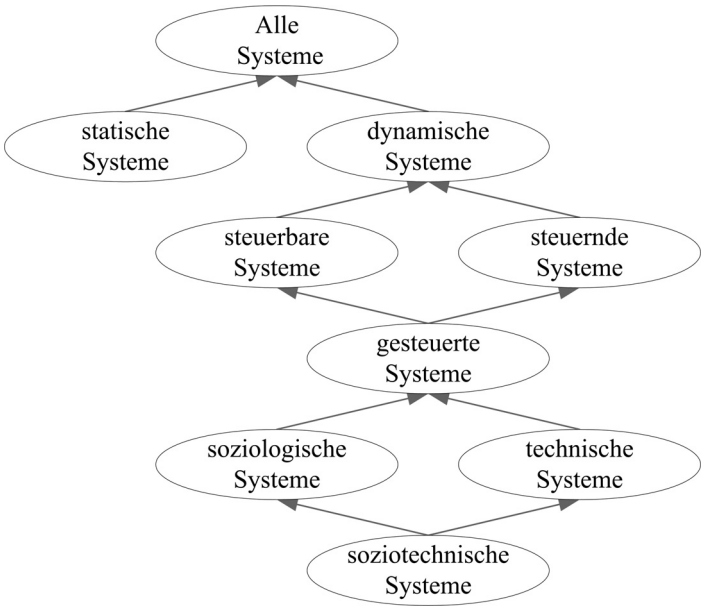


Abbildung 3.3: Klassifikation der Menge aller Systeme aus Steuerungssicht.

Formal können dynamische Systeme wie folgt beschrieben werden:

$$\Omega_{\Sigma_{DYN}} = \Omega_{\Sigma_{STB}} \cup \Omega_{\Sigma_{ST}} \cup \overline{\Omega_{\Sigma_{STB}}}$$
$$\Omega_{\Sigma_{STB}} \cap \Omega_{\Sigma_{ST}} \cap \overline{\Omega_{\Sigma_{STB}}} = \emptyset$$
$$\Omega_{\Sigma_{STB}} = \{\Sigma \mid X_I^{'} \in X\}$$
$$\Omega_{\Sigma_{ST}} = \{\Sigma \mid Y_I^{'} \in Y\}$$
$$\Sigma_{STSTB} = \Sigma_{ST} \cup \Sigma_{STB}$$

Steuernde Systeme kontrollieren nicht nur steuerbare Systeme, sondern können selbst kontrolliert werden. Dieser Umstand führt zu komplexen Steuerungshierarchien. Die Menge dieser *(komplex-)gesteuerten* Systeme ist ebenfalls weiter unterteilbar. In *sozialen* Systemen interagieren Menschen miteinander, während *technische* Systeme alle künstlich vom Menschen geschaffenen sind. Eine Zwischenform bilden *soziotechnische* Systeme, in denen Menschen mit technischen Systemen kooperieren, um ein gegebenes Ziel zu erreichen [86]. In Abhängigkeit des Automatisierungsgrades einer Produktionsanlage zählt diese zu einer der beiden letzten Unterkategorien [86].

3.3 Gesteuerte Systeme und Steuerungsmodelle

Produktionsanlagen zählen mit ihren gesteuerten Komponenten zur Menge der gesteuerten Systeme. Abbildung 3.4 zeigt ein einfaches gesteuertes System, das aus einem steuerbaren Subsystem A und einem steuernden Subsystem B besteht. Beide verfügen über Eingänge XX und Ausgänge YY, mit denen sie gegenseitig und mit ihrer Umwelt in Verbindung stehen. Damit Subsystem B Subsystem A kontrollieren kann, muss es über Ereignisse, die A betreffen, informiert werden. Dabei sind sowohl dessen Eingangsvariablen $\{x\}_A$ als auch sein Zustand ZZ_A relevant, da – von diesen abhängig – Änderungen der Ausgangsvariablen $\{y\}_A$ bewirkt werden. Besteht die Notwendigkeit, dass die Ereignisse der Ausgänge erfasst werden müssen, sind auch diese Variablen relevant. B überwacht durch Einsatz von Sensorik diese Größen. Hierbei ist von Bedeutung, dass B die Werte der Variablen zur weiteren Verarbeitung zwischenspeichern muss, wozu es interne Zustandsvariablen ZZ_B verwendet. Diese werden nachfolgend *Spiegelvariablen* genannt und gliedern sich in *gespiegelte* Eingänge XX_G, Ausgänge YY_G und Zustände ZZ_G:

$$XX_A \subseteq XX \in A \rightarrow XX_G \subset ZZ_B$$
$$YY_A \subseteq YY \in A \rightarrow YY_G \subset ZZ_B$$
$$ZZ_A \subseteq ZZ \in A \rightarrow ZZ_G \subset ZZ_B$$

Um A zu kontrollieren, reichen diese Informationen jedoch noch nicht aus. Zusätzlich ist dessen Verhalten von Bedeutung, wenn eine korrekte Steuerung davon abhängt. Deshalb müssen auch Funktionen $\varphi\varphi_A$ von A nach B gespiegelt werden. Um diese korrekt abzubilden, sind zudem häufig weitere Zustandsvariablen ZZ_F erforderlich:

$$\varphi\varphi_A \subseteq \varphi\varphi \in A \rightarrow \varphi\varphi_G \subset \varphi\varphi_B$$
$$ZZ_F \subset ZZ \in B$$

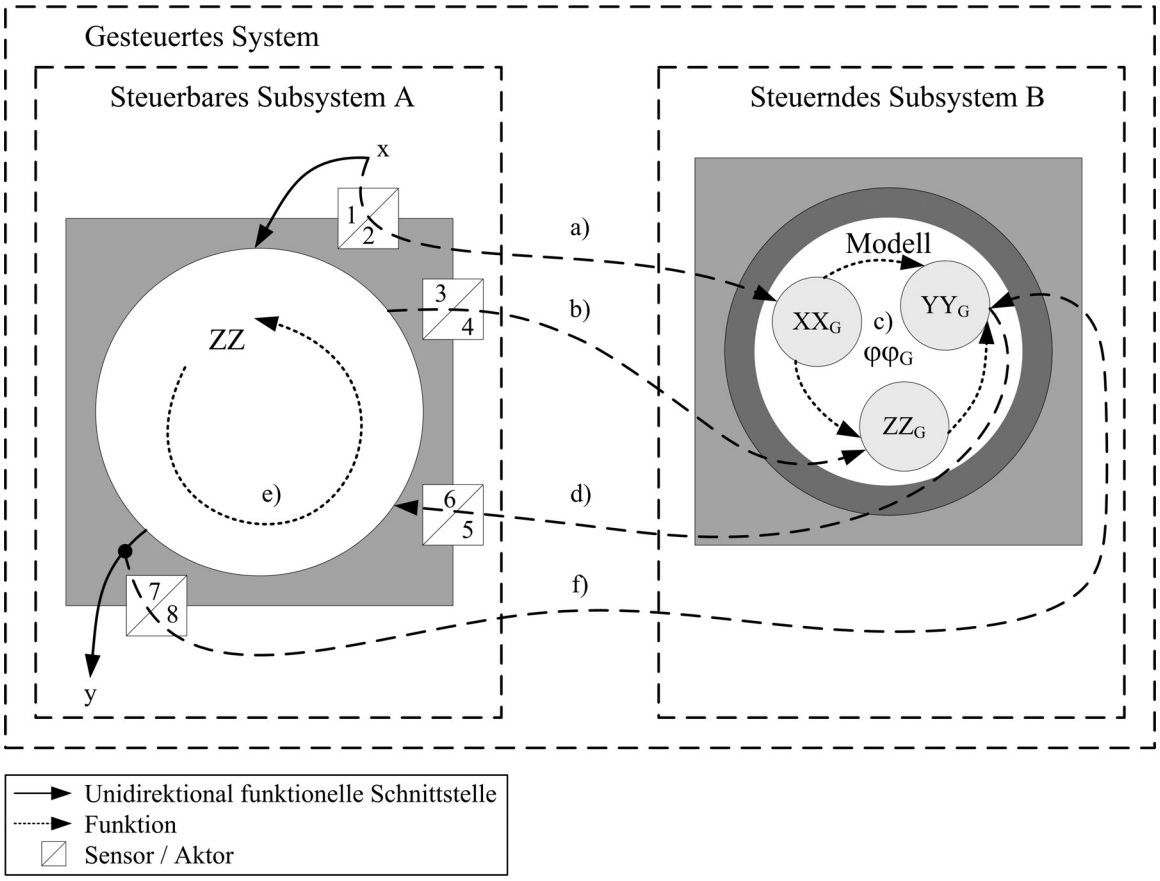


Abbildung 3.4: Modell eines einfachen gesteuerten Systems. a) Registrieren von Veränderungen an den Eingängen und Aktualisierung der gespiegelten Eingänge. b) Zustandsabfrage und Aktualisierung des gespiegelten Zustands. c) Berechnung zu erfolgender Zustands- und erwarteter Ausgangsänderungen. d) Instruktionen e) Durchführung der Zustandsänderung. f) Ermittlung tatsächlicher Ausgangsänderungen. $\varphi\varphi_G$ = gespiegelte Funktionen; x = Eingangsvariable; XX_G = gespiegelte Eingänge; y = Ausgangsvariable; YY_G = gespiegelte Ausgänge; ZZ_G = gespiegelter Zustand; ZZ = Zustand.

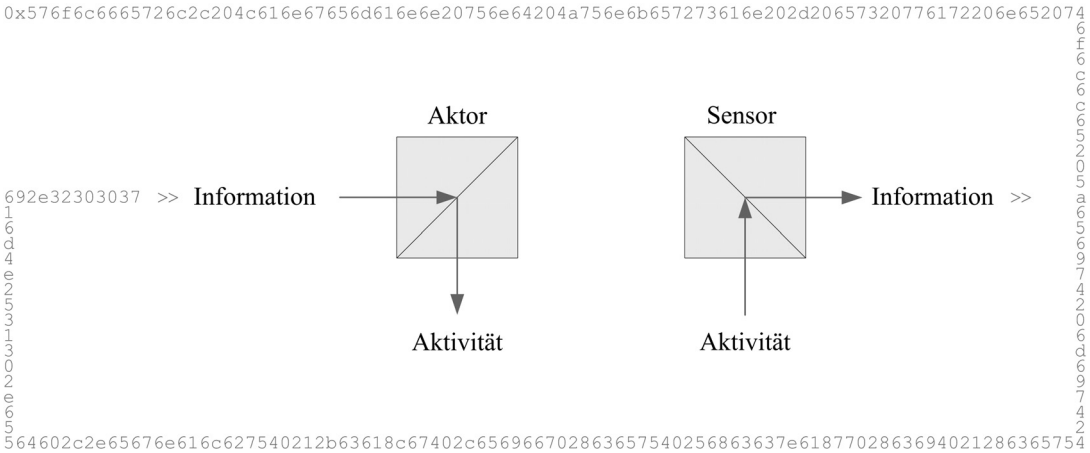


Abbildung 3.5: Prinzip der Umwandlung von Information in Aktivität und umgekehrt.

Zusammen bilden alle Spiegelvariablen und Spiegelfunktionen das *Steuerungsmodell* von A, auf dessen Grundlage die Erzeugung von Steuerinstruktionen erfolgt. Diese werden an das gesteuerte System weitergeleitet, das diese mit Hilfe von Aktoren in Aktionen umsetzt. Sensoren und Aktoren nehmen einen besonderen Platz ein, da sie Information in Aktion und Aktion in Information umsetzen. Sie sind damit Bindeglieder zwischen physischer und informationstechnischer Welt. Abbildung 3.5 zeigt dies schematisch.

Auf dieser Grundlage kann ein gesteuertes System in die beiden Teilsysteme *Informations-* und *Ausführungssystem* unterteilt werden (Begrifflichkeiten nach [86]). Ersteres besteht aus dem steuernden System, den Signalleitungen sowie den Informationsquellen der Sensoren und Informationssenken der Aktoren. Zweiteres besteht aus Aktivitätssenken der Sensoren und Aktivitätsquellen der Aktoren sowie der physischen Beschaffenheit des gesteuerten Systems.

3.4 Schlussfolgerungen zur Gestaltung der Wandelbarkeit

Die theoretische Betrachtung von Systemen erlaubt es, Rückschlüsse auf Anforderungen zur Gestaltung wandelbarer Produktionsanlagen zu ziehen.

Durchgängige Modularisierung mit nach innen gerichtetem Wirkbereich der Module

Die wichtigste aus der Systemdefinition abzuleitende Forderung ist eine stringente Modularisierung der Systemkomponenten. Dies schließt zum einen eine *funktionale Abgeschlossenheit* [42, 113] der Module und zum anderen eine konsequente *Innengerichtheit des Wirkbereichs* ein. Ersteres geht mit einem hohen anzustrebenden Autarkiegrad [42] einher, d.h. Module sollten ihre Funktionsfähigkeit möglichst ohne Zuhilfenahme anderer Module erlangen. Dies ist jedoch nur bei Modulen höherer Ordnung möglich. Zweiteres bedeutet, dass Module keinen Wirkbereich über ihre Modulgrenzen hinaus haben sollten, da sich dies auf Nachbar- oder Supermodule störend auswirken kann. Solche Störungen hätten eine Minderung der Wandlungsfähigkeit und damit auch der Wandlungseffizienz zur Folge.

Ganzheitliche Betrachtung der Schnittstellen

Die Rolle der Schnittstellen mit ihrer Bedeutung für die Kopplungsfähigkeit von Komponenten wurde bereits beleuchtet (Kap. 3.1.4). Zusammen mit der Darstellung von Produktionsanlagen als Flusssysteme in Kapitel 2.1.2 ergibt sich die Notwendigkeit einer ganzheitlichen Betrachtung aller Schnittstellen aus den Bereichen Stoff-, Energie-, Informations- und Signalfluss. Dies gilt neben physischen insbesondere auch für nicht physische Schnittstellen von Software. Durch die Betrachtung modularer Baukastensysteme (Kap. 2.4.3) wird jedoch schnell ersichtlich, dass neben der Betrachtung von Flüssen auch Aspekte der Mechanik wesentlich sind. Hierzu sind sämtliche struktur- und funktionsbildenden Schnittstellen von Bedeutung, wobei Letztere dem Austausch physikalischer Größen dienen.

Abgeschlossene Menge von Schnittstellentypen

Für die Gewährleistung einer effektiven Wandelbarkeit ist die Anzahl an verfügbaren Schnittstellentypen zu begrenzen. Dies dient zur Wahrung der Übersichtlichkeit, der Vermeidung überflüssiger Redundanzen und einer hohen Kompatibilität zwischen Modulen. Bei der Spezifikation sind die notwendigen Interaktionsbedarfe vollständig zu erfassen, so dass eine abgeschlossene, minimale Menge an Schnittstellentypen entsteht, die nachträglich möglichst nicht mehr erweitert werden muss. Ähnliche Typen sollten zu Gunsten einer einzigen zusammengefasst werden.

Ein weiterer Aspekt besteht in der Definition möglichst universeller Typelemente oder Typsysteme. Sind zwischen Modulen häufig oder immer eine Anzahl gleicher Schnittstellentypen erforderlich, bzw. existiert eine Vereinigungsmenge von Schnittstellen, die in einer größeren Anzahl an Modulen zu implementieren sind, bietet sich die Bildung von Typenelementen an. Lässt sich diese Vereinigungsmenge in disjunkte Teilmengen zerlegen, die in unterschiedlichen Kombinationen Einsatz finden, so ist die Schaffung von Typsystemen vorteilhaft. Durch diese Bündelung können Schnittstellenvielfalt sowie Kontaktierungsaufwendungen reduziert werden.

Grenzen der Hierarchieebenen

Da Systeme in der Regel hierarchische Gebilde sind, über deren Schachtelungstiefe a priori keine begrenzenden Aussagen getroffen werden können, kann gefolgert werden, dass auch für Produktionsanlagen eine beliebige Schachtelungstiefe anzunehmen ist. Die in der Literatur unterschiedlich benannten sieben oder neun Ebenen von Produktionsanlagen beschreiben diese Hierarchie nur grob [42, 86]. Tatsächlich weisen Produktionsanlagen wesentlich komplexere Hierarchien auf. Einzig können konkrete Aussagen über das obere und untere Ende der Hierarchie getroffen werden. Nach oben bildet die Anlagenebene und nach unten einerseits – aus Steuerungssicht – die Sensor-/Aktor-Ebene und andererseits – aus mechanischer Sicht – die kleinsten physischen Komponenten die Abschlüsse. Der Abschluss aus Steuerungssicht ist damit zu begründen, dass Sensoren wie Aktoren Schnittstellen zwischen der physischen und der virtuellen Welt darstellen. Eine weitere Untergliederung ist deshalb nicht sinnvoll. Diese Elemente stellen somit die kleinsten, unteilbaren *Basiselemente* einer Produktionsanlage dar.

Wandelbare Steuerungsmodelle und Zwillingsbausteine

Die Diskussion gesteuerter Systeme hat gezeigt, dass ein Informationssystem zum einen immer ein Modell des Ausführungssystems beinhaltet und zum anderen selbst auch gesteuert sein kann. Steuerungsmodelle sind dabei immer auch selbst Systeme, da sie Bestandteile des Ausführungssystems sind und deren Verknüpfungen reflektieren.

Für die Wandelbarkeit folgt daraus, dass Steuerungsmodelle ebenfalls hierarchisch modular gestaltet sein müssen, da Wandlungen im Ausführungssystem immer mit Wandlungen des Steuerungsmodells einhergehen (Abb. 3.6). Als Konsequenz ergibt sich die Forderung einer Isomorphie zwischen Ausführungssystem und Steuerungsmodell, d.h. die Komponenten und Strukturen müssen einander gleichen.

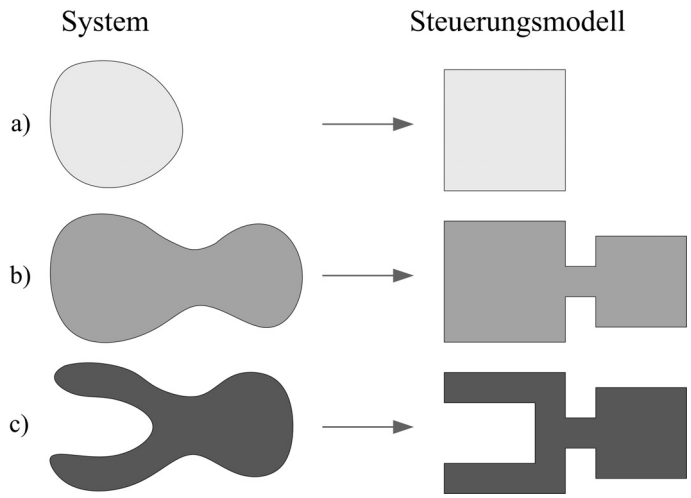


Abbildung 3.6: Wandlung eines Systems und seines Steuerungsmodells über drei Stufen a) - c).

Dies kann erreicht werden, indem Ausführungssysteme und Steuerungsmodelle jeweils aus einer definierten Menge isomorpher Bausteintypen aufgebaut werden. Für jeden, aus Steuerungssicht relevanten physischen Baustein, muss dabei ein Modellpendant existieren. Ein Paar aus einem physischem und einem Modellbaustein wird nachfolgend *Zwillingsbaustein* genannt.

Trennung von Steuerungsmodellen und Steuerungstechnik

Aus den oben genannten Gründen ist es für die Wandelbarkeit sinnvoll, Steuerungsmodelle von der Steuerungstechnik zu trennen. Damit ist gemeint, dass ein Modell aufwandsarm einerseits in Teilstücke zerlegbar sein sollte, so dass es entweder zentral auf einem oder verteilt auf mehreren Steuerrechnern ablauffähig ist (Ortsunabhängigkeit der Ausführung). Andererseits sollte es nicht auf eine bestimmte Hardware zugeschnitten werden, um eine spätere Migration auf neue Steuerungstechnik zu ermöglichen.

Verteilter Einsatz von Steuerungstechnik

Es ist von Vorteil, Module mit eigener Steuerungstechnik auszustatten, da dies unter Verwendung zerlegbarer Steuerungsmodelle den an die Wandelbarkeit gestellten Anspruch der aufwandsarmen Modifizierbarkeit unterstützt. In der praktischen Umsetzung ist der Wandelbarkeitsvorteil aus wirtschaftlichen Aspekten dem Investitionsbedarf für Rechen- und Kommunikationstechnik gegenüber zu stellen. Sind Steuerungsmodelle zerlegbar, sind sie im Umkehrschluss auch kombinierbar, so dass unterschiedliche Dezentralisierungsgrade aufwandsarm abgedeckt werden können.

Angemerkt sei an dieser Stelle, dass in einem wandelbaren Umfeld Module immer identifiziert und strukturelle Verknüpfungen ermittelt werden müssen. Dies kann manuell oder elektronisch erfolgen. Während eine manuelle Vorgehensweise mit hohem Zeitaufwand und Fehlerrisiko behaftet ist, sind im anderen Fall – auch bei zentralen Steuerungsansätzen – elektronische Schaltungen zur Erfassung

der Modulidentität erforderlich. Somit ist unter Zielsetzung einer aufwandsarmen Wandlung in jedem Fall Elektronik zu verwenden.

Trennung von Ablauf- und Ausführungsteuerung

Der Definition der Teilsysteme folgend, ist es möglich, das Steuerungsmodell eines gesteuerten Gesamtsystems und dessen Submodelle in die zwei Bereiche *Ablauf-* und *Ausführungssteuerung* zu unterteilen.

Die *Ablaufsteuerung* übernimmt logistische, koordinierende, ablauftechnische, organisatorische und verwaltungstechnische Aufgaben, die unabhängig von der konkreten Beschaffenheit des zu steuernden Gegenstands sind. Zur Verdeutlichung seien an dieser Stelle Roboter genannt, die im Wesentlichen durch ihre Kinematik und nicht durch den Hersteller bestimmt sind. Abläufe für Objekte gleichen Typs sind identisch und müssen nicht spezifisch angepasst werden. Die Ablaufsteuerung ist primär an Nachbarn der horizontalen und der Superebene gekoppelt.

Die *Ausführungssteuerung* ist effektiv für die konkrete Durchführung von Aufgaben verantwortlich und ist damit direkt abhängig von der Beschaffenheit des zu steuernden Gegenstands. Um den Unterschied am Beispiel der Roboter zu verdeutlichen – während die Steuerbefehle für Roboter mit gleicher Kinematik zweier unterschiedlicher Hersteller identisch sein können (Ablaufsteuerung), ist die Ausführungssteuerung (genauer die Regelkreise) gerätespezifisch.

Bei sorgfältiger Trennung ist die Ablaufsteuerung unabhängig von dem physischen Gegenstand, so dass sie bei gleichen oder ähnlichen Gerätetypen nicht geändert werden muss. Lediglich der tatsächlich gerätespezifische Teil ist entsprechend anzupassen.

Der Mensch als gesteuertes System

Eine Besonderheit der Wandelbarkeit ist das Potenzial, die Leistung einer Produktionsanlage an Bedarfen auszurichten, indem unter anderem der Automatisierungsgrad verändert wird. Neben Geräten sind auch Menschen an verschiedenen Positionen einsetzbar. Diese können bei hinreichender Abstraktion ebenfalls als gesteuerte Systeme aufgefasst werden und verfügen daher über ein Informations- und Ausführungssystem. So können Sie als Ablaufsteuerung, als Ausführungssteuerung oder als Ausführungssystem agieren.

Soll ein reibungsfreier Wechsel zwischen Automatisierungsgraden möglich sein, müssen Schnittstellen übergeordneter Informationssysteme derart ausgelegt sein, dass sowohl menschliche als auch technische Ausführungssysteme ankoppelbar sind. Soll ein Mensch auch steuernd in Erscheinung treten, sind Schnittstellen niedrigerer Ausführungssysteme ebenfalls entsprechend zu gestalten. Abbildung 3.7 verdeutlicht diesen Sachverhalt.

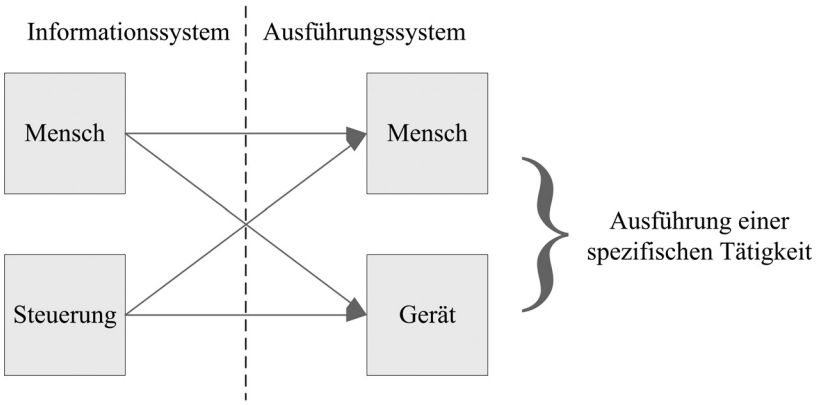


Abbildung 3.7: Mensch und Gerät als Informations- und Ausführungssystem.

Aufbau von Baukastensystemen

Da die Modularisierung von Systemkomponenten zeitliche wie finanzielle Investitionen erfordert, ist die Entwicklung von Baukastensystemen vorteilhaft. Dadurch wird eine Verteilung dieser Aufwendungen auf mehrere Produktionsanlagen möglich. Nach [85] ist der Begriff „Baukastensystem“ wie folgt definiert:

> *„Das Baukastenprinzip ist ein Gestaltungsprinzip der Systemsynthese, das darin besteht, aus einem gegebenen Repertoire von Elementen mit definierter Teilfunktion zahlreiche verschiedene Systeme mit unterschiedlicher Gesamtfunktion dadurch herzustellen, daß die Elemente in unterschiedlicher Auswahl und in verschiedenartigen Relationen kombinatorisch miteinander verknüpft werden.*
>
> *Als Baukastensystem bezeichnet man das Repertoire von Elementen, das für die Anwendung des Baukastenprinzips erforderlich ist.“*

Baukastensysteme können in unterschiedlichen Ausprägungen realisiert sein, in Abhängigkeit des abzudeckenden Bereichs an Automatisierungsgraden und der nach unten begrenzenden hierarchischen Ebene. Diese untere Grenze muss auf Steuerungsseite nicht zwingend der Sensor-/Aktor-Ebene entsprechen, da dies aus wirtschaftlichen wie praktischen Gesichtspunkten nicht in jedem Fall möglich ist.

Ein Baukastensystem besteht aus einer abgeschlossenen Menge kleinster Grundeinheiten, die sukzessive und hierarchisch zu immer komplexeren Einheiten zusammengesetzt werden. Auf hoher Stufe stehen Produktionseinrichtungen, die zu Zellen, Produktionslinien oder -inseln und schließlich zu Produktionsanlagen kombiniert werden. Komplexere Einheiten können in das Baukastensystem übernommen werden, um mehrfache Entwicklungsaufwendungen zu vermeiden. Aus praktischen Erwägungen ist auch die Integration bereits bestehender, komplexer und nicht modular aufgebauter Einheiten möglich, indem ihre Schnittstellen angepasst werden.

Jedes Baukastensystem, das mehr als nur den Aufbau rein manueller, nichtrechnergestützter Produktionsanlagen zum Ziel hat, lässt sich in drei Hauptmodulkategorien untergliedern. Die erste Kategorie beinhaltet rein physische Bausteine, die mechanisch gekoppelt werden. Beispiele hierfür sind Gehäuse, Gestelle, Grundplatten, Tische, Regale, genauso aber auch Steuerrechner und Kabel. Die zweite

Kategorie bilden oben genannte Zwillingsbausteine, somit also alle gesteuerten Komponenten, wie Achsen, Motoren, Greifer, Stopper, usw. Die dritte Kategorie schließlich besteht aus Softwarekomponenten zur Datenerfassung und Verarbeitung sowie zur Ablauforganisation auf höheren Steuerungsebenen. Dazu zählen bspw. Flusssteuerungen, Auftragseinlastung und Materialverwaltung.

3.5 Systemkonzept für wandelbare Produktionsanlagen

Unter Einbeziehung der vorangegangenen Schlussfolgerungen wird nun ein theoretisches Systemkonzept aufgestellt, auf dessen Grundlage Baukastensysteme erstellt werden können. Es definiert die fünf Bausteinkategorien *Basiselemente, Zwillings-Basiselemente, Elemente, Zwillingselemente* und *Verbünde*. Sie untergliedern sich zum Teil in *physische* und *Modellelemente* sowie in *Struktur-* und *Funktionselemente*. Da bereits beim Überblick der Systemtheorie ausführliche formale Definitionen erfolgten, sind nachfolgend nur die wichtigsten zusätzlichen Eigenschaften formalisiert (Eine Anwendung des Systemkonzepts außerhalb des Rahmens der vorliegenden Arbeit findet sich in [32]).

3.5.1 Basiselemente

Basiselemente BE stellen die kleinsten unteilbaren Elemente des Systemkonzepts dar und sind gleichbedeutend mit dem Elementebegriff der allgemeinen Systemtheorie (Kap. 3.1):

Basiselement:
$BE = (S, X, Y, Z, \varphi)\ mit\ S \cup X \cup Y \neq \emptyset$

Die Menge Ω_{BE} aller Basiselemente lässt sich zum einen in die beiden Teilmengen *Physische-* $\Omega_{P_{BE}}$ und *Steuerungsmodell-Basiselemente* $\Omega_{M_{BE}}$ und zum anderen orthogonal in die Teilmengen der *Struktur-* $\Omega_{S_{BE}}$ und *Funktions-Basiselemente* $\Omega_{F_{BE}}$ zerlegen. Basiselemente $S_{BE} \in \Omega_{S_{BE}}$ dienen rein zum Aufbau von Strukturen, weisen dazu Strukturschnittstellen auf, und sind zustandslos. Dagegen verfügen Basiselemente $F_{BE} \in \Omega_{F_{BE}}$ über strukturbildende Merkmale hinaus auch über Funktionalität, Funktionsschnittstellen und einen änderbaren Zustand. Durch Schneiden der Teilmengen lassen sich die vier disjunkten Subkategorien *Physische-Struktur-, Physische-Funktions-, Modell-Struktur-* und *Modell-Funktions-Basiselemente* erzeugen, deren Elementeigenschaften nachfolgend beschrieben werden.

Physische-Basiselemente P_{BE} sind alle physisch „greifbaren“ Grundbausteine. Die Strukturschnittstellen sind immer mechanischer und die Funktionsschnittstellen stofflicher, energetischer oder signaltechnischer Natur bzw. dienen zum Austausch physikalischer Größen. Funktionselemente $P_{F_{BE}}$ sind Objektquellen $P_{FQ_{BE}}$, -transformatoren $P_{FT_{BE}}$, -senken $P_{FS_{BE}}$ oder mechanische Funktionselemente $P_{FP_{BE}}$. Quellen schleusen stoffliche Flussgegenstände in ein System ein, Senken entfernen sie. Transformatoren verändern den Zustand eines Gegenstands. Über typisierte Funktionsschnittstellen werden Flussgegenstände zwischen physischen Funktions-Basiselementen ausgetauscht.

Die beiden physischen Kategorien sind formal wie folgt beschrieben:

Physisches-Struktur-Basiselement:
$P_{S_{BE}} = (S, \emptyset, \emptyset, \emptyset, \emptyset)$

$$S = \{S_i^{'}\}\ ;\ mit\ i = 1, , I \wedge I \in \mathbb{N} \wedge S^{'}\ ist\ mechanisch$$

Physisches-Funktions-Basiselement:
$P_{F_{BE}} = (S, X, Y, Z, \varphi)$
$S = \{S_i^{'}\}\ \ mit\ i = 1, , I \wedge I \in \mathbb{N} \wedge S^{'}\ ist\ mechanisch$
$X = \{X_i^{'}\}\ \ mit\ i = 1, , I \wedge I \in \mathbb{N} \wedge x \in$
$\{stofflich,\ energetisch,\ signaltechnisch\ physikalisch\}$
$Y = \{Y_i^{'}\}\ \ mit\ i = 1, , I \wedge I \in \mathbb{N} \wedge y \in$
$\{stofflich,\ energetisch,\ signaltechnisch\ physikalisch\}$

Modell-Basiselemente M_{BE} sind alle Basiselemente zum Aufbau von Steuerungsmodellen. Als Struktur-Basiselement $M_{C_{BE}}$ existiert mit dem *Knoten* ein Einziges. Er fungiert als Container und kann beliebig viele andere Modellbasiselemente, aber auch die später beschriebenen Elemente E und Verbünde V aufnehmen. Über Strukturschnittstellen ist der Knoten mit allen seinen Mitgliedern referenziell verbunden. Dadurch entsteht eine Baumhierarchie, über die alle Modell-Basiselemente eines Systems erreichbar sind. Funktionselemente $M_{F_{BE}}$ sind Informationsquellen $M_{FQ_{BE}}$, -transformatoren $M_{FT_{BE}}$ oder -senken $M_{FS_{BE}}$. Quellen schleusen Informationsobjekte in ein System ein, Senken entfernen sie. Transformatoren verarbeiten oder konvertieren Informationen und implementieren einfache wie komplexe Funktionen. Über typisierte Funktionsschnittstellen werden Informationsobjekte zwischen Modell-Funktions-Basiselementen ausgetauscht.

Im Gegensatz zu physischen Schnittstellen verfügen Modellschnittstellen über eine Kapazität $K \geq 1$, d.h. über die Schnittstelle eines Basiselements können maximal K andere Basiselemente gekoppelt werden. Dies liegt darin begründet, dass Software virtuell ist und grundsätzlich beliebig viele Referenzen auf ein Objekt bzw. eine Datenstruktur (genauer eine Speicheradresse) bestehen können. Falls dies nicht unbedingt erwünscht ist, ist die Kapazität entsprechend zu begrenzen.

Die beiden Modellkategorien sind formal wie folgt beschrieben:

Modell-Struktur-Basiselement:
$M_{C_{BE}} = (S, \emptyset, \emptyset, \emptyset, \emptyset)$
$S = \{S_i^{'}\}\ \ mit\ i = 1, , I \wedge I \in \mathbb{N} \wedge S^{'}\ ist\ referenziell$

Modell-Funktions-Basiselement:
$M_{F_{BE}} = (\emptyset, X, Y, Z, \varphi)$
$S = \{S_i^{'}\}\ \ mit\ i = 1, , I \wedge I \in \mathbb{N} \wedge S^{'}\ ist\ referenziell$
$X = \{X_i^{'}\}\ ;\ mit\ i = 1, , I \wedge I \in \mathbb{N} \wedge x\ ist\ informationell$
$Y = \{Y_i^{'}\}\ ;\ mit\ i = 1, , I \wedge I \in \mathbb{N} \wedge y\ ist\ informationell$

3.5.2 Zwillings-Basiselemente

Zwillings-Basiselemente Z_{BE} sind 2-Tupel aus einem physischen P_{BE} und einem Modell-Basiselement M_{BE}, die zueinander isomorph sind. Keines der beiden Basiselemente ist für sich allein genommen funktionstüchtig. Wieder lassen sich Struktur- $Z_{S_{BE}}$ und Funktions-Zwillings-Basiselemente $Z_{F_{BE}}$ unterscheiden. Ein Beispiel für $Z_{S_{BE}}$ ist ein Gehäuse, das physische Funktionsbausteine enthält,

und deshalb einen Knoten für die entsprechenden Modellpendants benötigt. Zur Menge der Funktionsbausteine zählen alle einfachen Sensoren und Aktoren mit einer einzelnen Wirkdimension, da sie sowohl physischer als auch informationstechnischer Natur sind. Beispiele sind Kraft- oder Drehmoment, Drehzahl- oder Temperatursensoren sowie An/Aus-Schalter oder Drehzahlgeber. Sensoren und Aktoren mit mehrdimensionalen Wirkbereich, etwa 3d-Kraftsensoren, oder kombinierte Sensor-Aktor-Systeme, etwa Motoren, werden nicht als Basiselemente aufgefasst. Sie können stattdessen aus Zwillings-Basiselementen zusammengesetzt werden.

Die beiden Modellkategorien sind formal wie folgt beschrieben:

Zwillings-Struktur-Basiselement:
$Z_{S_{BE}} = (P_{S_{BE}}, M_{S_{BE}})$

Zwillings-Funktions-Basiselement:
$Z_{F_{BE}} = (P_{F_{BE}}, M_{F_{BE}})$

3.5.3 Elemente

Elemente entstehen durch die *Komposition* von Sub- und Basiselementen. Sie sind durch die Eigenschaft charakterisiert, dass ihre Komponenten zu einem Ganzen „verschmolzen“ sind, welches konzeptionell als untrennbar angesehen wird. In seiner Natur kann ein Element rein physisch P_E sein oder ein reines Steuerungsmodell S_E darstellen. Da Elemente auch aus Subelementen bestehen können, können sich (entartete) Baumhierarchien aus Superelement, Element und Subelementen ergeben. Eine Folge davon ist, dass die Trennung auf höheren Elementebenen in physisch und Steuerungsmodell nicht aufrecht zu erhalten ist und Mischformen entstehen.

Element-Schnittstellen sind die nicht verbundenen Schnittstellen der Elementbestandteile. Allerdings gilt, dass nicht notwendigerweise alle Schnittstellen nach außen geführt sein müssen. So ist es möglich, trotz identischem inneren Aufbau, verschiedene Elemente zu schaffen.

Die Elementdefinition folgt formal der Systemdefinition der allgemeinen Systemtheorie (Kap. 3.1) und ist wie folgt abgewandelt:

Element:
$E = (B, FT, ST)$

Bestandteile:
$B = \{BE\} \cup \{E^-\}$

Nach außen geführte Schnittstellen:
$SS \subseteq \{S \mid (S, *) \notin \lambda \vee (S, *) \notin \lambda\}$
$XX \subseteq \{S_x \mid (*, S_x) \notin \lambda\}$
$YY \subseteq \{S_y \mid (S_y, *) \notin \lambda\}$

3.5.4 Zwillings-Elemente

Zwillings-Elemente Z_E sind ein 2-Tupel aus einem physischen P_E und einem Modell-Basiselement M_E, die zueinander isomorph sind. Keines der beiden Basiselemente ist für sich allein genommen funktionstüchtig. Im Gegensatz zu Zwillings-Basiselementen muss nicht zu jedem Bestandteil des physischen Elements ein Steuerungsmodellpendant existieren. Lediglich eine Schnittmenge der für die modulare Steuerung relevanten Teile muss vorhanden sein:

Zwillings-Element:
$Z_E = (E_P, E_M)$

Bestandteile von E_P und E_M :
$B_Z = \{(p_1, p_2) \mid p_1 \cong p_2 \wedge p1 \in B_P \wedge p2 \in B_M\}$
$B_M = \{p \mid p \in B_P \wedge p \notin B_Z\}$
$B_P = \{p \mid p \in B_M \wedge p \notin B_Z\}$
$B = B_Z \cup B_M \cup B_P$

3.5.5 Verbünde

Verbünde V entstehen durch die *Aggregation* von Subverbünden, Elementen und Basiselementen. Im Gegensatz zu Elementen kann ein Verbund konzeptionell in seine Bestandteile zerlegt werden. Es setzt somit die Voraussetzungen für die Wandelbarkeit von Systemen um.

Ebenfalls konträr zu den Elementen sind *alle* nicht verbundenen Schnittstellen der Systembestandteile nach außen verfügbar.

Die Verbunddefinition ist äquivalent zur Systemdefinition der allgemeinen Systemtheorie (Kap. 3.1) und ist wie folgt abgewandelt:

Verbund:
$V = (B, FT, ST)$

Bestandteile:
$B = \{BE\} \cup \{E\} \cup \{V^-\}$

Nach außen geführte Schnittstellen:
$SS = \{S \mid (S, *) \notin \lambda \vee (S, *) \notin \lambda\}$
$XX = \{S_x \mid (*, S_x) \notin \lambda\}$
$YY = \{S_y \mid (S_y, *) \notin \lambda\}$

Kapitel 4

Konzeption einer wandelbaren Systemlandschaft

Aufbauend auf der Beschreibung der Wandlungsfähigkeit von Produktionsanlagen (Kap. 2.4.2) und den Vorüberlegungen zur Gestaltung wandelbarer Produktionsanlagen (Kap. 3) wird in diesem Kapitel – als ein Fundament – das Konzept einer rechnergestützten wandelbaren Systemlandschaft erarbeitet.

Das Konzept fußt auf dem hierarchischen *6-Ebenen$^+$-Modell*, das sechs Hauptebenen beschreibt, auf denen Wandlung geschehen kann. Auf diesen Ebenen finden sich *strukturierte und unstrukturierte Bereiche*, die im Anschluss beschrieben werden. Sie geben einerseits Hinweis auf die Art der zu verwendenden Kommunikationstechnologie, andererseits sind sie für deren Selbstkonfiguration und die Selbstorganisation von Stoff- und Informationsflüssen von Bedeutung. Das Konzept postuliert, dass alle Softwaresysteme auf einer einheitlichen Softwarearchitektur basieren und eine *homogene informationelle Vernetzung* möglich ist. Sie dient einem durchgängigen horizontalen und vertikalen Informationsaustausch. Die Softwaresysteme setzen strikt die Erkenntnisse über gesteuerte Systeme (Kap. 3.3) um und basieren auf Steuerungsmodellen, aufgebaut aus Zwillingsbausteinen und Steuerungsmodell-(Basis-)Elementen (Kap. 3.5). *Steuerungssoftwaresysteme* sind damit wandelbar, Modifikationen von Produktionseinrichtungen können steuerungstechnisch aufwandsarm durchgeführt werden. Zur Erfüllung logistischer und organisatorischer Aufgaben, die zum Betrieb einer Produktionsanlage benötigt werden, sind Grunddienste als *selbstadaptierende, generische Informationssysteme* vorgesehen.

4.1 6-Ebenen$^+$-Modell

Das Modell umfasst die sechs Hauptebenen *1 – Produktionsanlage, 2 – Lager, Produktionslinien, -inseln und Transportsysteme, 3 – (Sub-)Zellen, 4 – Produktions-/Transfereinrichtungen, 5 – (Sub-) Module* sowie *6 – Basismodule*. Ebenen drei und fünf können Unterebenen aufweisen, wobei die Schachtelungstiefe nicht begrenzt ist. Deshalb ist das Modell als *6-Ebenen$^+$-Modell* benannt. Es ist zum besseren Verständnis in Abbildung 4.1 dargestellt.

Das Konzept postuliert die homogene Zusammensetzung einer Produktionsanlage aus Bausteinen eines Baukastensystems, das stringent dem theoretischen Systemkonzept folgt (Kap. 3.5). Ein weiteres Postulat ist die konsequente Beachtung des Prinzips der Innengerichtheit des Wirkbereichs (Kap. 3.4). Änderungen innerhalb eines Bausteins wirken sich demnach nur auf den Baustein selbst und seine Subbausteine aus, nicht jedoch auf benachbarte oder übergeordnete Bausteine. Die einzelnen

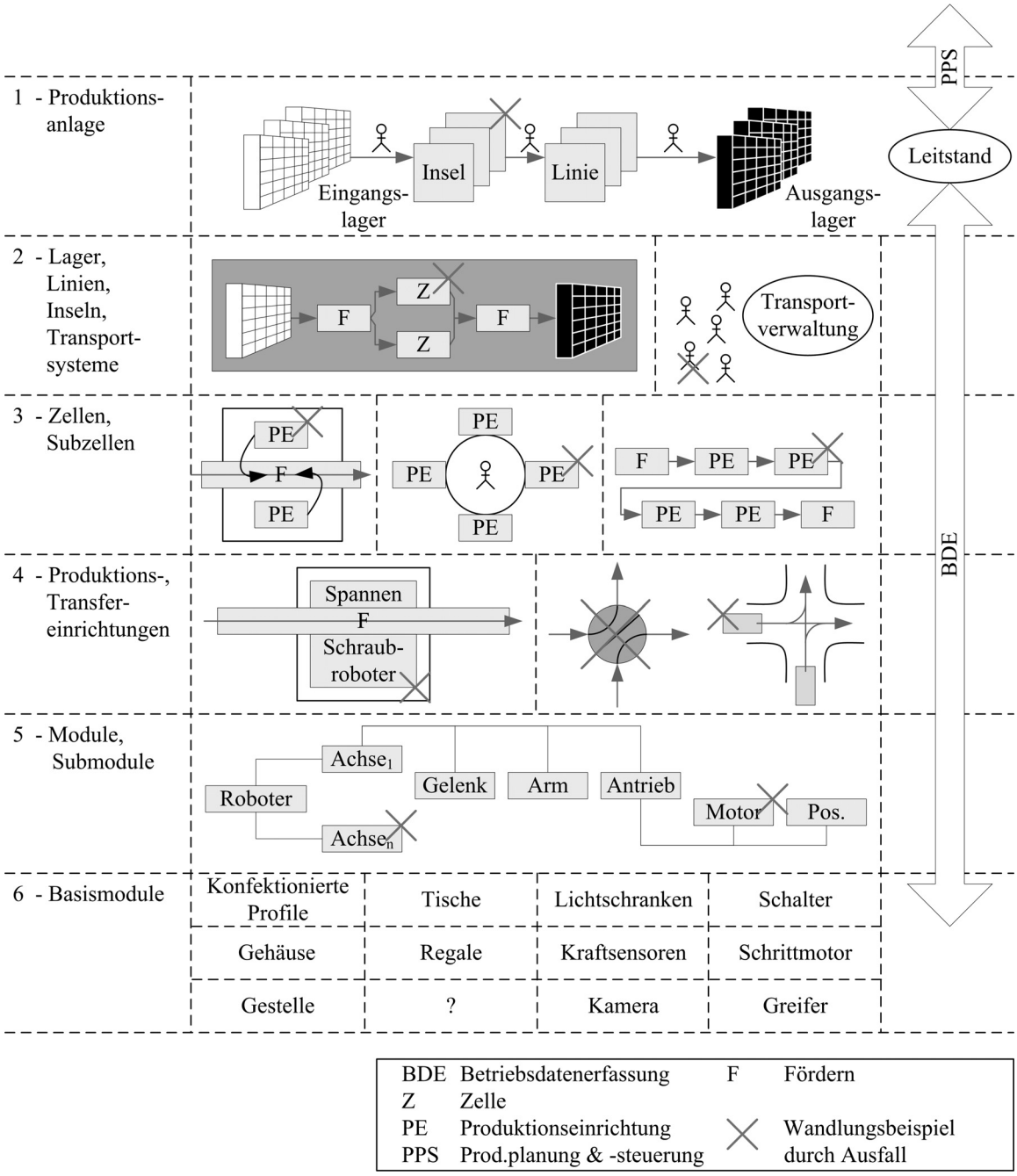


Abbildung 4.1: Darstellung des 6-Ebenen$^+$-Modells.

Hauptebenen werden nachfolgend näher beschrieben, mögliche Bausteintypen des theoretischen Konzepts und Wandlungspotenziale sowie entstehende Anpassungsbedarfe benannt. Die Bestandteile einer Ebene sind dabei – der Hierarchie von Systemen folgend (Kap. 3.1.2) – immer Bausteine der nächst niedrigeren Ebene.

4.1.1 Hauptebene 1 – Produktionsanlage

Die *Produktionsanlage* ist der ersten Hauptebene zugeordnet. Sie bildet einen Verbund von *Lagern, Produktionslinien/-inseln* und *Transportsystemen*. Lager treten als *Eingangs-, Zwischen-* oder *Ausgangslager* in Erscheinung. Erstere versorgen die Anlage mit Roh-, Fertigungs-, Kauf- oder Normteilen, Vorerzeugnissen sowie Hilfsstoffen, Werkzeugen und Transferhilfsmitteln, Letztere nehmen fertige Erzeugnisse auf, so dass sie der Anlage entzogen werden. Die zu Produktionslinien und -inseln zusammengefassten Produktionseinrichtungen dienen zur Herstellung von (Vor-)Erzeugnissen. Transportsysteme sind das Bindeglied zwischen verschiedenen Anlagenbereichen und führen Lager $\leftrightarrow$ Linie/Insel-Transporte und Linie/Insel $\leftrightarrow$ Linie/Insel-Transporte durch. Zur Koordinierung der Transportaufträge verfügt jedes Transportsystem über eine *Transportverwaltung*, die Aufträge entgegen nimmt und an die Transporteinheiten verteilt.

Die Koordination und Überwachung der Produktionsanlage erfolgt durch einen *Leitstand*, der nach oben an die Produktionsplanung und -steuerung und nach unten an eine *Betriebsdatenerfassung* gekoppelt ist. Diese ist ein sich über alle Ebenen erstreckendes Teilsystem der Produktionsanlage, das an die Ablaufsteuerungen (Kap. 3.4) aller gesteuerten Komponenten gekoppelt ist.

Auswirkungen auf die erste Hauptebene haben, aus Sicht der Wandelbarkeit, Änderungen an *Gebäuden, Linien, Inseln* und *Transportsystemen*. Änderungen am Gebäude, aus physischer Sicht das Supersystem der Produktionsanlage [41], können Anpassungsbedarfe an der Infrastruktur der Transportsysteme sowie der Produktionslinien und -inseln mit räumlichen Veränderungen von Übergabestationen zum Transportsystem induzieren. Strukturelle Änderungen bei Linien oder Inseln vermögen die gleichen Konsequenzen nach sich zu ziehen. Modifikationen des Funktionsspektrums einer Linie oder Insel führen zu einer Abänderung des herstellbaren Produktspektrums, wodurch sich Stoffflüsse auch auf der Ebene der Produktionsanlage deutlich wandeln können. Verursacht wird dieser Effekt durch eventuell benötigte Vorerzeugnisse, da auch hier Anpassungsbedarfe entstehen, so dass Anpassungskaskaden möglich sind. Die Flusssteuerungen sind deshalb in diesen Fällen entsprechend anzugleichen. Veränderungen in Art und Anzahl der Transporteinheiten sind ebenso denkbar. Es wird jedoch angenommen, dass die Modellschnittstellen zwischen Leitstand $\leftrightarrow$ Transportauftragsverwaltung $\leftrightarrow$ Transporteinheiten entsprechend gestaltet sind, so dass an dieser Stelle keine manuellen Adaptionen vorgenommen werden müssen.

4.1.2 Hauptebene 2 – Lager, Produktionslinien und -inseln, Transportsysteme

Lager bestehen aus Lagerplätzen, Lagervorgänge ausführenden Ressourcen (Bediengeräte, Lageristen) und einer Lagerverwaltung zur Koordinierung von Ein- und Auslagerungen. Aus Sicht der Wandelbarkeit sind Änderungen der zu lagernden Stofftypen sowie der Lagerkapazitäten zu erwarten.

Lager werden allerdings nicht für einzelne Typen sondern für Stoffgruppen unter Einhaltung gruppenspezifischer Randbedingungen konzipiert. Beispiele sind Abmaß- und Gewichtsspektrum, Grenzen für Luftfeuchtigkeit und Temperatur, Schutz vor Korrosion, Erschütterungen oder elektrostatischen Entladungen. Veränderungen des herzustellenden Produktspektrums ziehen deshalb kaum strukturelle Anpassungsbedarfe nach sich. Stattdessen sind dauerhafte Veränderungen der Lagermengen eher von Bedeutung. In jedem Fall werden jedoch Anpassungen der Lagerverwaltung notwendig, da Veränderungen von Gegenstandstypen und -mengen organisatorisch berücksichtigt werden müssen.

Linien und Inseln bestehen aus Verbünden von Eingangs-, Zwischen- und Ausgangslagern, Zellen sowie Produktionseinrichtungen. Bei Linien sind diese Bausteine miteinander über Fördereinrichtungen verkettet, bei Inseln über Transporteinrichtungen lose gekoppelt, so dass Stoffflüsse möglich sind. Beide verfügen über eine Auftragsverwaltung, die anstehende Aufträge kontrolliert in das System einlastet. Aus Sicht der Wandelbarkeit sind zum einen das Hinzufügen und Entfernen von Produktionseinrichtungen, Zellen bzw. Fördereinrichtungen als Reaktion auf ein gewandeltes Produktspektrum oder dauerhaft geänderte Produktmixe und zum anderen das Ersetzen zur Änderung des Automatisierungsgrads, der Prozesstechnik oder der Leistung möglich. Diese können Anpassungen an der Auftragsverwaltung nach sich ziehen, da dieser bekannt sein muss, welche Produkte in welchen Varianten wie erzeugbar sind. Ebenfalls können Veränderungen des Transfers notwendig werden, wenn sich Stoffflüsse ändern. Dann ist auch die Flusssteuerung entsprechend anzupassen.

Aufgabe der *Transportsysteme* ist der Transport von Gegenständen von einem Start- zu einem Zielort. Sie verbinden Produktionslinien und -inseln untereinander bzw. mit Lagern oder innerhalb von Inseln Lager und Produktionseinrichtungen miteinander. Transportaufträge werden von einer Transportverwaltung koordiniert. Diese bildet mit den Ablaufsteuerungen (Kap. 3.4) der Transporteinheiten ein eigenes Teilsystem. Transporteinheiten sind in ihrer Art und Anzahl variabel, so dass die Auftragsverwaltung entsprechend anzupassen ist. Zu den Transporteinrichtungen zählen hier neben Geräten auch Transporteure.

4.1.3 Hauptebene 3 – (Sub-)Zellen

Zellen gruppieren örtlich mehrere *Produktionseinrichtungen* unter optionalem Einsatz von Transfereinrichtungen. Eine Motivation kann die Schaffung eines Elements des Baukastensystems sein, womit der Entwicklungsaufwand auf mehrere Anlagen bzw. Linien oder Inseln verteilt werden kann. Ein zusätzlicher Grund kann das Erzeugen räumlich eng zusammengefasster *Produktionseinrichtungen* zur sequenziellen Durchführung von Arbeitsschritten in getrennten Arbeitsräumen, unter Minimierung von Transferaufwendungen, sein. Auch eine zeitliche oder ereignisdiskrete Koordination von Produktionseinrichtungen in überlappenden Arbeitsräumen kann eine Ursache für die Bildung einer Zelle sein. Mischformen mit mehreren Arbeitsräumen sind möglich, in denen eine oder mehrere Produktionseinrichtungen kooperativ agieren.

Zellen können Subzellen enthalten, die möglicherweise ebenfalls Bestandteile des Baukastensystems sind. Prinzipiell können so beliebige Schachtelungstiefen erreicht werden. Zellen sind somit Verbünde, die Verbünde (Subzellen), Zwillingselemente (Gesteuerte Produktions- oder Transfereinrichtungen), physische (z.B. Grundgestelle) aber auch Modellelemente (z.B. Koordination) enthalten.

Änderungen an Zellen können in dem Hinzufügen, Entfernen oder Austauschen von Produktionseinrichtungen bestehen. Dies kann Anpassungsaufwendungen bei der Koordinierung der Produktionseinrichtungen, ggf. bei der Fördersystemstruktur sowie bei der Flusssteuerung verursachen.

4.1.4 Hauptebene 4 – Produktions- und Transfereinrichtungen

Produktionseinrichtungen dienen der Durchführung von Prozessen oder Prozessschritten. Sie sind entweder Elemente oder Verbünde, je nachdem, ob ein Umbau möglich ist. Sie bestehen aus *Modulen*, die spezifische Teilfunktionen übernehmen. Diese Teilfunktionen können ereignisdiskret gesteuert oder zeitkontinuierlich bzw. zeitdiskret geregelt werden, was zentral wie dezentral möglich ist. Die Einrichtung verfügt entweder über eine einzelne übergeordnete Steuereinheit oder über mehrere, die auf einzelne *Module* verteilt sind. Auch sind Mischformen möglich, etwa derart, dass die übergeordnete Einheit die Ablauf- und untergeordnete Einheiten die Ausführungsteuerung (Kap. 3.4) übernehmen.

Aus Sicht der Wandlungsfähigkeit können Module entfernt, hinzugefügt oder ausgetauscht werden, mit möglichen Anpassungsbedarfen an Koordinierung oder Regelung der Elemente.

Transfereinrichtungen sind analog zu Produktionseinrichtungen, jedoch meist einfach aufgebaut. Änderungen sind eher selten zu erwarten, da ein Wandel des Produktspektrums entweder keine Auswirkungen auf Transfereinrichtungen hat oder einen Austausch dieser Systeme induziert. Dennoch können auch Transfereinrichtungen wandelbar gestaltet werden. Wahrscheinlicher ist jedoch bei einem Wandel des Produktspektrums, dass eine Anpassung der Transferhilfsmittel (TUL) erforderlich wird, etwa wenn erzeugnisspezifische Formnester benötigt werden.

4.1.5 Hauptebene 5 – (Sub-)Module

Module erfüllen Teilaufgaben zur Durchführung von Prozessen oder Prozessschritten. Sie werden entweder durch eine übergeordnete Steuereinheit kontrolliert oder verfügen selbst über Rechentechnik auf der ihr Steuerungsmodell ausgeführt wird. Module können aus Submodulen aufgebaut oder untrennbare Elemente sein. Sie sind entweder physische Struktur- (z.B. Gehäuse) oder Funktionselemente (z.B. Spannvorrichtung, manuelle Verrichtungen), gesteuerte Zwillingselemente oder Verbünde aus Submodulen. Auch ein Modellelement oder ein Verbund aus Modellelementen können als Module organisiert sein, etwa zur Koordination von Submodulen, zur Betriebsdatenerfassung und -bereitstellung oder dem Filtern von Informationen.

Die Komplexität des Aufbaus von Modulen kann variieren. Die einfachsten Module sind Verbünde, Elemente oder Zwillingselemente aus Basismodulen und entsprechen damit den Elementesystemen (Kap. 3.1.2). Kombinierte Sensoren und Aktoren zählen hierzu. Motoren mit Vorgabe und Abfrage der Drehrichtung oder Stopper mit Positionsabfrage sind zwei Beispiele. Oft verfügen diese Misch-Sensor-Aktor-Module über zusätzliche Modellelemente zur Erweiterung der Funktionalität.

Module können gewandelt werden, indem Submodule entfernt, hinzugefügt oder ausgetauscht werden. Konsequenzen hat dies für die Steuerungsmodelle, da sie entsprechend zu adaptieren sind.

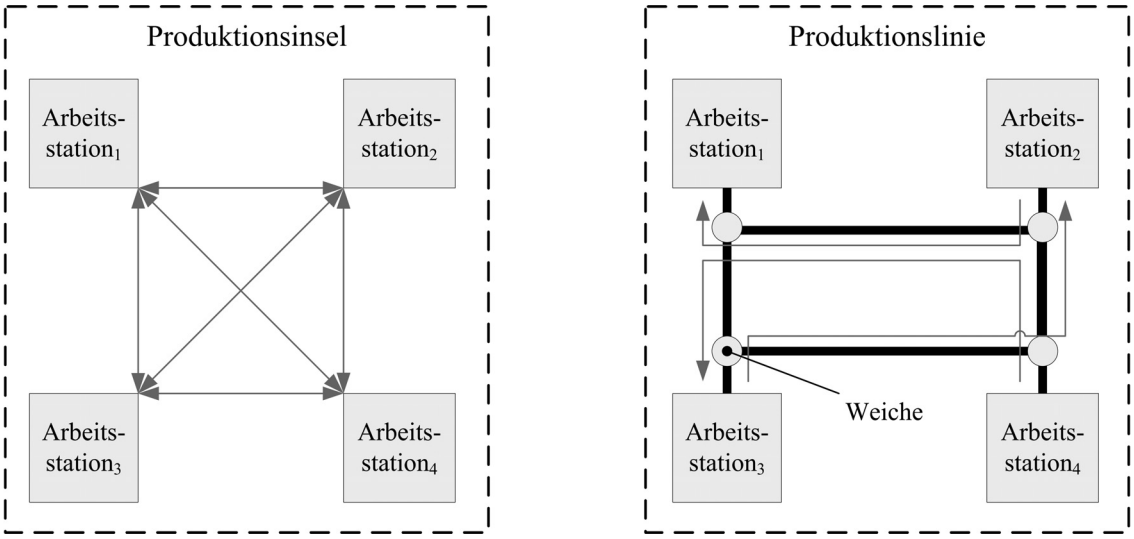


Abbildung 4.2: Links – Produktionsinsel mit vermaschter Topologie. Rechts – Produktionslinie mit fester Infrastruktur.

4.1.6 Hauptebene 6 – Basismodule

Basismodule entsprechen den Basiselementen der Systemtheorie (Kap. 3.5) und stellen die kleinsten, untrennbaren Einheiten dar. Sie existieren als physische Struktur oder Funktionsmodule, als Modell-Struktur- oder Modell-Funktionsmodule sowie als Zwillingsbasismodule.

Basismodule können im Zuge einer Wandlung ausgetauscht werden, wobei dies weitere Anpassungen hervorrufen kann.

4.2 Strukturierte und unstrukturierte Bereiche

Aus der systemtheoretischen Betrachtung (Kap. 3.1.2) geht hervor, dass Systeme einen Strukturteil besitzen. Dies gilt somit auch für Produktionsanlagen. Allerdings gibt es bezogen auf Stoffflüsse Bereiche, die sich einer Strukturierung entziehen. Bei der Gestaltung der Systemlandschaft muss dies berücksichtigt werden.

In Produktionslinien bilden Produktionseinrichtungen mit Fördersegmenten und Verzweigungen eine feste Infrastruktur, während bei Produktionsinseln eine lose Kopplung vorherrscht und der Transport Punkt-zu-Punkt durch Transporteure oder Fahrzeuge erfolgt. Dadurch liegt eine Vollvermaschung zugrunde, aus der keine Struktur mehr erkennbar ist. Derselbe Sachverhalt liegt auch aufgrund von Punkt-zu-Punkt-Verbindungen zwischen Linien, Inseln und Lagern auf der Ebene der Produktionsanlage vor. Abbildung 4.2 zeigt dies im Vergleich von Linien und Inseln.

Grundsätzlich resultieren aus dem Einsatz von Fördersystemen *strukturierte* und aus dem Einsatz von Transportsystemen *unstrukturierte Bereiche*. Dies hat Konsequenzen für die in Kapitel 8 vorgestellten Verfahren zur Selbstkonfiguration und Selbstorganisation.

4.3 Dezentrale Steuerungstechnik

Rechentechnik

Für eine maximale Wandelbarkeit ist bei der Systemlandschaft eine möglichst große Dezentralisierung der Softwaresysteme unter hohem Einsatz von Rechentechnik vorgesehen. Sie werden derart aufgeteilt, dass ein Softwaresystem exakt eine komponentenspezifische Aufgabe erfüllt und auf einem eigenen Rechner ausgeführt wird. Dieser ist mit der Komponente physisch gekoppelt, so dass ein Verbund aus mechanischem Bauteil, Rechnerhardware und Software entsteht, der in seiner Ganzheit ausgetauscht werden kann. Das Softwaresystem kontrolliert keinesfalls andere Komponenten auf gleicher oder Superebene. Dieser Ansatz führt neben einer verbesserten Skalierbarkeit auch zu einer Steigerung der Autonomie der Komponenten und zu einer Verringerung der Gesamtausfallwahrscheinlichkeit der Produktionsanlage. Das Konzept sieht die Möglichkeit vor, Rechentechnik bedarfsgerecht auf allen Ebenen einzusetzen.

Im Gegensatz zu der in der Praxis üblichen Heterogenität der Rechentechnik, sieht das Konzept Homogenität vor. Das bedeutet, dass nur eine Rechnerarchitektur verwendet wird. Damit wird das Ziel verfolgt, Mehrfachanpassungen für verschiedene Plattformen bei Anlagenmodifikation zu vermeiden und so Entwicklungsaufwendungen zu reduzieren. Zudem ist dies für die geforderte Trennung zwischen Steuerungsmodellen und Steuerungstechnik (Kap. 3.4) von Vorteil, da so nur für eine Plattform entwickelt werden muss.

Kommunikationstechnik

Für die Verwendung von Kommunikationstechnik ist in der wandelbaren Systemlandschaft vorgesehen, dass *alle* Rechner auf *allen* Hierarchieebenen kommunikationstechnisch direkt oder indirekt miteinander verknüpft sind, so dass Kommunikation zwischen *allen* Softwaresystemen stattfinden kann.

In strukturierten Bereichen wird eine kabelgebundene Vernetzung favorisiert, da damit eine Abbildung der Strukturen auf Kommunikationsnetze möglich ist, die dann automatisiert informationstechnisch erfasst werden können. Diese Informationen dienen als Grundlage für selbstorganisierende logistische Verfahren (Kap. 8). Eine kabelgebundene Vernetzung bietet damit ein erhebliches Potenzial zu Unterstützung der Wandlungsfähigkeit, über das nach derzeitigem Stand der Technik funkgestützte Kommunikationstechnik nicht verfügt. Auf Ebene der Produktionslinien erfolgt dies bspw. dadurch, dass Netzwerkkabel entlang von Fördersegmenten verlegt und mit Steuerrechnern von Produktionseinrichtungen und Förderkreuzungen bzw. -weichen gekoppelt werden.

In unstrukturierten Bereichen ist die Verwendung von funkbasierter Kommunikationstechnik vorgesehen, da in diesen Bereichen *mobile Einheiten* vorkommen. Deshalb ist an dieser Stelle Funkübertragung kabelgebundener Technik eindeutig überlegen. Teilnehmer lassen sich hier nicht strukturell erfassen sondern lediglich identifizieren.

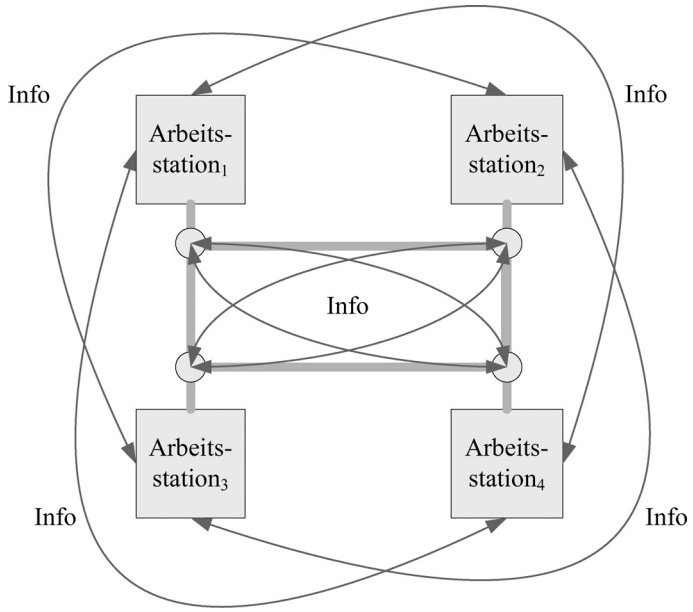


Abbildung 4.3: Übergreifende Steuerungsstruktur. Modellelemente interagieren über horizontale und vertikale (nicht dargestellte) Grenzen hinweg.

4.4 Informationelle Vernetzung

Für die wandlungsfähige Systemlandschaft ist eine *homogene informationelle Vernetzung* aller Softwaresysteme derart vorgesehen, dass prinzipiell *alle* Bestandteile von Steuerungsmodellen einer Produktionsanlage miteinander Informationen austauschen können. Die Philosophie ist hierbei, Information möglichst ohne Zwischenspeicherung oder -verarbeitung vom Erzeuger direkt zum Verbraucher übermitteln (Abb. 4.3) zu können. Dies reduziert die Systemkomplexität durch Einsparung zusätzlicher Software, wodurch sich die Wandlungsfähigkeit erhöht.

Um dies zu erreichen, sind vier Punkte von Bedeutung. Erstens müssen alle Modellbausteine in einer Produktionsanlage eindeutig adressierbar sein, um eine eindeutige Ansprechbarkeit zu ermöglichen. Dies erfordert ein anlagenübergreifendes *Adressierungsschema*. Zweitens besteht die Notwendigkeit der Vergabe einer eindeutigen *Identität* an Modellbausteine und damit im Falle von Zwillingsbausteinen auch an ihre physischen Pendants. Dies dient zur stetigen Identifikation und damit Unterscheidung der Komponenten, unabhängig von ihren aktuellen Positionen innerhalb der Anlagenstruktur. Notwendig ist dies, da zum einen die Positionsänderung einer Komponente möglich ist, indem etwa eine Produktionseinrichtung von einer Produktionslinie in eine andere oder ein Modul von einem Gerät an ein anderes versetzt wird. Zum anderen können mehrere Komponenten gleichen Typs verwendet werden. Da die systemtheoretischen Schlussfolgerungen (Kap. 3.4) eine Trennung von Steuerungsmodellen und Steuerungstechnik vorsehen, muss ein Steuerungsmodell grundsätzlich zentral auf einem oder dezentral verteilt auf mehreren Rechnern ablauffähig sein. Da die Modelle aus interagierenden Steuerungsbausteinen bestehen (Kap. 3.5), ist aus Sicht der Softwareentwicklung von einer eventuell zwischen zwei Bausteinen liegenden Kommunikation zu abstrahieren. Dies trägt zu einer Verminderung der Softwarekomplexität und damit zu niedrigen Entwicklungsaufwendungen bei. Deshalb ist drittens eine Abstraktion von der Kommunikationstechnik erforderlich, um die gleiche

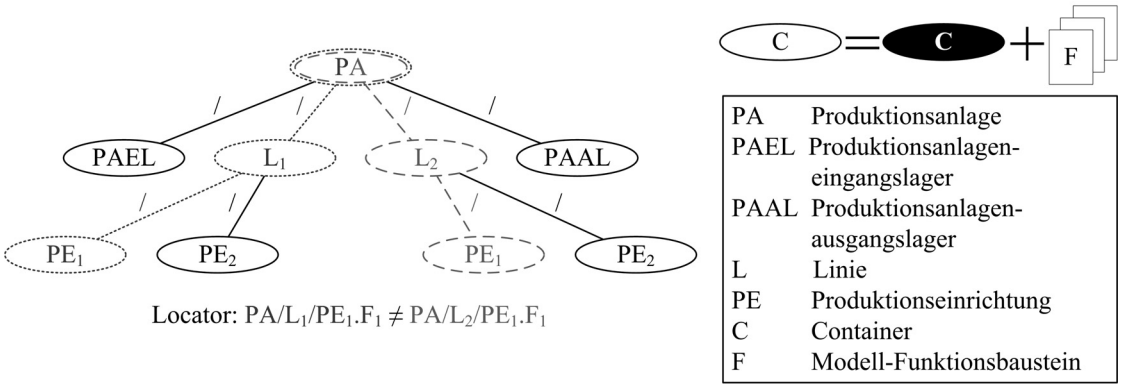


Abbildung 4.4: Adressierungsschema auf Basis der Systemhierarchie. Aus Gründen der Übersichtlichkeit sind nur Knoten, jedoch keine Funktionsbausteine im Baum dargestellt. Jeder Knoten enthält einen oder mehrere Funktionsbausteine, wie oben rechts angedeutet.

Steuerungssoftware in unterschiedlichen Konstellationen der Steuerungstechnik unverändert zur Ausführung bringen zu können. Hierzu sieht das Konzept den Einsatz einer *Kommunikations-Middleware* [5, 22] vor. Viertens sind *Interaktionsformen* zwischen Bausteinen der Steuerungsmodelle festzulegen, damit darauf aufbauend Interaktion konzipiert werden kann.

4.4.1 Übergreifendes Adressierungsschema

Eine Adressierung der einzelnen Modellbausteine lässt sich durch Bildung einer Hierarchie bewerkstelligen. Hierzu wird festgelegt, dass jedes Modell-Basiselement Bestandteil eines Knotens mit Containerfunktionalität ist (Kap. 3.5). Darüber hinaus kann ein Knoten mehrere Funktionsbausteine aber auch Subknoten aufnehmen. Jeder Knoten steht mit seinem Super- und seinen Subknoten in referenzieller Beziehung, so dass sich eine baumartige Vernetzung ergibt – die gewünschte hierarchische Beziehung. Jedes Modell-Basiselement wird benamt, wobei als Regel gilt, dass *keine zwei Elemente eines Knotens den gleichen Namen tragen*. Jeder Modellbaustein ist adressierbar, indem die Namen aller Knoten entlang des Pfads durch den Baum, ausgehend von der Produktionsanlage bis zum Zielknoten, über ein spezifisches Trennzeichen (z.B. '/') konkateniert werden. Optional kann der Name eines Funktionsbausteins über ein weiteres Trennzeichen (z.B. '.') angehängt werden. Durch die beiden verschiedenen Trennzeichen ist eindeutig unterscheidbar, ob das adressierte Element ein Knoten oder ein Funktionsbaustein ist (Abb. 4.4). Die durch die Konkatenation entstehende Zeichenkette wird *Locator* genannt.

Für die Adressierung der Modellbausteine sieht das Schema drei verschiedene Möglichkeiten vor:

Absolute Adressierung ab der Basis: Sie dient dazu, ausgehend von der festen Basis Produktionsanlage, Bausteine anzusprechen. Allerdings können hieraus Probleme bei Wandlung entstehen, da durch eine Anlagenmodifikation Elemente entfernt werden und neue hinzukommen können. Dadurch können sich Pfade ändern, so dass Basiselemente neue Adressen erhalten und deshalb unter der Ursprünglichen nicht mehr erreichbar sind.

Absolute Adressierung ab der Wurzel eines Subbaums: Eine Möglichkeit das Problem der absoluten Adressierung ab der Basis zu entschärfen ist die absolute Adressierung ab der Wurzel eines Subbaums.

Hierbei bieten sich Subbäume an, die jeweils auf einem eigenen Steuerrechner zur Ausführung gebracht werden. Ein solcher Subbaum in Verbindung mit einem Rechner wird *Entität* genannt (Kap. 5.7). Gerade auf der Ebene der Produktions- und Transfereinrichtungen (Hauptebene vier) sowie der Module (Hauptebene fünf) sind Entitäten zu erwarten. Der Aufbau einer Entität wird als (relativ) konstant angesehen, so dass hier diese Art der Adressierung von Vorteil ist.

Relative Adressierung: Die relative Adressierung ermöglicht es schließlich, Standardzellen oder -module in unterschiedlichen Kontexten anzusprechen, ohne ihre genaue Lage in der Anlagenstruktur zu kennen. Allerdings dürfen sich die relative Positionen der Bausteine zueinander nicht verändern. Die relative Adressierung wird durch die Navigation im Baum um eine Stufe nach oben bzw. nach unten ermöglicht. Für die Aufwärtsnavigation beginnt der Pfad mit einen Aufwärtsbezeichner (z.B. '..'), für die Abwärtsnavigation mit dem Namen eines direkten Subknotens.

4.4.2 Identifikation von Modellelementen

Das Adressierungsschema verbietet nicht, im Gesamtssystem *Produktionsanlage* mehrere gleich benamte Modellbausteine einzusetzen. Um diese jedoch eindeutig unterscheiden zu können, verfügt jeder über eine weltweit eindeutige Identität. Es stellt sich die Aufgabe der Erzeugung eindeutiger Identitäten vor dem Hintergrund weltweit verschiedener Bausteinhersteller. In der Praxis sind als Vorbilder sowohl zentrale wie dezentrale Verfahren zu finden. Ein Beispiel für den ersten Ansatz ist die zentrale Vergabe von MAC-Adressblöcken für Ethernet-Netzwerkarten durch die IEEE [47]. Verfahren zur dezentralen Erzeugung weltweit eindeutiger Identitäten finden sich in der Empfehlung X.667 der ITU-T [107]. Dem Grundgedanken einer maximalen Dezentralisierung folgend, wird der zweite Ansatz gewählt. Dadurch wird eine (weltweite) zentrale Koordinierungsinstanz vermieden.

4.4.3 Kommunikations-Middleware – Abstraktion von Kommunikation

Zur Abstraktion von der Kommunikation sind aus der Informatik Anforderungen bekannt, die auch auf die wandelbare Systemlandschaft zutreffen. Die Wichtigsten sind in Tabelle 4.1 zusammengefasst und kurz erläutert. Näheres findet sich in [103, 121].

Eine wichtige Voraussetzung für das erfolgreiche Zusammenwirken von Modellbausteinen über Rechnergrenzen hinweg – sowohl horizontal auf einer Ebene als auch vertikal über mehrere Ebenen – ist die Verwendung homogener Kommunikationsprotokolle, die von allen Teilnehmern verstanden werden. Durch Einführung einer Kommunikations-Middleware, als quasi achte Schicht des ISO-OSI-Modells [130, 102], kann dies erreicht werden. Sie abstrahiert auf Entwicklungsseite weitgehend von Details der Kommunikation, wodurch sie die Softwareentwicklung vereinfacht. [22, 71]

Die zwei Arten *prozedurorientierte* und *objektorientierte Middleware* werden unterschieden. Beispiele für prozedurorientierte Middleware sind RPC [6] und SOAP [123]. Beispiele für objektbasierte bzw. -orientierte Middleware sind Ice [128], CORBA [106], DCOM [116], .NET [67] und XML Webservices [124]. Im Bereich der Prozessautomatisierung existiert OPC (UA) [73].

Transparenzbegriffe	Erläuterungen
Ortstransparenz	Der Ausführungsort eines Modellbausteins ist unbekannt.
Zugriffstransparenz	Der Zugriff auf Modellbausteine erfolgt stets gleich, egal ob lokal oder entfernt.
Nebenläufigkeitstransparenz	Mehrere parallele Ressourcenzugriffe werden derart synchronisiert, dass kein Informationsverlust oder Schaden entsteht.
Skalierungstransparenz	Änderungen am System erfolgen möglichst ohne Störung der restlichen Komponenten.
Replikationstransparenz	Zur Steigerung der Leistung oder Ausfallsicherheit können Komponenten mehrfach existieren. Die Auswahl einer Komponente erfolgt durch das System.
Fragmentierungstransparenz	Teile eines Modellelements oder -verbunds können an unterschiedlichen Orten ausgeführt werden.
Fehler- & Ausfalltransparenz	Teilausfälle werden automatisch kompensiert, so dass das System mit verminderter Leistung operiert.
Sprachtransparenz	Die Kommunikation zwischen den Komponenten ist unabhängig von der jeweils verwendeten Programmiersprache.

Tabelle 4.1: Auswahl von Transparenzbegriffen verteilter Systeme – an die wandelbare Systemlandschaft adaptiert. Nach [121].

4.4.4 Interaktionsformen

Zur Interaktion zwischen Modellbausteinen sind für das Konzept einer wandelbaren Systemlandschaft drei Möglichkeiten vorgesehen, von denen Abbildung 4.5 zwei verdeutlicht. Ein Austausch von Information zwischen zwei Elementen ermöglicht das *Request-Reply-Prinzip*. Ein Modellbaustein stellt eine Anfrage und erhält eine Antwort. Alternativ kann als zweite Möglichkeit ein Baustein einen anderen zielgerichtet informieren. Das dritte Prinzip ist das sog. *Publisher-Subscribe-Verfahren*. Hier registrieren sich Interessenten bei einem Modellbaustein für Nachrichten. Tritt eine Situation ein, aus der die Erzeugung einer Nachricht hervorgeht, wird diese an alle Interessenten weitergeleitet. Der erste Ansatz erfolgt *synchron*, d.h. auf eine Anfrage erfolgt unmittelbar eine Antwort. Dagegen verläuft der dritte Ansatz *asynchron*, da eine Benachrichtigung mehrfach und zu beliebigen Zeitpunkten erfolgen kann.

Über das Adressierungsschema, die Kommunikations-Middleware und die Interaktionsformen ist es jedem Modellbaustein möglich, mit jedem anderen der Produktionsanlage lokal oder rechnerübergreifend zu interagieren.

4.5 Wandelbare Steuerungssoftwaresysteme

Zur Unterstützung der Wandelbarkeit sieht das Konzept Steuerungssoftware vor, die sich an neue Situationen möglichst selbstständig adaptiert, so dass keine manuellen Entwicklungen oder Konfigu-

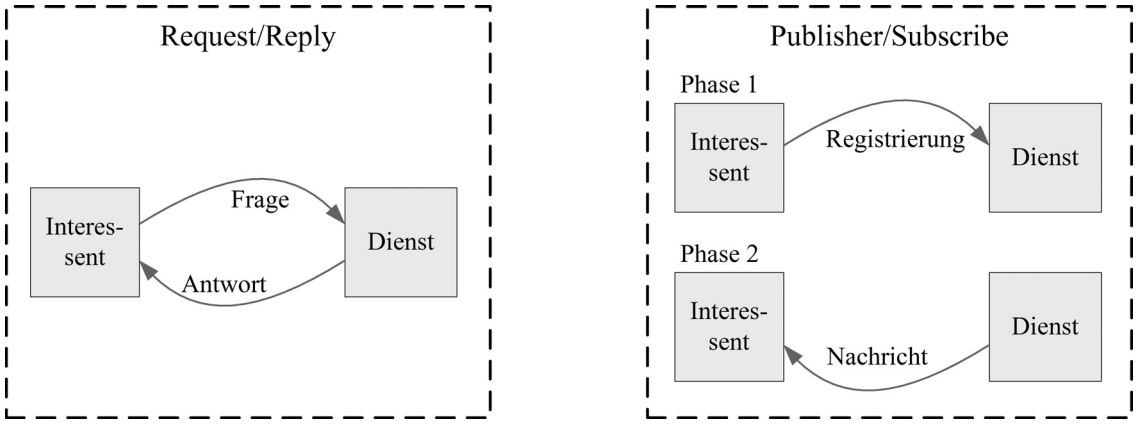


Abbildung 4.5: Links – Request-Reply-Verfahren zum synchronen Informationsaustausch. Rechts – Publisher-Subscriber-Verfahren zum asynchronen Informationsaustausch.

rationen erforderlich werden, bzw. aufwandsarm anpassbar ist. Nach Kapitel 2.2 werden *Regelung, Steuerung* und *Koordination* unterschieden.

4.5.1 Steuerung und Regelung

Steuerung und Regelung kann auf den Hauptebenen *eins* bis *fünf* stattfinden. Auf den ersten drei Hauptebenen erfolgen Steuerungen *logistischer Abläufe*, losgelöst von physischen Bausteinen. Auf Hauptebene vier und fünf finden Steuerungen und Regelungen von *Prozessen* unter Einbeziehung physischer Bausteine statt.

Steuerung logistischer Abläufe

Die *Steuerung logistischer Abläufe* erfolgt durch ein Zusammenwirken mehrerer Steuerungssoftwaresysteme auf allen drei Hauptebenen. Softwaresysteme einer höheren Ebene stehen mit Softwaresystemen niedrigerer Ebenen über Informationsschnittstellen und dem Adressierungsschema in Verbindung.

Logistische Abläufe betreffen hauptsächlich die Organisation von Stoffflüssen unter Einbeziehung der zu erzeugenden Produktmixe, dem aktuellen Anlagenzustand und ggf. Prognosen über erwartete zukünftige Zustände. Der Ermittlung von Zustand und Prognosen dienen Betriebsdaten und -historien. Beide stellen Rückkopplungen eines Regelkreises dar.

Wandlungen der Produktionsanlage können zu Strukturmodifikationen führen, woraus Anpassungsbedarfe resultieren. Die logistischen Abläufe werden durch eine Vielzahl von Steuerrechnern und Steuermodellen realisiert, die Bestandteile der verschiedenen Ebenen sind. Sie müssen für die Umsetzung der Adaptionen entsprechend koordiniert werden bzw. sich selbst koordinieren. Deshalb sind für die Systemlandschaft geeignete Verfahren vorgesehen, um Selbstorganisation und -konfiguration zu ermöglichen.

In Kapitel 8 werden hierzu mehrere Verfahren vorgestellt, die im Rahmen der vorliegenden Arbeit entwickelt wurden. Sie umfassen die *Selbsterkennung von Entitäten* (Kap. 5.7) und ihren *strukturellen Verknüpfungen*, die *Selbstkonfiguration des Kommunikationsnetzwerks*, die *Selbstorganisation von Stoffflüssen*, die *Selbstorganisation auf Ebene der Produktionslinien und -inseln* sowie die *Selbstorganisation auf Ebene der Produktionsanlage*.

Prozessregelung und Prozesssteuerung

Aufgabe der *Prozessregelung* ist das exakte Dirigieren eines oder mehrerer physischer Funktionsbausteine zur Durchführung einzelner (elementarer) Prozessschritte, unter Einhaltung spezifischer Produktionstoleranzen. Jeder physische Baustein als Regelstrecke benötigt zunächst einen eigenen Regler – das Steuerungsmodell bzw. Regelungsmodell –, der Stellwerte durch den zyklischen Vergleich von Ist- mit Sollwerten berechnet.

Physischer Baustein und Regelungsmodell bilden ein Zwillingselement. Agieren mehrere Funktionsbausteine konzertiert zur Durchführung eines Schritts, sind geschichtete (kaskadierte) Regelungsmodelle vorgesehen, die sukzessive auf einfacheren Regelungsmodellen aufsetzen. Regler der verschiedenen Schichten tauschen relevante Informationen über entsprechende Schnittstellen aus. Eine Wandlung kann derart erfolgen, dass Funktionsbausteine ausgetauscht werden, wobei dies immer einen Wechsel des dazugehörigen Regelungsmodells zur Folge hat.

Sofern die Entwicklung parametrierbarer Regelungsmodelle möglich ist, kann dadurch auch lediglich eine Umparametrierung erforderlich werden. Schnittstellen und Modelle sind dann geeignet ausgelegt, wenn im Fall eines geschichteten Regelungsmodells der Austausch eines Funktionsbausteins, die übergeordnete Schicht nicht invalidiert. Es ist davon auszugehen, dass nicht für jedes komplexe Regelungsproblem ein geschichtetes Modell erstellt werden kann. In diesem Fall entsteht ein monolithisches, nicht wandelbares Regelungssystem.

Zur Verdeutlichung soll als Beispiel ein einfaches Achssystem dienen (Abb. 4.6), bestehend aus zwei Achsen als Ausführungs- und zwei Antriebseinheiten als Informationssysteme. Die Antriebseinheiten beinhalten jeweils ein Achsmodell, das zur Einzelregelung dient. Zur Koordinierung beider Achsen existiert ein übergeordnetes 2-Achsmodell. Wird nun eine Achse getauscht, ist womöglich auch ein Wechsel der Antriebseinheit erforderlich – jedoch ist in jedem Fall das entsprechende Achsmodell zu modifizieren.

Der Einsatz einer *Prozesssteuerung* ist dann vorgesehen, wenn ein physischer Baustein nicht oder nur indirekt an der Einhaltung spezifischer Produktionstoleranzen beteiligt ist und deshalb keine Regelung notwendig ist. Indirekt beteiligt ist eine Steuerung, wenn sie Bestandteil eines komplexeren Regelungssystems ist. Prozesssteuerungen bilden mit den gesteuerten Bausteinen Zwillingselemente. Mehrere Funktionsbausteine können konzertiert an der Durchführung eines Prozessschritts beteiligt sein, was den Einsatz geschichteter Steuerungsmodelle erfordert. Deshalb gelten für die Wandlung die gleichen Anforderungen, wie für die Prozessregelung.

Einfache Beispiele für gesteuerte Bausteine sind Greifer, Stopper, Schrittmotoren oder ungeregelte Motoren.

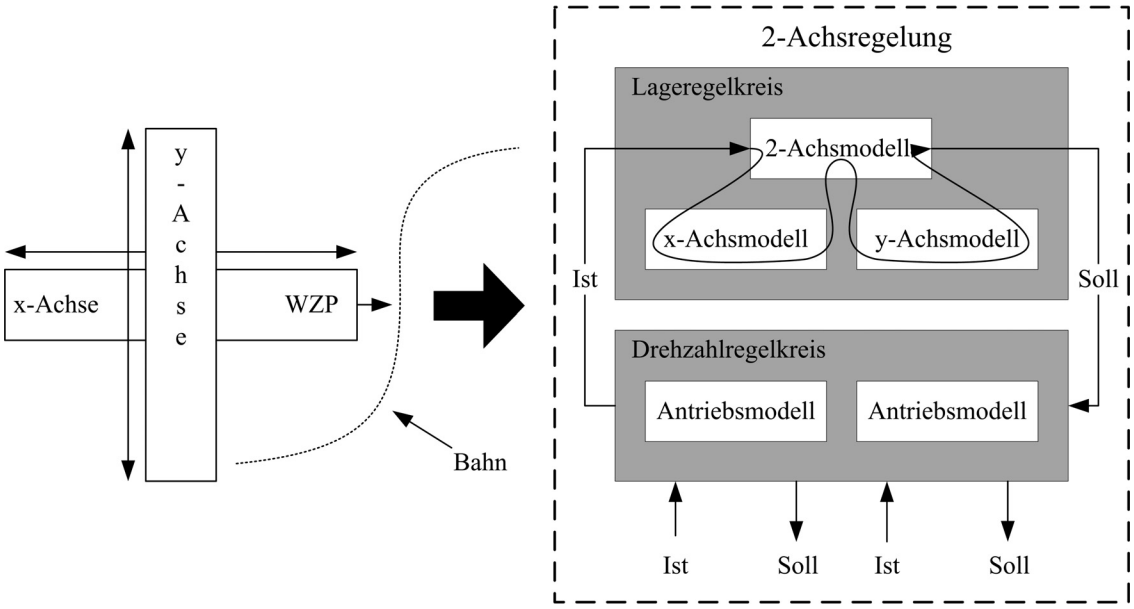


Abbildung 4.6: Beispiel 2-Achsregelung. Beim Achsaustausch mit Typwechsel sind zudem Regler und Modell auszutauschen oder umzukonfigurieren. x = Istwert; Δx = Stellwert; y = Istwert; Δy = Stellwert; WZP = Werkzeugpunkt.

4.5.2 Prozesskoordination

Die *Prozesskoordination* ist die wichtigste Komponente zur Steuerung von Zellen. Sie dient zur ereignisdiskreten, koordinierten Kontrolle von Produktionseinrichtungen – optional in Verbindung mit Transfereinrichtungen – zur Ausführung von Prozessen. Je nach Produkt, Variante, Bearbeitungszustand – und damit des durchzuführenden Prozesses – sind die Einrichtungen in unterschiedlicher Reihenfolge zu parametrieren und aktivieren. Die Parametrierung einer Einrichtung kann zudem abhängig von ihrem Kontext erforderlich sein, d.h. von der Art, Anzahl, Leistung, kinematischem Grundmodell, usw. der benachbarten Einrichtungen.

Die Koordination muss variabel gestaltet sein, weshalb für die Prozesskoordination die Verarbeitung austauschbarer *Rezepte* vorgesehen ist. Rezepte sind Sequenzen von Steueranweisungen, die von der Prozesskoordination schrittweise interpretiert und abgearbeitet werden. Sie sind spezifisch für Produkte, deren Variante, ihrem Bearbeitungszustand und der durchzuführenden Tätigkeit.

Die Prozesskoordination selbst ist von Wandlung insofern betroffen, als dass sich bei Austausch von Produktionseinrichtungen oder Modulen das Funktionsspektrum und damit auch das bearbeitbare Produktspektrum ändert. Dies hat Auswirkungen auf die Menge der Rezepte, da diese entweder angepasst oder entfernt werden bzw. neu hinzukommen. Deshalb ist vorgesehen, dass jede Prozesskoordination stets über die aktuellen Rezepte verfügt.

4.6 Selbstadaptierende generische Informationsdienste und allgemeine Informationssysteme

Für die Betriebsfähigkeit der wandelbaren Produktionsanlagen sind Softwaresysteme erforderlich, die wesentliche Grunddienste zur Informationsverwaltung und -bereitstellung implementieren. Zu ihnen zählen die ebenenübergreifende *Betriebsdatenerfassung*, auf Ebene der Produktionsanlage die *Arbeitsplanverwaltung und Rezepteverwaltung* sowie auf Ebene der Produktionslinien und -inseln *Auftragsverwaltungen*.

Zusätzlich ist auf allen Ebenen der Einsatz weiterer Informationssysteme möglich, die optional auch ebenenübergreifend realisiert sein können. Sie werden allerdings an dieser Stelle als branchen- bzw. fallspezifisch angenommen und sind daher nicht näher spezifizierbar. Beispiele finden sich in den Bereichen Mensch-Maschinen-Schnittstellen, Mitarbeiterinformationssysteme, usw.

4.6.1 Betriebsdatenerfassung

Die über alle Ebenen verteilte Betriebsdatenerfassung ist ein dezentrales Gebilde, bestehend aus Informationserzeugern, -speichern und -verbrauchern. Das Konzept sieht eine zentrale Verwaltung vor, die im Zentrum der Datenerfassung steht. Sie verfügt über Schnittstellen zur Informationspeicherung und -abfrage. Sie wird im wandelbaren Umfeld als Konstante angesehen, da sie stets benötigt wird. Nur Informationserzeuger und -verbraucher können in ihrer Art und Anzahl einer Wandlung unterworfen sein. Deshalb sind Schnittstellen und Abläufe derart ausgelegt, dass keine Abhängigkeiten entstehen und der Wechsel von Bausteinen keine Störungen verursacht. Das Konzept gesteht deshalb der Verwaltung nur ein passives Verhalten zu, d.h. jegliche Initiative geht von den beteiligten Komponenten aus, deren Betriebsdaten zu verwalten oder an die Informationen zu übermitteln sind. So ist die Verwaltung auf keine bestimmte Komponente angewiesen und dadurch auch nach Wandlungen funktionsbereit.

Kapitel 9.6 stellt eine generische Betriebsdatenverwaltung mit ihren Schnittstellen zum Informationsaustausch vor.

4.6.2 Rezepte- und Modellverwaltung

Die *Rezeptverwaltung* wird aus zwei Gründen benötigt. Rezepte sind abhängig von dem Produkt, der Variante und dem Bearbeitungszustand. Darüber hinaus sind sie spezifisch für Zellen und der darin enthaltenen Produktionseinrichtungen, d.h. ein identischer Prozess für eine bestimmte Produkt-Varianten-Kombination benötigt an zwei nicht baugleichen Zellen jeweils ein anderes Rezept. Die Rezeptverwaltung ist die zentrale Stelle, von der jede Prozesskoordination benötigte Rezepte abfragen kann. Rezepte werden außerhalb der Produktionsanlage von der Produktionsvorbereitung erstellt und an die Produktionsanlage übermittelt. Dieser Vorgang erfolgt bei Änderungen des Produktspektrums bzw. bei Umbau der Anlage, wenn Zellen verändert, hinzugefügt oder entfernt werden.

Die *Modellverwaltung* dient der Speicherung von Steuerungsmodellen. Sie ist erforderlich, wenn aus Zellen Produktionseinrichtungen entfernt, hinzugefügt oder ausgetauscht werden. Dies gilt ebenso für

Module oder Submodule. Insbesondere in Szenarien, wo die strukturelle Wandelbarkeit zur Erweiterung der elastischen Flexibilität (Kap. 2.4.2) eingesetzt wird, ist eine Modellverwaltung notwendig. Wenn bspw. eine Produktionslinie täglich, wöchentlich oder saisonal an Produktmixe strukturell angepasst wird, ist eine Modellverwaltung essenziell.

Kapitel 9.5 stellt eine generische Rezepte- und Modellverwaltungskomponente vor und beschreibt ihren inneren Aufbau. Zudem wird das Zusammenwirken zwischen Komponenten und Produktionseinrichtungen erörtert.

4.6.3 Arbeitsplanverwaltung

Für Produkte und deren Varianten sind die zur Herstellung durchzuführenden Tätigkeitsschritte mit erforderlichen Prozessparametern und Qualitätsmerkmalen in Arbeitsplänen hinterlegt. Die Reihenfolge der Tätigkeitsschritte bestimmt indirekt den Fluss eines Erzeugnisses durch die Produktion, da nach Durchführung einer Tätigkeit nachzuschlagen ist, welcher Arbeitsschritt als nächstes zu vollziehen ist. Arbeitspläne bilden somit Eingangsgrößen für Prozesse und Flusssteuerungen, weshalb sie an zentraler Stelle verwaltet werden. Jeder Arbeitsplan ist produkt- und variantenspezifisch, wird von der Produktionsvorbereitung erstellt und im Anschluss an die Produktionsanlage übertragen. Die Arbeitsplanverwaltung ist insofern einer Wandlung unterzogen, als dass sich die gespeicherten Arbeitspläne über die Zeit verändern, da bestehende Produkte und Varianten auslaufen oder verändert werden bzw. neue hinzukommen können.

Kapitel 8.3.6 erörtert das Zusammenspiel zwischen Arbeitsplanverwaltung und Produktionslinien und Inseln. Kapitel 9.3 beschreibt eine generische Komponente in ihrem Aufbau.

4.6.4 Auftragsverwaltung für Produktionslinien und -inseln

Das Konzept sieht auf Ebene der Produktionslinien und -inseln Auftragsverwaltungen vor. Jeder Verbund verfügt über eine eigene Verwaltung, deren Aufgabe die kontrollierte Freigabe von Aufträgen ist. Die Auftragsverwaltung ist insofern einer Wandlung unterworfen, als dass sie sich an ändernde Systemstrukturen anpassen muss. Da sich Linien bzw. Inseln und deren Produktionseinrichtungen über die Zeit ändern können, können sich die herstellbaren Produktspektren wandeln. Die Auftragsverwaltung soll sich an neue Situationen möglichst ohne manuelle Eingriffe adaptieren.

Kapitel 8.3.2 beschreibt eine generische Auftragsverwaltung für die wandelbare Systemlandschaft im Kontext der Selbstorganisation. Kapitel 8.4 stellt die Selbstorganisation der Produktionsanlage vor und deren Ankopplung an die Produktionsplanung und -steuerung.

4.6.5 Ausfallsicherheit

Alle zentralen Informationsspeicher der Produktionsanlage können redundant ausgelegt werden, so dass eine Primäreinheit und ein oder mehrere Sekundäreinheiten existieren. Dies soll einem Datenverlust vorbeugen und eine fortwährende Betriebsbereitschaft der Produktionsanlage gewährleisten. Fällt eine Primäreinheit aus, übernimmt eine der Sekundäreinheiten. Drei Schritte sind dazu erforderlich.

Erstens müssen die Informationen der Primäreinheit auf die restlichen Einheiten repliziert werden, wofür Synchronisationsmechanismen vorgesehen sind. Zweitens ist eine Strategie notwendig, mit deren Hilfe nach Ausfall der Primäreinheit eine dedizierte Sekundäreinheit zu einer neuen Primäreinheit wird. Alle beteiligten Steuerungsmodelle sind davon in Kenntnis zu setzen. Drittens wird ein Mechanismus benötigt, der Steuerungsmodellen das Auffinden dieser Einheiten ohne manuelle Eingriffe ermöglicht.

Kapitel 9.1.2 stellt eine generischen Ansatz vor, auf dem alle genannten Verwaltungen aufsetzen.

Kapitel 5

Konzeption einer auf Wandelbarkeit ausgerichteten Softwarearchitektur

Zur Realisierung der vorgestellten wandelbaren Systemlandschaft bedarf es einer durchgängigen Softwarearchitektur, die konsequent dem Systemkonzept aus Kapitel 3.5 folgt. Es wird angenommen, dass alle Softwaresysteme der Systemlandschaft – und damit einer Produktionsanlage – einheitlich mit Bausteinen eines Softwarebaukastensystems realisiert sind, das wiederum die hier vorgestellte Softwarearchitektur umsetzt. Für Entwicklung und Verwendung der Bausteine gilt das Softwareparadigma der *Softwarecomponents*.

Zunächst werden die softwaretechnischen Umsetzungen der in Kapitel 4.4.4 genannten Interaktionsformen und anschließend die Elemente der Architektur vorgestellt. Danach werden mögliche Verhaltensformen für Steuerungsmodell(-Basis)elemente konzipiert und die Möglichkeiten zur Erstellung komplexer Steuerungsmodell-Elemente benannt. Im Rahmen der vorliegenden Arbeit wurden einige grundlegende Basiselemente entworfen, die einerseits exemplarischen Charakter haben, andererseits jedoch bereits den Aufbau einfacher Steuerungsmodelle erlauben. Abschließend wird die *Entität* als Verbund aus Software und Rechnerhardware beschrieben. Alle Entitäten einer Produktionsanlage bilden gemeinsam die „Intelligenz“.

5.1 Softwareparadigma „Componentware“ als Grundlage der Architekturkonzeption

Zur Unterstützung der Wandelbarkeit ist für die Softwarearchitektur die Wahl eines geeigneten Softwareparadigmas von großer Bedeutung. Hierzu kommen vor allem die zwei Ansätze *Objektorientierung* und *Componentware* in Frage. Ziel ist es, die Erzeugung wiederverwendbarer Bausteine mit definierten Schnittstellen und Funktionalität aus softwaretechnischer Sicht zu untermauern.

Objektorientierung

Die Objektorientierung folgt dem Gedanken, den Aufbau und das Zusammenwirken realer Objekte in Software nachzubilden und so eine Effizienzsteigerung bei der Softwareentwicklung zu erzielen [7, 66]. Software wird durch Verschaltung von Klasseninstanzen (Objekten) zu Objektsystemen erzeugt. Klassen kapseln Funktionalität, die über spezifische Schnittstellen zur Interaktion mit anderen Objekten bereitgestellt wird. Wiederverwendbarkeit ist zwar gegeben, bleibt allerdings auf den Entwicklungszeitpunkt beschränkt – ein dynamisches Rekonfigurieren zur Betriebszeit sieht das Konzept

nicht vor. Ursache hierfür ist, dass Klassen auf Quellcodeebene wiederverwendet und durch Vererbung erweitert werden [7]. Eine ungünstige Eigenschaft der Objektsysteme ist, dass Objekte selten autark genug sind, um eigenständig Funktionalität zu erbringen. Stattdessen sind sie in der Regel von vielen anderen abhängig und dadurch zu feingranular vernetzt [37]. Die für die Wandlungsfähigkeit erforderliche Dynamik der Softwaresysteme zur Betriebszeit ist deshalb mit der Objektorientierung allein nicht zu erreichen, weshalb es eines erweiterten Ansatzes bedarf, der über die Objektorientierung hinausgeht.

Componentware

Das Softwareparadigma „Componentware“ zielt auf die Wiederverwendbarkeit von Softwareelementen in unterschiedlichen Kontexten ab [37]. Der Ansatz ist von dem Gedanken getrieben, „Softwareentwicklung als Ingenieursdisziplin im Sinne einer Erstellung komplexer Systeme aus einfacheren vorgefertigten Komponenten [37]“ zu verstehen [65]. Zwar herrscht in der Informatik bis heute kein Konsens über ein abgeschlossenes formales Konzept, allerdings besteht Einigung über wesentliche Eigenschaften von Softwarekomponenten. Die beiden folgenden zwei Zitate zeigen dies:

> „A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. [101],,

> „[A component is] a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard [17]“.

Softwarekomponenten sind abgeschlossene, fertig entwickelte und ausgetestete Softwarebausteine, die dynamisch in verschiedenen Kontexten zur Lösung unterschiedlicher Aufgaben genutzt werden können. Sie werden hierzu idealerweise nicht durch Programmierung, allenfalls durch Konfiguration angepasst [17]. Komponenten verfügen über standardisierte Schnittstellen, die eine Interaktion mit anderen Komponenten ermöglichen [69, 74]. Komponenten sind insofern autark, als dass sie über eine abgeschlossene Funktionalität verfügen und daher selbstständig bestehen können [37, 101].

Ein Vergleich von Softwarekomponenten mit der allgemeinen Systemtheorie (Kap. 3.1) zeigt, dass Softwarekomponenten sich ideal zum Aufbau von Softwaresystemen eignen. Deshalb eignet sich das Paradigma geradezu ideal als Grundlage für die Konzeption der wandelbar ausgerichteten Softwarearchitektur.

5.2 Interaktionsformen

Bevor die Softwarearchitektur dargestellt wird, erfolgt zunächst die Umsetzung der in Kapitel 4.4.4 vorgestellten Interaktionsformen, da die nachfolgenden Erläuterungen darauf zurück greifen. Die Interaktionsformen sind gleichermaßen für lokale als auch für rechnerübergreifende Interaktion vorgesehen.

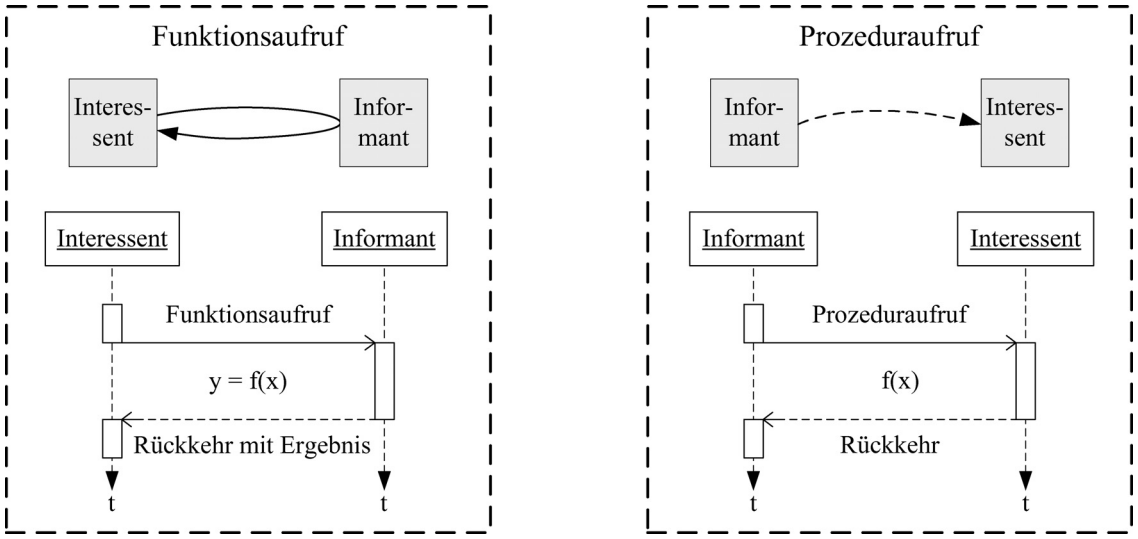


Abbildung 5.1: Vergleich von Funktions- und Prozeduraufruf.

5.2.1 Funktions- und Prozeduraufruf

Das Request-Reply-Prinzip wird in der Architektur durch *Funktionsaufrufe* und das Prinzip der zielgerichteten Informierung durch *Prozeduraufrufe* erbracht, wie Abbildung 5.1 zeigt. Funktionen und Prozeduren sind Bestandteile aller Programmiersprachen unabhängig der zugrunde liegenden Sprachparadigmen (imperative, objektorientierte, funktionale oder prädikative Programmiersprachen) [14]. Dabei unterscheiden nicht alle Programmiersprachen explizit zwischen Funktionen und Prozeduren, wie bspw. Pascal. Deshalb wird an dieser Stelle festgelegt, dass eine Funktion einen Rückgabewert liefert, eine Prozedur nicht.

Funktionen gelangen zum Einsatz, wenn Informationen abgefragt werden sollen oder Aktionen zu veranlassen sind und dabei eine Rückmeldung erwartet wird. Prozeduren dienen zur Informationsübermittlung oder zur Veranlassung von Aktionen ohne Rückmeldung. Funktionen wie Prozeduren warten ggf. auf Abschluss einer gestarteten Aktion.

5.2.2 Publish & Subscribe

Ziel des Publisher-Subscriber-Mechanismus ist eine rechnerübergreifende Nachrichtenverbreitung eines Informationserzeugers an Verbraucher, wodurch Interessenten über eingetretene Ereignisse informiert werden. Die Nachrichtenverbreitung erfolgt über ein virtuelles, drei-stufiges Publisher-Subscriber-Callback-Netzwerk, das für jeden Publisher des Gesamtsystems aufgebaut wird.

Der Aufbau eines solchen Netzwerks erfolgt in zwei Schritten, wie in Abbildung 5.2 oben links und rechts dargstellt. Um Ereignisse erzeugen zu können, erstellt der Erzeuger zunächst ein Publisherobjekt (a), an dem sich über Rechnergrenzen hinweg Subscriber für die Zustellung von Ereignissen registrieren (b). Durch die Registrierung wird eine virtuelle Verbindung im Netzwerk geschaltet (c) und sobald ein Ereignis erzeugt wird, erfolgt dessen Zustellung über das Kommunikationsnetz an die Subscriber. Damit ist der erste Schritt abgeschlossen. Der Verbraucher registriert sich nicht direkt bei

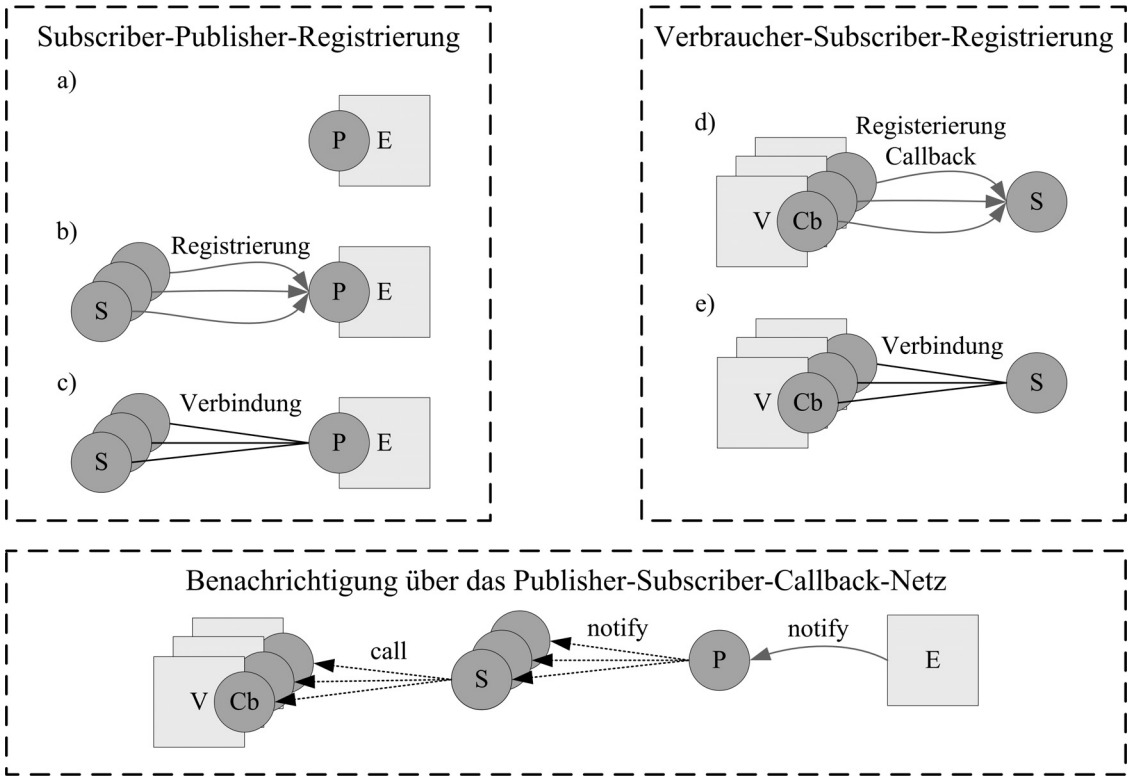


Abbildung 5.2: Oben links – Zwei Phasen der Subscriber-Publisher-Registrierung. Oben rechts – Facetten-Subscriber-Registrierung. Unten – Benachrichtigung über das so entstandene Publisher-Subscriber-Callback-Netz. Jeder Publisher versorgt ein eigenes Netz. Cb = Callback; E = Erzeuger; P = Publisher; S = Subscriber; V = Verbraucher.

dem Publisher, sondern bei einem lokalen Subscriber (d). Durch die Registrierung wird eine virtuelle Verbindung im Netzwerk geschalten (e), wodurch der zweite Schritt abgeschlossen ist. Sobald eine Nachricht am Subscriber ankommt, verteilt er diese an die Verbraucher, indem die registrierten Callbackprozeduren aufgerufen werden. Verbraucher können jederzeit auch von Subscribern und Subscriber von Publishern deregistriert werden, wenn die Zustellung von Ereignissen nicht mehr erforderlich oder vorübergehend unerwünscht ist.

Abbildungen 5.3 und 5.4 zeigen die Schnittstellen der Publisher- und Subscriberobjekte zum Aufbau der Publisher-Subscriber-Netzwerke und zur Übermittlung von Ereignissen. Ein Verbraucher registriert eine Callbackprozedur an einem Subscriber über die Prozedur *Register(callback)* und deregistriert diese über die Prozedur *Unregister(callback)*. Analog gilt dies für (De-)Registrierung von Subscribern an Publishern.

Absenderabhängige Verarbeitung von Ereignissen

Bei der Zustellung von Ereignissen übermittelt der Publisher neben der eigentlichen Information immer auch den eigenen Locator. Dadurch kann im Callback eine Unterscheidung mehrerer Absender erfolgen, so dass spezifische Ereignisbehandlungen möglich sind.

```
PUBLISHER = [
   Attributes = [
        SUBSCRIBER subscribers
   ]
   Register(SUBSCRIBER subscriber) THROWS SameAlreadyExistent
   Unregister(SUBSCRIBER subscriber) THROWS Unknown
   Notify(...)
   GetInfo(): PUBLISHER_INFO
]
```

Abbildung 5.3: Struktur des Publishers.

```
SUBSCRIBER = [
   Attributes = [
        FUNCTION callbacks
   ]
   Register(FUNCTION callback) THROWS SameAlreadyExistent
   Unregister(FUNCTION callback) THROWS Unknown
   Notify(...)
]
```

Abbildung 5.4: Struktur des Subscribers.

Adressierung von Publishern

Ein Problem des Publisher-Subscriber-Mechanismus ist das Auffinden der Publisher, damit eine Registrierung der Subscriber überhaupt erfolgen kann. Es wird durch die Erweiterung des in Kapitel 4.4.1 vorgestellten Adressierungsschemas gelöst, indem ein weiteres Trennzeichen (z.B. „:“) eingeführt wird. Dadurch kann ein Funktionsbaustein mehrere Publisher erzeugen, die über einen eindeutigen Locator verfügen. Hierbei gilt, dass keine zwei Publisher eines Funktionsbausteins gleich benamt sein dürfen.

Subscriberverwaltung

Pro Publisher wird auf einem Rechner maximal nur ein einziger Subscriber erzeugt, um Mehrfachübertragungen zwischen zwei Rechnern zu vermeiden. Dabei stellt sich die Frage, wer diesen Subscriber erzeugt.

Aus Sicht der Wandelbarkeit sind Verbraucher unabhängig voneinander zu gestalten, da andernfalls die Wandlungsfähigkeit sinken würde. Dies bedeutet, dass Verbraucher keine Kenntnis voneinander haben müssen, so dass diese zur Erzeugung der Subscriber ungeeignet sind. Deshalb wird eine zentrale Subscriberverwaltung (*Subscriber Manager*) eingeführt (Abb. 5.5), die Subscriber erzeugen aber auch vernichten kann. Verbraucher registrieren sich nicht mehr direkt an den Subscribern sondern an der Subscriberverwaltung über die Prozedur *Register(locator, callback)* unter Angabe des Publisherlocators und einer Callbackprozedur. Existiert zum Zeitpunkt der Anmeldung kein Subscriber wird er von der Verwaltung erzeugt und die Callbackprozedur registriert. Der Subscriber selbst registriert sich zum Zeitpunkt seiner Erzeugung am Publisher. Werden nacheinander alle Callbackprozeduren

```
SUBSCRIBER_MANAGER  = [
   Attributes = [
        SUBSCRIBER subscribers
   ]
   Register(STRING locator, FUNCTION callback) THROWS SameAlreadyExistent
   Unregister(STRING locator, FUNCTION callback) THROWS Unknown
]
```

Abbildung 5.5: Struktur der Subscriberverwaltung.

deregistriert und muss der Subscriber niemanden mehr benachrichtigen können, deregistriert er sich beim Publisher und vernichtet sich anschließend selbst.

In einem wandelbaren Umfeld kann es allerdings geschehen, dass ein Subscriber etwa durch Systemausfall nicht mehr existent ist, ohne sich vorher ordnungsgemäß abgemeldet zu haben. Der Versuch einen nicht existenten Subscriber zu benachrichtigen bindet Ressourcen über einen längeren Zeitraum (*Timeout*), weshalb ein Publisher nicht mehr erreichbare Subscriber entfernt.

5.3 Architekturelemente

Die Architektur fußt auf den drei Elementen *Knoten, Facette* und *Modul*. Während der Knoten die Aufgabe einer Strukturbildung übernimmt, dienen Facetten und Module der Implementierung von Funktionalität.

5.3.1 Knoten

Die systemtheoretische Analyse von Systemen hat gezeigt, dass die Systemhierarchie einer Baumstruktur gleicht (Kap. 3.1.2). Das daraus abgeleitete Systemkonzept für wandelbare Produktionsanlagen berücksichtigt dieses Merkmal durch die Einführung eines Knotens mit Containerfunktionalität als Modell-Struktur-Basiselement (Kap. 3.5). Seine Aufgabe ist die Aufnahme weiterer Knoten und Modell-Funktions-Basiselemente (*Facetten*). Ein Knoten mit Facetten entspricht einem Modell-Funktions-Element des Systemkonzepts. Alle Knoten bilden miteinander durch ihre hierarchische Verknüpfung als Baumstruktur das Gerüst des gesamten Softwareverbunds der Produktionsanlage. Jeder Knoten ist von allen anderen über das Adressierungsschema erreichbar, wodurch die geforderte informationelle Vernetzung gegeben ist (Kap. 4.4).

Struktureller Aufbau

Der strukturelle Aufbau des Knotens ist in Abbildung 5.8 dargestellt. Er verfügt über *Attribute* und *Schnittstellen* zum Informationsaustausch.

Als Attribute weist der Knoten einen Namen (*name*), einen Locator (*locator*) zur eindeutigen Referenzierbarkeit und eine weltweit eindeutige Identität (*id*) auf. Zur strukturellen Verknüpfung hält er

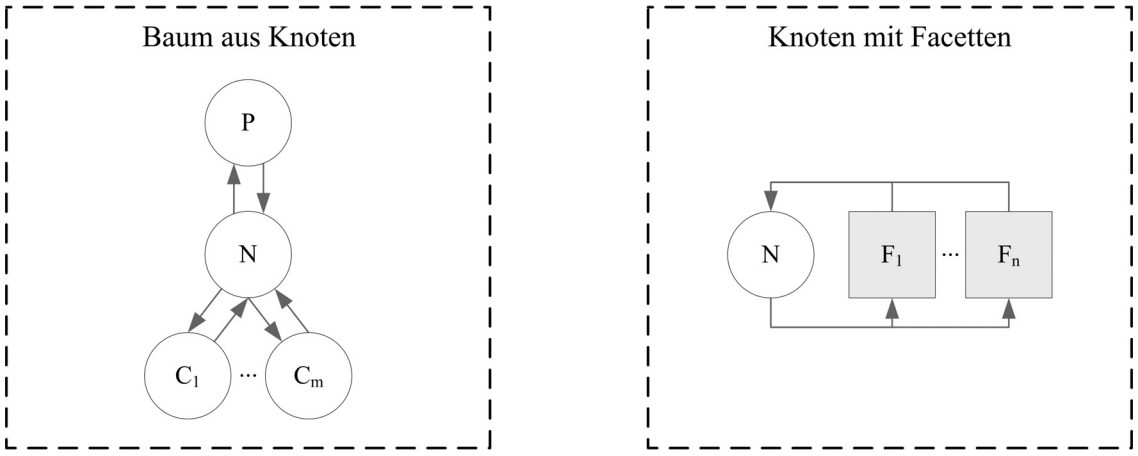


Abbildung 5.6: Links – Mit Eltern und Kindern bidirektional verknüpfter Knoten im Baum. Rechts – Bidirektionale Referenzen zwischen einem Knoten und seinen Facetten. C_m = Kind (*Child*); N = Knoten (*Node*); P = Vater (*Parent*); F_n = Facette.

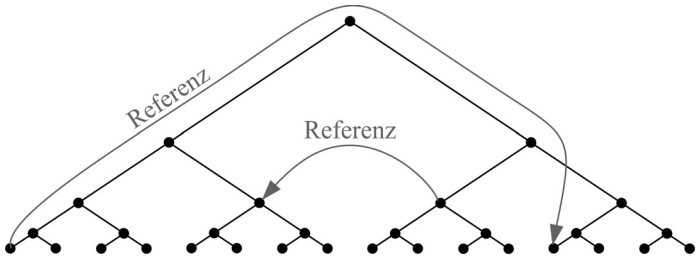


Abbildung 5.7: Referenzierung von Knoten in nichtdirekter Verwandschaft.

Referenzen zu seinem Vater- (*parent*) und seinen Kindknoten (*children*). Da der Knoten Kind seines Vaters und gleichzeitig Vater seiner Kinder ist, halten sein Vater und seine Kinder wiederum Referenzen auf ihn. Dadurch entstehen bidirektionale Referenzen, was eine auf- wie abwärts gerichtete Navigation im Baum ermöglicht (Abb. 5.6 links). Zur Aufnahme von Facetten hält der Knoten auf diese ebenfalls Referenzen (Abb. 5.6 rechts). Umgekehrt hält auch jede Facette eine Referenz auf ihren Knoten (Kap. 5.3.2).

In allen nachfolgenden Abbildungen der Arbeit repräsentieren *Kreise* Knoten.

Aufnahme von Modell-Funktions-Basiselementen

Die Fähigkeit eines Knotens, mehrere Modell-Funktionsbausteine aufnehmen zu können, dient als Entwurfsmöglichkeit, mehrere, für eine spezifische Aufgabe erforderliche Funktionsbausteine an einem Punkt zu bündeln. Diese können als wiederverwendbare Softwarekomponenten (Kap. 5.1) in ein Baukastensystem aufgenommen werden. So kann ein Knoten zur Erfüllung eines Teilaspekts verschiedener Aufgaben herangezogen werden.

```
NODE = [
   Attributes = [
        STRING name
        STRING locator
        STRING id

        NODE parent
        NODE children
        FACET facets
   ]
   GetNode(STRING locator): NODE
   GetFacet(STRING locator): FACET
   GetInfo(): NODE_INFO
   GetSubTreeInfo(): NODE_INFO

   SetType(STRING type)
]
```

Abbildung 5.8: Struktur des Knotens.

Beschaffung von Referenzen

Zur Beschaffung von Referenzen weist der Knoten als Schnittstellen die zwei Funktionen *GetNode(locator)*, *GetFacet(locator)* auf. Ihnen wird als Parameter ein Locator übergeben, als Rückgabewert liefern sie einen Knoten bzw. eine Facette. Damit ist die Referenzierung von Knoten und Facetten von jedem Ort aus möglich. Abbildung 5.7 zeigt ein abstraktes Beispiel zweier Referenzierungen. Der Knoten selbst ist neben seiner Eigenschaft als Gerüstelement lediglich ein Speicher für Facetten – eine weitere Implementierung von Funktionalität ist an dieser Stelle nicht vorgesehen. Deshalb werden an dieser Stelle keine externen Referenzen benötigt, so dass die Beschaffung von Referenzen den gespeicherten Facetten vorbehalten bleibt.

5.3.2 Facette

Das Systemkonzept für wandelbare Produktionsanlagen beschreibt das Modell-Funktions-Basiselement als Träger elementarer Funktionalität (Kap. 3.5). Sein Äquivalent in der Softwarearchitektur ist die *Facette*. Die Bezeichnung resultiert aus der Eigenschaft, dass mehrere Funktions-Basisbausteine in einem Knoten zusammengefasst sein können, so dass dieser je nach Nutzung eine andere „Seite“ – sprich Facette offenbart.

Struktureller Aufbau

Der strukturelle Aufbau der Facette ist in Abbildung 5.9 dargestellt. Als Attribute weist die Facette einen Namen (*name*), einen Locator (*locator*) zur eindeutigen Referenzierbarkeit und eine weltweit eindeutige Identität (*id*) auf. Zur strukturellen Verknüpfung hält sie eine Referenz auf ihren Knoten (*host*). Indem sie an diesem die Funktionen *GetNode(locator)* oder *GetFacet(locator)* aufruft, kann die Facette Referenzen anderer Knoten und Facetten zur Interaktion beziehen. Die so beschafften

```
FACET  = [
   Attributes = [
        STRING name
        STRING locator
        STRING id
        NODE host

        LIST_FACETS facets
        LIST_CHILDREN nodes
        LIST_PUBLISHERS publishers
        Callbacks = [
        ]
   ]
   GetInfo(): FACET_INFO
]
```

Abbildung 5.9: Struktur der Basisfacette.

Referenzen werden in Listen (*facets*, *nodes*) gespeichert. Eine Facette kann Publisher realisieren, um andere Facetten über Ereignisse zu informieren. Diese werden in einer weiteren Liste (*publishers*) gespeichert. Umgekehrt kann sich eine Facette über die Subscriberverwaltung für die Zustellung von Ereignissen an Publishern anderer Facetten registrieren (Kap. 5.2.2). Für jede Ereignisart verfügt sie über eine passende Callbackprozedur.

Die Basisfacette selbst beinhaltet noch keine Funktionalität. Sie muss erst durch Spezialisierung in weitere Facetten hinzugefügt werden, wie die exemplarischen Grundfacetten zeigen (Kap. 5.6).

In allen nachfolgenden Abbildungen der Arbeit repräsentieren *Quadrate* Facetten.

5.3.3 Module

Stellen Facettenmodule komplexe Funktionalität zur Verfügung, kann zur Wahrung der Übersichtlichkeit eine Auslagerung der Implementierung in mehrere *Module* von Vorteil sein. Module sind rein interne Strukturen der Facette und treten gegenüber anderen Facetten oder Knoten nicht in Erscheinung. Sie sind nach außen schnittstellenlos.

Struktureller Aufbau

Ein Modul besitzt eine Referenz auf seine Facette (*facet*). Darüber hat das Modul Zugriff auf alle Knoten- und Facettenreferenzen, die in den Listen (*facets*, *nodes*) seiner Facette gespeichert sind. Außerdem können Module über ihre Facette auf deren Knoten zugreifen und über diesen alle weiteren Knoten und Facetten des Gesamtsystems erreichen. Module anderer Facetten sind allerdings nicht erreichbar und bleiben verborgen, da das Architekturkonzept keine Schnittstellen zum Zugriff auf facettenfremde Module vorsieht. Zusätzlich können Module auf die Publisher ihrer Facette zugreifen, eigene Publisher erzeugen und sich für die Zustellung von Ereignissen an fremden Publishern registrieren.

5.4 Verhaltensformen von Facetten

Zur Umsetzung der Steuerungs-, Informationsverwaltungs-, Informationserzeugungs- und Informationsverarbeitungsaufgaben (Kap. 2.2) können je nach Situation unterschiedliche Verhaltensformen notwendig sein. Dies berücksichtigt die Architektur und unterscheidet *reaktives und proaktives* sowie *synchrones und asynchrones Verhalten* als Grundformen einzelner Facetten. Komplexere Abläufe zwischen mehreren Facetten bilden *Reflex- und koordiniertes Verhalten* ab, die auf den Grundformen aufbauen.

Die Beschreibung des Verhaltens erfolgt losgelöst von einer zu verwendenden Programmiersprache, da prinzipiell jede Sprache geeignet ist, sofern sie Nebenläufigkeiten und Synchronisationsmechanismen unterstützt.

Reaktives und proaktives Verhalten

Facetten können reaktives oder proaktives Verhalten aufweisen. *Reaktives Verhalten* wird aktiviert, wenn an einer Facette Funktionen oder Prozeduren aufgerufen werden. Eine reaktiv aktivierte Facette kann wiederum an weiteren Facetten reaktives oder proaktives Verhalten induzieren, so dass Aktivierungskaskaden entstehen. Beim *proaktiven Verhalten* ist die Facette ein Aktivitätsträger. Sie ist entweder ständig oder in zyklischen Intervallen aktiv oder wird von außen aktiviert. Eine proaktive Facette kann an sich selbst oder an anderen Facetten reaktives Verhalten durch Funktions- und Prozeduraufrufe induzieren. Durch proaktives Verhalten kann eine Facette bspw. selbstständig Informationen (zu bestimmten Zeitpunkten) abfragen, Zustände überwachen oder einen Facettenverbund koordinieren.

Synchrones und asynchrones Verhalten

Synchrones Verhalten zeigt eine Facette, die an einer zweiten Facette proaktiv oder reaktiv einen Aufruf tätigt und die Abarbeitung des Aufrufs abwartet. Bei *asynchronem Verhalten* wartet die Facette das Ende des Aufrufs nicht ab, sondern führt ihre Aktivitäten unmittelbar fort. Infolgedessen können sinnvoll nur Prozeduren aufgerufen werden, da sie keine Rückgabewerte liefern. Die Ereignisübermittlung von Publishern an Subscriber erfolgt ebenfalls asynchron. Asynchrones Verhalten kann zur parallelen Benachrichtigung mehrerer Empfänger oder zum gleichzeitigen Anstoßen mehrerer Aktivitäten genutzt werden. Abbildung 5.10 stellt beide Verhaltensweisen dar.

Reflexverhalten und koordiniertes Verhalten

Reflexverhalten entsteht als Reaktion auf äußere Ereignisse, bei der kaskadenartig mehrere Facetten aktiviert werden. Der Begriff wurde in Anlehnung an die, unter anderem aus der Humanphysiologie bekannten, Reflexe gewählt [95]. Ebenso wie dort Reflexbögen vorkommen, etwa bei dem Kniereflex, werden zur Realisierung von Reflexverhalten Reflexketten aufgebaut. Sie übernehmen Aufgaben der sensorischen Reizdetektion, der Reizverarbeitung sowie der Aktivierung von Aktorik als Reflexantwort. Neben einfachen Reflexketten, bei der etwa eine Facette ein einzelnes Sensorsignal verarbeitet

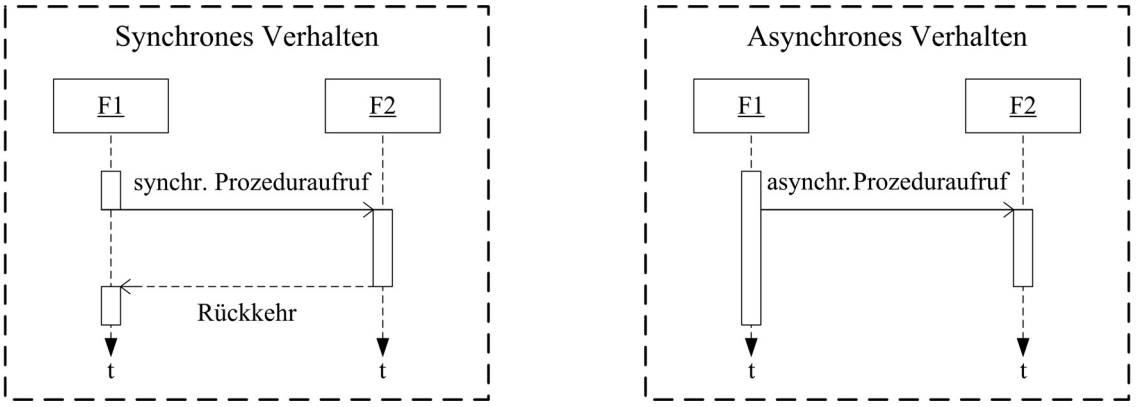


Abbildung 5.10: Vergleich von synchronem und asynchronem Verhalten. F_n = Facette.

und als Folge einen einzelnen Aktor aktiviert, können auch komplexe Reflexketten realisiert werden. So ist die Verarbeitung einer oder mehrerer Reize zu einer oder mehrerer Antworten möglich, wobei eine oder mehrere Facetten involviert sein können. Abbildung 5.11 zeigt auf linker Seite schematisch zwei Reflexketten.

Die Aktivierung von Verhalten hängt ausschließlich von der Reizung eines Sensors ab. Sensoren verfügen über spezifizierte Schnittstellen, so dass Reflexketten mit unterschiedlicher Sensorik gekoppelt werden können. Deshalb zeigen sie bei sorgsamen Entwurf einen echten modularen Charakter und zeichnen sich dadurch aus, dass eine direkte Ankopplung von Steuerungssystemen aus höheren Ebenen nicht notwendig ist. Dadurch ist das Erzeugen von autarkem Verhalten möglich. Somit sind Reflexe ein wichtiger Schlüssel zur Realisierung selbstorganisierenden Systemverhaltens.

Als Sensoren eignen sich physische Funktionselemente und Steuerungsmodell-Funktionselemente, d.h. auch Software kann Reflexe auslösen. Eine gesteuerte Reflexauslösung ist zwar möglich aber nicht Teil des Konzepts, da dies dem asynchronen Charakter – das reflexauslösende Ereignis gilt als beliebig eintretbar – entgegensteht.

Kapitel 7.4, 8.2, 8.3.3, 8.4 und 8.4.2 behandeln Teillösungen zur logistischen Selbstorganisation von Produktionsanlagen auf verschiedenen Hauptebenen, wobei Reflexketten zum Einsatz gelangen.

Koordiniertes Verhalten dient der zeitlichen und ereignisabhängigen Abstimmung mehrerer Facetten unter Leitung einer Koordinierungsinstanz. Es sind neben linearen auch verzweigte Abläufe realisierbar, die von den Zuständen der Facetten abhängig sein können. Neben der Verarbeitung von asynchronen Ereignissen ist auch die Verwendung von Zeitgliedern denkbar. Allerdings widersprechen Letztere tendenziell dem Modularisierungsgedanken, da sie häufig exakt auf ein bestimmtes Umfeld ausgerichtet sind. Ändert sich dieses, sind Zeitglieder mit möglichen manuellen Aufwendungen notgedrungen anzupassen.

Abbildung 5.11 zeigt auf rechter Seite die Koordinierung vierer Facetten durch einen übergeordneten Controller. Dieser führt Schritt 1 aus, ermittelt in Schritt 2 den Zustand der Facette, um davon abhängig Schritt 3a oder 3b auszuführen.

In einer Koordinierungsinstanz kann ein Ablauf fest implementiert werden, wobei dann eine fixe Zuordnung zu den koordinierten Facetten besteht. Ändern sich die Facetten, ist infolgedessen der Ablauf

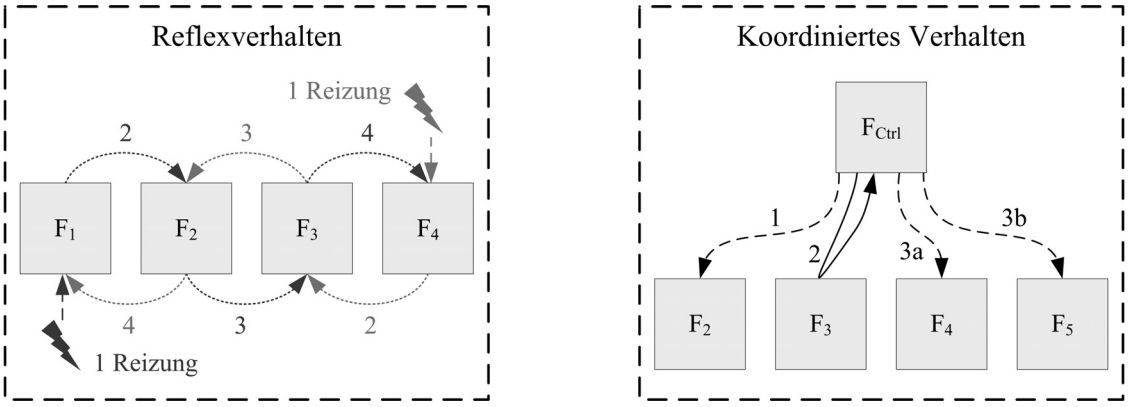


Abbildung 5.11: Vergleich von Reflex- und koordiniertem Verhalten. F_n = Facette.

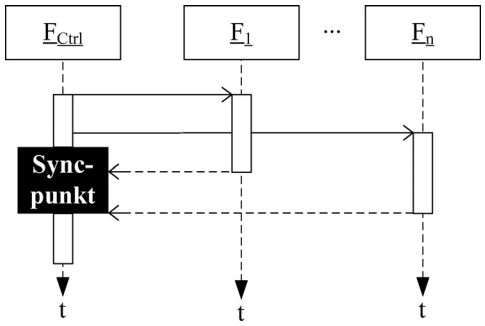


Abbildung 5.12: Synchronisationspunkt – Der Controller wartet, bevor er mit seiner Ausführung fortfährt, auf das Beenden der von ihm an anderen Facetten angestoßenen Aktivitäten. F_n = Facette.

anzupassen, so dass auch hier manuelle Aufwendungen nicht in jedem Fall vermeidbar sind. Alternativ besteht die Möglichkeit einer generischen Koordinierungsinstanz, die Ablaufskripte mit Hilfe eines Skriptinterpreters ausführen kann. In diesem Fall müssen bei Änderungen des Umfelds lediglich die Skripte ausgetauscht werden. Kapitel 5.6.2 stellt einen Skriptinterpreter als Facette vor.

Synchronisationspunkte und gegenseitiger Ausschluss

Zur Koordinierung von nebenläufigem Verhalten ist das Setzen von *Synchronisationspunkten* von Bedeutung. Mit ihrer Hilfe ist es möglich, Aktivitäten mehrerer Facetten parallel auszulösen und den Aktivitätsauslöser an seiner Fortführung solange zu hindern, bis die Aktivitäten abgeschlossen sind (Abb. 5.12). Synchronisationspunkte können bspw. dazu verwendet werden, mehrere Einzelachsen einer Spannvorrichtung parallel in Stellung zu bringen, bevor mit der Durchführung des eigentlichen Prozesses fortgefahren wird.

Parallel aktive Facetten führen zu Problemen von Nebenläufigkeiten, die in der Informatik aus den Bereichen Multithreading, Mehrprozessorsystemen oder verteilten Systemen bekannt sind [40, 119, 129]. Aufgabe hierbei ist die Vermeidung inkonsistenter Daten, was durch den gegenseitigen Ausschluss von Aktivitätsträgern erreicht wird. Hierzu sind entsprechende Mechanismen wie *Semaphoren, Monitore, Locks, Reader-Writer-Locks, Conditions*, u.a. bekannt [118]. Durch sie wird der Zu-

griff auf „sensible“ Daten sequenzialisiert. Für die Umsetzung der Architektur wird die Verfügbarkeit dieser Synchronisationsmechanismen postuliert.

5.5 Komplexe Softwareelemente und ihre Wandlungsfähigkeit

Mit den bisher vorgestellten Mitteln ist die Erstellung komplexer Softwareelemente (Modell-Funktions-(Basis-)Elemente aus Sicht der Systemtheorie) mit vier wesentlichen Vorgehensweisen *Funktionserweiterung von Facetten, Vereinen von Facetten in Knoten, Fügen von Knoten zu Bäumen* sowie die *Kopplung über Reflexe und Koordination* möglich.

Funktionserweiterung von Facetten

Die Funktionserweiterung einer bestehenden Facette ist eine Möglichkeit, eine Facette mit höherer Funktionalität bzw. Verhalten zu realisieren. Dabei kann die alte Facette ersetzt oder eine neue erstellt werden. Die Anwendung dieses Ansatzes ist sorgsam abzuwägen, da entweder das Prinzip des Basisbausteins oder des Erhalts der minimalen Bausteinmenge des Baukastensystems verletzt werden kann. Ersteres tritt auf, wenn die Facette eine Übermenge an Funktionalität aufweist, die aber in keinem Anwendungsfall vollständig benötigt wird. Zweiteres geschieht, wenn die benötigte Funktionalität durch Fügen bestehender Bausteine realisiert werden kann bzw. Teilfunktionalität der Facette auch an anderer Stelle benötigt wird und deshalb vorteilhaft in eine eigene Facette ausgelagert wird.

Vereinen von Facetten in Knoten

Das Vereinen von Facetten in einzelne Knoten dient den Aspekten Organisation und Wiederverwendbarkeit. So können mehrere gleichgeartete Facetten in einem Knoten organisiert werden, ebenso wie mehrere unterschiedliche Facetten, die zur Umsetzung einer Aufgabe benötigt werden. Die Verteilung ist dann in Abhängigkeit der „Sollbruchstelle“ derart zu wählen, das eine Wandlung aufwandsarm vollzogen werden kann.

Fügen von Knoten zu (Sub-)Bäumen

Mit dem Fügen mehrer Knoten zu Bäumen wird das Ziel der Erstellung komplexer Teilsysteme höherer Ordnung verfolgt. Bäume sind als Subbäume Bestandteile der Hauptebenen eins bis fünf des 6-Ebenen$^+$-Modells (Kap. 4.1) oder wiederverwendbare Bausteine eines Baukastensystems. Mehrere Bäume sind zu größeren Bäumen fügbar, so dass zunehmend komplexere Systemstrukturen aufgebaut werden können. Die Wandlung eines Baums erfolgt durch Austausch einzelner Subbäume, wobei in diesem Zusammenhang bereits ein einzelner Knoten als Subbaum aufgefasst wird.

Kopplung über Reflexe und Koordination

Die funktionale Kopplung mehrerer Facetten ist über Knoten- und Baumgrenzen hinaus möglich. So können die beteiligten Facetten in Knoten unterschiedlicher Teiläste liegen. Jedoch gilt, je größer die Distanz der Facetten im Baum, desto aufwändiger gestaltet sich die wandelbare Auslegung. Werden Schnittstellen der Facetten sorgsam definiert, sind Elemente der Reflexketten austauschbar, so dass diese einer Wandlung unterzogen werden können. Zudem ergibt sich daraus der Vorteil, Reflexketten unabhängig von konkreten physischen Elementen entwerfen und in unterschiedliche Szenarien einbinden zu können. Gleiches gilt für koordiniertes Verhalten, insbesondere auch dann, wenn das Verhalten in Skripten hinterlegt ist.

Realisierung von Zwillingselementen

Die Kopplung von Facetten und Knoten mit physischen Elementen zu Zwillingselementen erfolgt dadurch, dass die strukturelle Hierarchie physischer Gerätschaften in Subbäume übergeführt wird und das Geräteverhalten auf unterschiedlichen Hauptebenen des 6-Ebenen$^+$-Modell (Kap. 4.1) in den entsprechenden Ebenen des Subbaums nachgebildet wird. Der Subbaum und der hierarchische Aufbau einer Gerätschaft sind in ihrer Struktur nicht zwingend identisch, da Verhalten auf einer Hauptebene in mehrere Softwareebenen aufgeteilt werden kann. Wandlungsfähigkeit wird dadurch erzielt, dass die physischen „Sollbruchstellen“ einer Gerätschaft, Zelle, Produktionslinie, Produktionsinsel oder der gesamten Produktionsanlage, gleichsam in den Steuerungsmodellen nachgebildet wird (Kap. 3.4).

5.6 Grundfacetten

Im Rahmen dieser Arbeit wurden Facetten konzipiert, die Grundaufgaben abdecken. Mit ihnen können bereits Funktionselemente und einfache Systeme aufgebaut werden. Es sind *Dateisystemverwaltung, skriptfähiger Controller* sowie *Digital-, Analog-* und *Zeichenketten-Eingänge* und *-Ausgänge*, *Pulsweitenmodulator* und *Funktionsgenerator*. Die Facetten zu Ein- und Ausgängen bilden die Basis für die Anbindung von Sensoren oder Aktoren.

Bei der Gestaltung der Facetten wurde auf Generizität Wert gelegt, dem Ziel einer hohen Wiederverwendbarkeit in unterschiedlichen Kontexten folgend. Steuerungsmodelle zu Verfahren der Selbstorganisation bauen teilweise auf den hier vorgestellten Facetten auf (Kap. 7, 8.3, 8.4).

5.6.1 Dateisystemverwaltung

Für einen Knoten und seine Facetten kann es als Funktionselement erforderlich sein, Daten auf Festspeichern des Rechners zu hinterlegen, weshalb eine Dateisystemverwaltung (*FacetFile*) entworfen wurde (Abb. 5.13). Sie kontrolliert exklusiv einen eigenen Festspeicherbereich, auf den nur sie zugreifen kann. Die Kapselung des Festspeichers liegt darin begründet, dass für andere Facetten eine möglichst plattformneutrale Entwicklung möglich sein soll, um eine hohe Wiederverwendbarkeit zu

```
FACET_FILE = [
  Publisher = [
      FileChanged(FILE_STATE state)
      DirChanged(DIR_STATE state)
  ]
  GetBasePath(STRING which): STRING
  Exists(STRING which, STRING path): BOOL THROWS Malformed
  List(STRING which, STRING path, BOOL recursive): STRINGS THROWS Malformed, Unknown
  IsFile(STRING which, STRING path): BOOL THROWS Malformed, Unknown
  IsDir(STRING which, STRING path): BOOL THROWS Malformed, Unknown
  GetFile(STRING which, STRING path): DATA THROWS Malformed, Unknown
  AddFile(STRING path, STRING name, DATA data) THROWS Malformed, SameAlreadyExistent
  DelFile(STRING path) THROWS Malformed, Unknown
  AddDir(STRING path) THROWS Malformed, SameAlreadyExistent
  DelDir(STRING path) THROWS Malformed, Unknown
]
```

Abbildung 5.13: Facette zum kontrollierten Festspeicherzugriff für Facetten eines Knotens.

erreichen. Da der Zugriff auf Festspeicher von dessen Art und dem Betriebssystem des Rechners abhängt, entstehen so nur in dieser Facette Anpassungsbedarfe, wenn Plattformen gewandelt werden bzw. neue zu unterstützten sind.

Standardmäßig enthält jeder Knoten eine eigene Dateiverwaltung. Mehrere Facetten in einem Knoten gebündelt können so kooperativ Aufgaben lösen und dabei Dateien untereinander austauschen. Hierzu stellt die Dateiverwaltung zwei Publisher (*FileChanged(state)*, *DirChanged(state)*) zur Verfügung, die Veränderungen von Dateien signalisieren. Damit ist die Realisierung von ereignisgesteuerten Erzeuger-Verbraucher-Ketten möglich.

Die Facette unterscheidet zwischen nurlesbaren (ROM – Read Only Memory) und wiederbeschreibbaren Speicherbereichen (RAM – Random Access Memory), womit eine räumliche Trennung elementarer Grunddaten von anderen Daten möglich ist, die während des Betriebs anfallen.

5.6.2 Skriptfähiger Controller

Der skriptfähige Controller (*FacetScriptableController*) dient zur Realisierung koordinierten Verhaltens (Abb. 5.14). Seine Funktionalität kann in die drei Abschnitte *Skriptverwaltung, Skriptsteuerung* und *Skriptdebugging* untergliedert werden.

Die Skriptverwaltung umfasst das Hinzufügen neuer und das Entfernen oder Abfragen bereits enthaltener Skripte. Mit der Skriptsteuerung können Skripte gestartet, gestoppt oder zurückgesetzt werden. Das Skriptdebugging erlaubt die schrittweise Ausführung von Skripten, das Hinzufügen von Breakpoints oder das Löschen sowie das Abfragen der Liste gesetzter Breakpoints.

Die Struktur der Skriptverwaltung eignet sich für beliebige Skriptsprachen, weshalb an dieser Stelle keine einschränkende Auswahl getroffen wird. Dennoch stellt das Konzept Anforderungen, die Ausschlusskriterien für spezifische Skriptsprachen darstellen können. Knoten und Facetten des gesamten Baums müssen aus einem Skript ansprechbar sein, wozu Mechanismen erforderlich sind, die eine

```
FACET_SCRIPTABLE_CONTROLLER = [
  Attributes = [
      LIST_STRING scripts
      STRING activeScript
      INT steps
      LIST_INTS breakpoints
  ]
  Publisher = [
      ScriptChanged(STRING name)
      ExecutionStarted()
      ExecutionStopped()
      Reset()
  ]
  AddScript(STRING name, STRING script, BOOL force) THROWS Empty, SameAlreadyExistent
  DelScript(STRING name) THROWS Empty, Unknown
  GetScript(STRING name): STRING THROWS Empty, Unknown
  SelectScript(STRING name) THROWS Unknown
  Start() THROWS NoActiveScript
  Stop() INT THROWS NoActiveScript
  Reset() THROWS NoActiveScript
  Step(INT steps) THROWS NoActiveScript
  StopAt(INT line) THROWS NoActiveScript
  AddBreakpoint(INT line) THROWS NoActiveScript, SameAlreadyExistent
  DelBreakpoint(INT line) THROWS NoActiveScript, Unknown
  GetBreakpoints(): INTS THROWS NoActiveScript
]
```

Abbildung 5.14: Facette zur ereignis- oder zeitdiskreten Koordination mehrerer Facetten mittels Skripten.

Auflösung von Locator nach Referenzen erlauben. Zudem ist sowohl die Erzeugung von Publishern als auch das Registrieren von Callbackfunktionen für die Ereigniszustellung aus einem Skript heraus notwendig. Anweisungen müssen sowohl sequenziell als auch parallel ausführbar sein, wie es etwa für das gleichzeitige Ansprechen von Aktorik erforderlich sein kann. Kapitel 10.5 stellt eine Skriptsprache sowie die Referenzimplementierung eines skriptfähigen Controllers vor.

5.6.3 Digital-Eingang und -Ausgang

Digital-Eingangs- und -Ausgangsfacetten dienen der Kapselung entsprechender Hardwareschnittstellen für die Ankopplung binärer Sensorik und Aktorik. Sie sind derart konzipiert, dass ein flexibler Einsatz in unterschiedlichen Kontexten möglich ist. Dabei erfolgt eine bewusste Abstraktion von realer Hardware, da Facetten Teil der Ablaufsteuerung und nicht Bestandteil der Ausführungssteuerung sind (Kap. 3.4).

Digital-Eingang

Die Digital-Eingangsfacette (*FacetDigitalIn*) stellt sowohl Schnittstellen zur Zustandsabfrage (*Get-/SetState*) eines binären Eingangssignals als auch zur Konfiguration der Signalerfassung zur Verfü-

```
FACET_DIGITAL_IN = [
  Attributes = [
      BOOL active
      DURATION duration
      PERCENTAGE fraction
      STRING mode
      INT number
      BOOL state
  ]
  Publishers = [
      Toggle(BOOL state)
      RisingEdge()
      FallingEdge()
  ]
  GetActiveState(): BOOL
  SetActiveState(BOOL state)
  GetMeasurementDuration(): DURATION
  SetMeasurementDuration(DURATION duration)
  GetMeasurementSuccessFraction(): PERCENTAGE
  SetMeasurementSuccessFraction()
  GetNumberOfMeasurements(): INT
  SetNumberOfMeasurements(INT number)
  GetState(): BOOL
]
```

Abbildung 5.15: Facette zur Ankopplung eines digitalen Sensors.

gung (Abb. 5.15). Ferner realisiert sie mehrere Publisher (*Toggle(state), RisingEdge, FallingEdge*), die Interessenten über Zustandswechsel des Eingangssignals informieren.

Die Konfiguration umfasst zwei Aspekte. Erstens ist aus der Praxis bekannt, dass der logische Wert nicht dem anliegenden Signalpegel entsprechen muss, d.h. die logische Eins kann ungleich dem physikalischen High-Pegel sein. Für die Zuordnung des Signalpegels zur logischen Eins steht eine Funktionsschnittstelle (*Get-/SetActiveState*) zur Verfügung. Als zweiter Aspekt ist ein Flattern zwischen High- und Low-Pegel des Eingangssignals möglich, so dass dieses für ein gewisses Zeitintervall undefiniert ist. Um dies zu kompensieren und immer einen definierten Zustand zu detektieren, können Messreihen aufgenommen und ein gültiger Zustand nach dem Mehrheitsprinzip berechnet werden. Die Messreihen sind durch Anzahl der Messungen und dem Zeitabstand zwischen zwei Messungen bestimmt. Für ihre Konfiguration stehen Funktionsschnittstellen (*Get-/SetMeasurementDuration, Get-/SetMeasurementSuccessFraction, Get-/SetNumberOfMeasurements*) bereit.

Digital-Ausgang und Pulsweitenmodulator

Die Aufgabe einer Digital-Ausgangsfacette (*FacetDigitalOut*) besteht in dem kontrollierten Setzen (*Get-/SetState*) eines binären Signals zwischen High- und Low-Pegel. Ebenso wie bei der Eingangsfacette steht für die Zuordnung des Signalpegels zur logischen Eins eine Funktionsschnittstelle (*Get-/SetActiveState*) bereit (Abb. 5.16) und ebenso verfügt sie über die gleichen Publisher wie die Digital-Eingangsfacette.

```
FACET_DIGITAL_OUT = [
  Attributes = [
      BOOL active
      BOOL state
  ]
  Publishers = [
      Toggle(BOOL state)
      RisingEdge()
      FallingEdge()
  ]
  GetActiveState(): BOOL
  SetActiveState(BOOL state)
  GetState(): BOOL
  SetState(BOOL state)
  SetPWM(FACET_PULSE_WIDTH_MODULATOR pwm)
]
```

Abbildung 5.16: Facette zur Ansteuerung eines digitalen Aktors.

Digital-Ausgänge können durch *Pulsweitenmodulation (PWM)* für die Übertragung analoger Informationen genutzt werden [39]. Um PWM für Digital-Ausgangsfacetten zu ermöglichen wurde ein einfacher Pulsweitenmodulator (*FacetPulseWidthModulator*) konzipiert (Abb. 5.17). Er ist als Binärsignalgeber gestaltet und erzeugt zyklisch Rechtecksignale aus einer Liste freiwählbarer Paare aus High-/Low-Pegeln. Anhand dieser Liste, für die Funktionsschnittstellen (*Get-/SetDurationList*) bestehen, können analoge Kurvenverläufe nachgebildet werden. Zur Kontrolle bietet der Modulator weitere Schnittstellen (*Start(reset), Stop*) an. Über Publisher (*Toggle(state), RisingEdge, FallingEdge*) werden Interessenten über Zustandswechsel informiert. Die Verknüpfung von PWM und Digital-Ausgangsfacette erfolgt über die Funktionsschnittstelle *SetPWM(pwm)* des Ausgangs. Durch den Aufruf registriert sich dieser am Publisher *Toggle* des PWMs. Damit können mehrere Digital-Ausgänge „synchron“ geschaltet werden.

5.6.4 Analog-Eingang und -Ausgang

Die Analog-Eingangs- und -Ausgangsfacetten dienen der Kapselung entsprechender Hardwareschnittstellen für die Ankopplung analoger Sensorik und Aktorik. Sie sind Teil der Ablauf- und nicht Bestandteil der Ausführungssteuerung (Kap. 3.4). Ihre Konzeption erfolgt für den flexiblen Einsatz in unterschiedlichen Kontexten. Deshalb werden reale Hardware, elektrische Parameter (z.B. Spannungs-, Strombereich) und Auflösung abstrahiert und ein normierter Wertebereich von 0.0 bis 1.0 angenommen.

Analog-Eingang

Die Analog-Eingangsfacette (*FacetAnalogIn*) stellt zur Zustandsabfrage (*Get-/SetValue*) eines analogen Eingangssignals sowie zur Konfiguration der Erzeugung von Ereignissen Schnittstellen zur Verfügung (Abb. 5.18). Zur Benachrichtigung von Interessenten sind verschiedene Publisher (*BarrierRisingEdge(value), BarrierFallingEdge(value), LevelChanged(level)*) vorhanden.

```
FACET_PULSE_WIDTH_MODULATOR = [
  Attributes = [
      FLOAT frequency
      FLOAT value
  ]
  Publisher = [
      Toggle(BOOL isHigh)
      RisingEdge()
      FallingEdge()
  ]
  GetDurationList(): HIGH_LOW_DURATIONS
  SetDurationList(HIGH_LOW_DURATIONS highLowDurations): THROWS Empty, Malformed
  GetState(): FLOAT
  SetState(FLOAT value)
  Start()
  Stop()
]
```

Abbildung 5.17: Facette zur Pulsweitenmodulation.

Um die Anzahl der zu erzeugenden Ereignisse einschränken zu können, sind zwei verschiedene Mechanismen vorgesehen – *Barrieren* und die *Diskretisierung des Wertebereichs*. Barrieren sind frei wählbare Schranken, von denen eine beliebige Anzahl gesetzt und wieder entfernt werden kann. Werden Schranken über- oder unterschritten, wird ein Ereignis über die entsprechenden Publisher (*BarrierRisingEdge(value)*, *BarrierFallingEdge(value)*) ausgelöst. Die Diskretisierung erfolgt als äquidistante Unterteilung des Wertebereichs. Sobald ein Wechsel in einen anderen Bereich erfolgt, wird ebenfalls über den Publisher *LevelChanged(level)* ein Ereignis unter Angabe des Bereichsindex ausgelöst.

Analog-Ausgang und Funktionsgenerator

Die Aufgabe einer Analog-Ausgangsfacette (*FacetDigitalOut*, Abb. 5.19) besteht in dem kontrollierten Setzen (*Get-/SetValue*) eines normierten, analogen Signals zwischen 0.0 und 1.0. Ebenso wie bei der Eingangsfacette sind die beiden Mechanismen *Barrieren* und die *Diskretisierung des Wertebereichs* (Abb. 5.20) vorgesehen. Im Gegensatz zum Eingang kann eine Stufe der Diskretisierung über eine Funktionsschnittstelle (*SetLevel(level)*) ausgewählt werden, wobei sich als Analogwert $W = \frac{W_{level}+W_{level+1}}{2}$ ergibt.

An dieser Stelle sei angemerkt, dass das Setzen eines Analogwerts oder einer Stufe grundsätzlich mit einem Quantisierungsfehler behaftet ist, dessen Größe von der Auflösung der tatsächlichen analogen Hardwareschnittstelle abhängt [92].

Über Analog-Ausgänge ist die Ausgabe eines durch Funktionen erzeugten Analogsignals möglich, weshalb ein generischer Funktionsgenerator (*FacetFunctionGenerator (FG)*) entworfen wurde (Abb. 5.21).

Dieser erzeugt in freiwählbarer Frequenz (*Get-/SetFrequency*) Funktionswerte aus einer, über Funktionsschnittstellen (*Get-/SetFunctionList(functionDomains)*) konfigurierbaren Funktionsliste. Sie ent-

```
FACET_ANALOG_IN = [
   Attributes = [
        FLOAT value
        INT levels
   ]
   Publishers = [
        BarrierRisingEdge(FLOAT value)
        BarrierFallingEdge(FLOAT value)
        LevelChanged(INT level)
   ]
   GetValue(): FLOAT
   DiscretizeIntoLevels(INT number, FLOAT distance) THROWS Invalid
   AddBarrier(FLOAT barrier) THROWS SameAlreadyExistent
   DelBarrier(FLOAT barrier) THROWS Unknown
   ListBarriers(): FLOATS
]
```

Abbildung 5.18: Facette zur Ankopplung eines analogen Sensors.

```
FACET_ANALOG_OUT = [
   Attributes = [
        FLOAT value
        INT levels
   ]
   Publishers = [
        BarrierRisingEdge(FLOAT value)
        BarrierFallingEdge(FLOAT value)
        LevelChanged(INT level)
   ]
   GetValue(): FLOAT
   SetValue(FLOAT state)
   AddBarrier(FLOAT barrier) THROWS SameAlreadyExistent
   DelBarrier(FLOAT barrier) THROWS Unknown
   ListBarriers(): FLOATS
   DiscretizeIntoLevels(INT number, FLOAT distance) THROWS Invalid
   SetLevel(INT level)
   SetFG(FACET_FUNCTION_GENERATOR fg)
]
```

Abbildung 5.19: Facette zur Ansteuerung eines analogen Aktors.

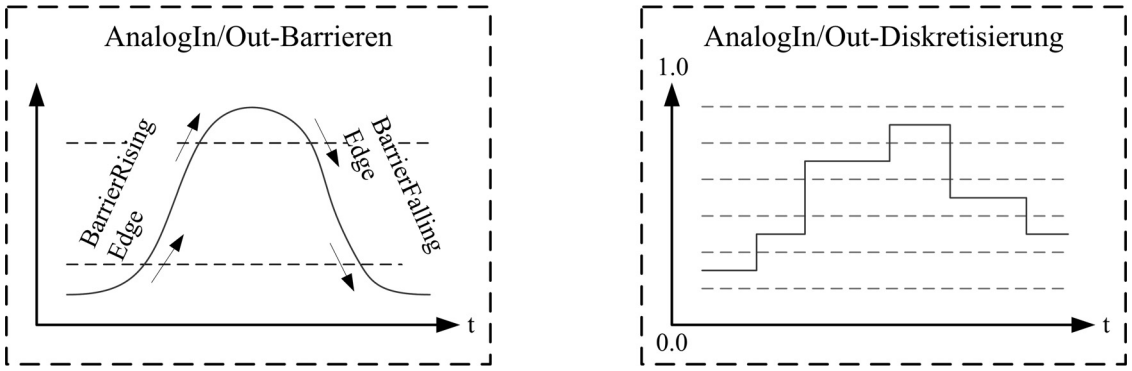


Abbildung 5.20: Links – Funktionsweise der Barrieren. Rechts – Funktionsweise der Diskretisierung des analogen Sensor-/Aktorwertebereichs

```
FACET_FUNCTION_GENERATOR = [
   Attributes = [
        FLOAT value
   ]
   Publishers = [
        ValueChanged(FLOAT value)
   ]
   GetFrequency(): FLOAT
   SetFrequency(FLOAT f)
   GetFunctionDomainList(): FUNCTION_DOMAINS
   SetFunctionDomainList(FUNCTION_DOMAINS functionDomains) THROWS Empty, Malformed
   Start(BOOL reset)
   Stop()
]
```

Abbildung 5.21: Facette zur Funktionsgenerierung mit Zeit t als Definitionsbereich. Die Normierung des Wertebereichs für Analogausgänge erfolgt implizit über die übergebene Funktion.

hält Paare aus Funktionen und Wertebereichen (als Zeitintervall $[t1, t2)$), so dass gestückelte Funktionen möglich sind. Die Liste wird zyklisch abgearbeitet, so dass periodische Signale erzeugt werden können. Die Normierung des Wertebereichs zwischen 0.0 und 1.0 erfolgt implizit durch die Wahl der Funktion, wodurch der Generator für andere Aufgaben verwendbar ist. Zur Kontrolle bietet der Generator weitere Funktionsschnittstellen (*Start(reset), Stop*) an. Die Verknüpfung des Generators mit einer Analog-Ausgangsfacette erfolgt über die Funktionsschnittstelle *SetFG(fg)* des Ausgangs. Durch den Aufruf registriert sich die Facette am Publisher *ValueChanged* des Generators. Mehrere Analog-Ausgänge können von diesem „synchron“ geschalten werden.

5.6.5 Zeichenketten-Eingang und -Ausgang

Wie die bereits vorangegangenen Ein- und Ausgangstypen auch, dienen Zeichenkettenfacetten zur Kapselung entsprechender Hardwareschnittstellen für die Ankopplung zeichenkettenbasierter Sensorik und Aktorik. Ebenso erfolgt eine bewusste Abstraktion von realer Hardware und darüber hinaus von Kommunikationsprotokollen, wie sie etwa für serielle Schnittstellen immer gerätespezifisch vorkommen, da die Facetten Teil der Ablaufsteuerung und nicht Bestandteil der Ausführungssteuerung

```
FACET_STRING_IN = [
   Attributes = [
        STRING string
   ]
   Publishers = [
        NewString()
        StringChanged(STRING string)
   ]
   GetString(): STRING
]
```

Abbildung 5.22: Facette zur Ankopplung eines zeichenkettenbasierten Sensors.

```
FACET_STRING_OUT = [
   Attributes = [
        STRING string
   ]
   Publishers = [
        NewString()
        StringChanged(STRING string)
   ]
   GetString(): STRING
   SetString(STRING)
]
```

Abbildung 5.23: Facette zur Ansteuerung eines zeichenkettenbasierten Aktors.

sind (Kap. 3.4).

Zeichenketten-Eingang

Der Zweck des Eingangs (*FacetStringIn*) liegt im Empfang einer Zeichenkette. Interessenten können sich sowohl für das Ereignis einer neu eingetroffenen Zeichenkette (*NewString*) als auch für dessen gleichzeitige Zustellung (*StringChanged(string)*) an zwei Publishern registrieren (Abb. 5.22). Beide Ereignisse werden immer gleichzeitig ausgelöst. Die Begründung für eine doppelte Benachrichtigung liegt in der Tatsache, dass Ereignisse auch über das Kommunikationsnetzwerk zugestellt werden. Da Zeichenketten beliebig lang sein können, ist ihr Verschicken an alle Interessenten fallabhängig abzuwägen. Zur Abfrage der zuletzt empfangenen Zeichenfolge steht eine Funktionsschnittstelle (*GetString*) zur Verfügung.

Zeichenketten-Ausgang

Über den Ausgang (*FacetStringOut*) kann eine Zeichenkette ausgegeben (*Get-/SetString(string)*) werden (Abb. 5.23). Interessenten können analog zum Eingang über ein solches Ereignis informiert werden.

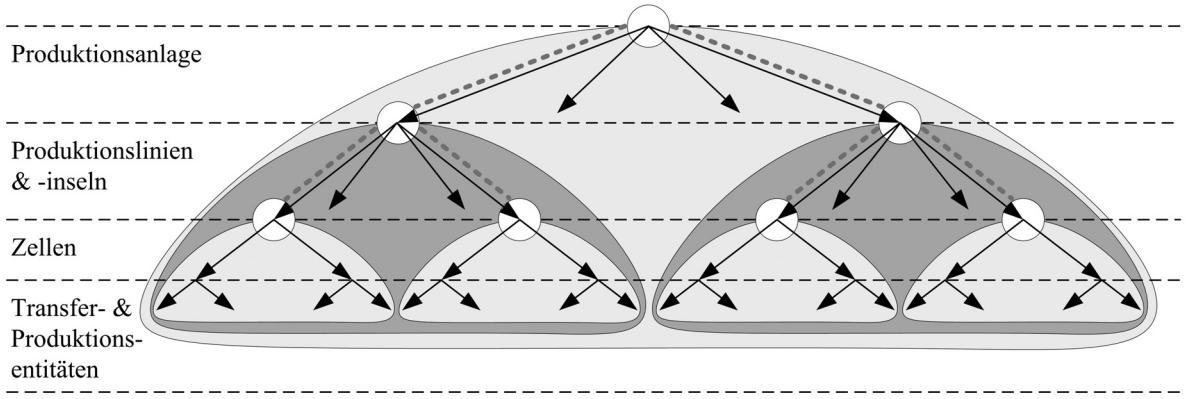


Abbildung 5.24: Zusammenhang zwischen der Hierarchie (gestrichelte Kanten) der Entitäten (Kreise) und dem Gesamtsteuerungsmodell (Pfeile). Als flache Struktur interpretiert, markieren die hell und dunkel markierten Bereiche geschachtelte Zonen der Entitätenhierarchie (Kap. 6.6, S. 83).

5.7 Entität – Abgeschlossene Einheit im Systemverbund

Bereits in Kapitel 4.4.1 wurde der Begriff *Entität* für die Symbiose aus Steuerungsmodell-Subbaum und Rechnerhardware genannt. Da an der Hardware sensorische und aktorische Peripherie angeschlossen wird, verbinden Entitäten Softwaresysteme mit dem physischen Teil der Produktionsanlage. Alle Entitäten des Gesamtsystems bilden gemeinsam die komplette Anlagenintelligenz. Sie stehen zueinander in hierarchischer Beziehung, wobei die Rechnerhierarchie dem 6-Ebenen$^+$-Modell (Kap. 4.1) entspricht. Da jede Entität wiederum einen Subbaum des Gesamtsteuerungsmodells hält, weist auf Softwareseite jede Hauptebene beliebig tief schachtelbare Unterebenen auf (Abb. 5.24).

Funktionsweise und Wandelbarkeit

Die Entität ist, neben der Kopplung von Software und Hardware, für den Aufbau und die Ausführung des Steuerungsmodells verantwortlich. Da der Aufbau unter wandelbaren Gesichtspunkten erfolgen soll, ist das Steuerungsmodell zunächst nicht fest auf der Entität hinterlegt. Stattdessen wird dieser durch manuelle Konfiguration eine Typliste zugewiesen, wobei jedem Typ eine spezifische Steuerungsaufgabe zugeordnet ist. Die Entität kann dadurch mehrere Steuerungsaufgaben wahrnehmen. Durch Abarbeitung dieser Liste bezieht die Entität von der Modellverwaltung (Kap. 4.6.2) sukzessive eine Beschreibung der Modellstruktur und der dazugehörigen Modellelemente (Facetten). Das Entitäten-Steuerungsmodell wird dann automatisiert aufgebaut.

Modellstruktur und Modelle können lokal gespeichert werden, so dass bei einem Neustart der Entität die Informationen nicht erneut beschafft werden müssen. In diesem Fall sind lediglich bei strukturellen Wandlungen der Entität die Informationen zum Steuerungsmodell erneut zu beziehen. Verfügt eine Entität und ihre Peripherie über Sensorik zur Erkennung struktureller Änderungen, kann eine Anpassung automatisiert angestoßen werden. Andernfalls muss die Auslösung manuell geschehen.

Hierarchiebildung und Wandelbarkeit

Entitäten stehen organisatorisch in hierarchischer Beziehung zueinander. Der Steuerungsmodell-Subbaum einer jeden Entität verfügt genau über einen Wurzelknoten. Die Hierarchie entsteht dadurch, dass jedem Wurzelknoten ein anderer Wurzelknoten als Vater (Kap. 5.3.1) zugewiesen wird. Die Hierarchie ist für das übergreifende Adressierungsschema (Kap. 4.4.1) elementare Voraussetzung, da nur so hierarchischen Adressen (*Locator*) bestimmte Knoten, Facetten und Publisher zuordbar sind.

Zwar gestaltet sich eine manuelle Hierarchiebildung durch explizite Verknüpfung der Wurzelknoten einfach, da ein Mensch die physischen Komponenten einer Produktionsanlage einfach in das 6-Ebenen$^+$-Modell (Kap. 4.1) einordnen kann. Aus Sicht der Wandelbarkeit muss die Erkennung allerdings automatisch erfolgen, damit eine stetige Adressierbarkeit der Softwareelemente möglich ist. Die Hierarchie ist eine Organisationsform, die sich graphentheoretisch nur als Baum darstellen lässt. Produktionsanlagen können im Allgemeinen jedoch nur als Graphen beschrieben werden. Deshalb besteht die Aufgabe darin, über einen solchen Graphen einen geeigneten Baum aufzuspannen [20].

Kapitel 6.6 stellt ein geeignetes Verfahren zur automatischen Hierarchiebildung vor.

Kapitel 6

Selbsterkennung der Entitäten und Selbstkonfiguration des Kommunikationsnetzes

Nach Wandlung einer Produktionsanlage ist für die steuerungstechnische Funktionsbereitschaft der verschiedenen Anlagenebenen die Kenntnis über die vorhandenen Entitäten (Kap. 5.7) und deren strukturellen Beziehungen eine Voraussetzung. Damit die Steuerungsmodelle der Entitäten miteinander interagieren können, muss zudem für die Anwendung des Adressierungsschemas (Kap. 4.4.1) die hierarchische Beziehung der Entitäten ermittelt werden. Das Schema allein genügt allerdings noch nicht für eine Interaktion zwischen Entitäten, da hierzu eine korrekte Konfiguration der zugrunde liegenden Kommunikationstechnik von Bedeutung ist. Letzteres ist abhängig von den strukturellen Beziehungen der Entitäten.

Kenntnis über Struktur und Hierarchie sowie eine korrekt konfigurierte Kommunikationstechnik sind eine grundlegende Voraussetzung für die steuerungstechnische Funktionsbereitschaft einer Produktionsanlage. Deshalb werden im Folgenden Verfahren vorgestellt, die Selbsterkennung und Selbstkonfiguration ermöglichen (Ein Vorläuferverfahren wurde gezeigt in [31]).

Zur besseren Verständlichkeit werden die Verfahren anhand von Produktionsanlagen, Produktionslinien und -inseln (= Produktionssysteme) beschrieben. Sie beschränken sich jedoch nicht auf Hauptebene eins und zwei des 6-Ebenen$^+$-Modells (Kap. 4.1), sondern gelten für alle Ebenen.

6.1 Isomorphie zwischen Entitäten und Kommunikationsnetzwerk

Dem systemtheoretischen Konzept (Kap. 3.5) folgend sind Entitäten Zwillingselemente. Sie verfügen über Software und Rechnerhardware hinaus auch über eine physische Komponente und damit über physische Strukturschnittstellen. Mit diesen stehen Entitäten untereinander in physischer Beziehung. Zudem sind Entitäten kommunikationstechnisch miteinander verknüpft, so dass ein Informationsaustausch erfolgen kann.

Die nachfolgenden Verfahren zur Anwesenheits- und Strukturerkennung basieren auf der Annahme, dass zwischen den physischen Bindungen und den Kommunikationsverbindungen der Entitäten Isomorphie besteht. Für jede physische Verbindung zwischen zwei Entitäten existiert kommunikationstechnisch eine kabelgebundene Punkt-zu-Punkt-Verbindung. Umgekehrt ist für jede kabelgebundene Verbindung eine physische Verbindung vorhanden (Abb. 6.1). Unter dieser Voraussetzung können *strukturierte* Anlagenbereiche durch das Kommunikationsnetz nachgebildet werden, jedoch nicht *unstrukturierte*, da dort Entitäten miteinander nicht physisch verbunden sind. Beispiele hierfür sind etwa mehrere Produktionssysteme, die über ein Transportsystem verbunden sind oder Produktionsinseln,

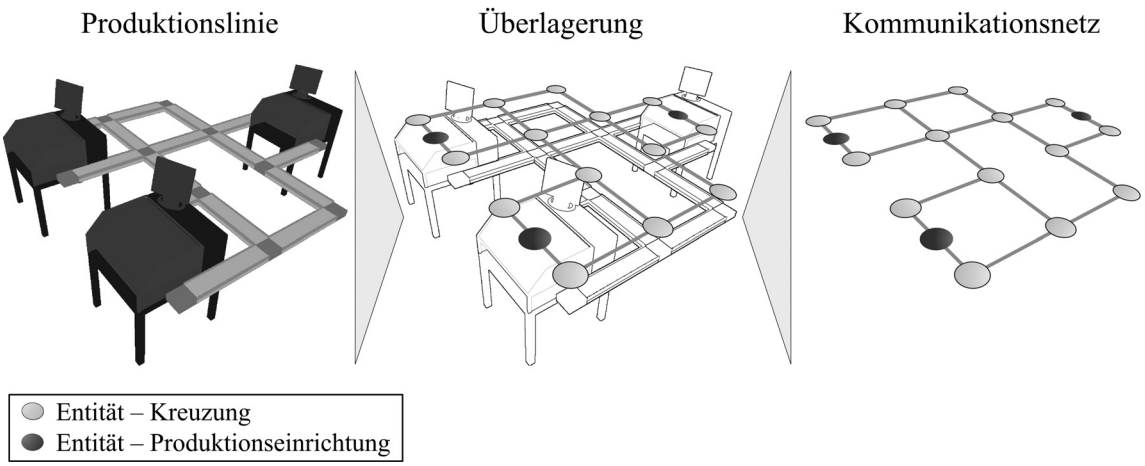


Abbildung 6.1: Isomorphismus zwischen Produktionslinie und kabelgebundenem Kommunikationsnetz.

deren Einrichtungen ebenfalls über ein Transportsystem verbunden sind. In unstrukturierten Bereichen sind Entitäten entweder sternförmig über Switches oder über Funk miteinander gekoppelt.

Eine Entität kann mit mehreren Entitäten in direkter Beziehung stehen, wofür sie über mehrere Kommunikationsschnittstellen verfügen muss. In strukturierten Bereichen ist an jeder Schnittstelle keine oder eine Entität gekoppelt. In unstrukturierten Bereichen können mehrere Entitäten an einer Schnittstelle angeschlossen sein, da Teilnehmer an Switches und in Funknetzen für alle anderen direkt „sichtbar“ sind. Im Folgenden werden die für eine Entität direkt sichtbaren Teilnehmer *Nachbarn* genannt.

Eine Entität und ihre Nachbarn bilden ein sog. *Mikronetz*. Indem alle Mikronetze eines Kommunikationsnetzes übereinander gelagert werden, ist das Gesamtnetz rekonstruierbar (Abb. 6.2). Dadurch, dass das Kommunikationsnetz aufgrund der Isomorphie den physisch-strukturellen Beziehungen zwischen Entitäten gleicht, können wesentliche Teile der Anlagenstruktur automatisiert ermittelt werden.

6.2 Kommunikationstechnische Voraussetzung

Die anlagenweite Selbsterkennung der Teilnehmer und darauf aufbauend alle weiteren Verfahren stellen zwei wesentliche Anforderungen an die Kommunikationstechnik. Erstens ist ein paketorientierter Nachrichtenversand erforderlich und zweitens muss ein *Broadcasting*-Mechanismus vorhanden sein, der einen Nachrichtenversand an unbekannte und daher nicht adressierbare Teilnehmer erlaubt.

Zur Umsetzung sind gesichert Industrie-PCs mit Windows oder Unix-Derivaten als Betriebssysteme geeignet, wenn Kommunikationstechnik eingesetzt wird, die das Internet Protokoll (*IP*) als ISO-OSI-Ebene-3-Protokoll [130] beherrscht. Dadurch sind auch PC-basierte Soft- und Einsteck-SPS (Kap. 2.3.1) geeignet, sofern sie Zugriff auf die Kommunikationsschnittstellen des Gastrechners haben. Als Kommunikationstechnik sind prinzipiell alle Ethernetvarianten aber auch Feldbusse geeignet, sofern sie IP unterstützen (Kap. 2.3.2).

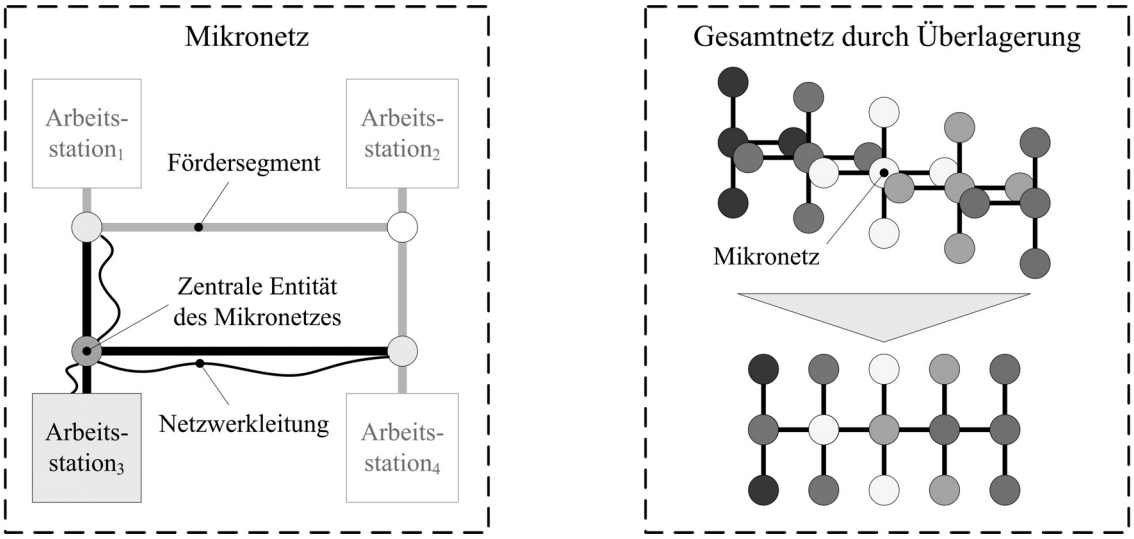


Abbildung 6.2: Links – Mikronetz bestehend aus zentraler Entität über Kommunikationsleitungen mit Nachbarn verbunden. Rechts – Gesamtnetz durch Überlagerung der Mikronetze.

6.3 Dezentrales Verfahren zur Anwesenheitserkennung

Das Verfahren zur Anwesenheitserkennung beruht darauf, dass jede Entität über eine (weltweit) eindeutige Identität *Id* verfügt und diese an alle Nachbarn broadcasted. Nachbarn speichern diese Nachrichten in Teilnehmerlisten, wobei für jede Entität die letzte empfangene Nachricht abgelegt wird. Die Entität sendet ihre Nachricht zyklisch mit einer Wiederholfrequenz f_s, da Nachbarn zu einem späteren Zeitpunkt in die Produktionsanlage integriert werden können. Dadurch kann eine neue Entität ihre Nachbarn detektieren. Jede Entität, die eine Nachricht an einem Eingang empfängt, leitet diese über alle anderen Schnittstellen an ihre Nachbarn weiter. Dadurch erhalten alle Entitäten im System voneinander Kenntnis (Flooding-Prinzip [63]).

Entitäten können auch aus dem System entfernt werden oder ausfallen. Dies muss zuverlässig erkannt werden, was durch die Einführung eines *Timeouts* und eines Zeitstempels möglich wird, mit dem eingehende Nachrichten versehen werden. Zwei Fälle zur Detektion nicht mehr vorhandener Entitäten sind zu unterscheiden. Erstens kann ein lokaler Nachbar ausfallen und zweitens eine entfernte Entität. Eine Entität betrachtet eine Schnittstelle als nicht verbunden, wenn über diese für eine bestimmte Zeitdifferenz Δt keine Nachricht eingetroffen ist. Dadurch ist der Ausfall eines lokalen Nachbarn erkennbar. Der Ausfall entfernter Entitäten wird dadurch erkannt, dass eine Entität die Zeitstempel der in der Teilnehmerliste gespeicherten Nachrichten mit der aktuellen Zeit vergleicht, wenn eine neue Nachricht eintrifft. Alle Entitäten, deren letzte Nachricht länger als Δt zurückliegt, werden als verloren betrachtet und aus der Teilnehmerliste gelöscht. Alle Entitäten des Systems verfügen durch diese Vorgehensweise über den aktuellen Kenntnisstand der Anlagenstruktur.

Ein entstehendes Problem des Verfahrens in unstrukturierten Bereichen besteht darin, dass eintreffende Nachrichten über die weiteren Schnittstellen repliziert werden müssen, damit auch entfernte Entitäten erreicht werden. Das Kommunikationsnetz wird auf diese Weise geflutet, da die entstehende Nachrichtenzahl quadratisch von der Entitätenzahl abhängt. Ein dabei auftretendes Phänomen sind kreisende Nachrichten, die mehrfach bei einem Teilnehmer eintreffen (Abb. 6.3). Diese Nachrichten

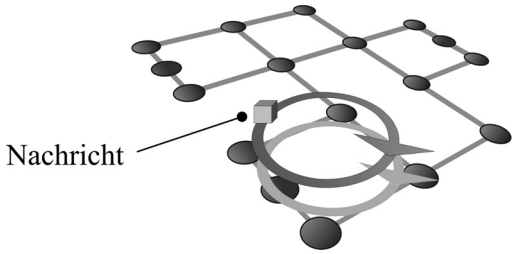


Abbildung 6.3: Aufgrund eines zyklischen Anlagengraphs kreisende Nachricht.

dürfen nicht weiter repliziert werden, da das Kommunikationsnetz aufgrund der stark wachsenden Nachrichtenzahl verstopfen würde.

Zur Erkennung zirkulierender Nachrichten versieht jede Entität ihre eigenen originären (nicht replizierten) Nachrichten mit dem Zeitstempel (*clock*) einer logischen Uhr [53]. Empfänger replizieren eintreffende Nachrichten nur dann, wenn dieser Zeitstempel höher ist, als die entsprechende in der Teilnehmerliste gespeicherte letzte Nachricht der Entität. Hierbei besteht ein weiteres Problem. Wird eine Entität erneut in das System eingebracht (Reparatur, Wartung, etc.), fängt die logische Uhr der Entität von vorne bei Null zu zählen an. Entfällt eine Entität nur kurzzeitig ($\leq \Delta t$) und verfügen die anderen Entitäten noch über eine gespeicherte Nachricht, würden eigentlich neue Nachrichten solange verworfen werden, bis deren logische Zeit größer würde, als die in der alten Nachricht zuletzt gespeicherte. Die Entität wäre somit für eine unbestimmte Zeit unsichtbar. Um dies zu vermeiden wird ein Lebenszähler *lifeCounter* eingeführt. Dieser wird bei jedem Neustart einmalig inkrementiert und dominiert über die logische Uhrzeit.

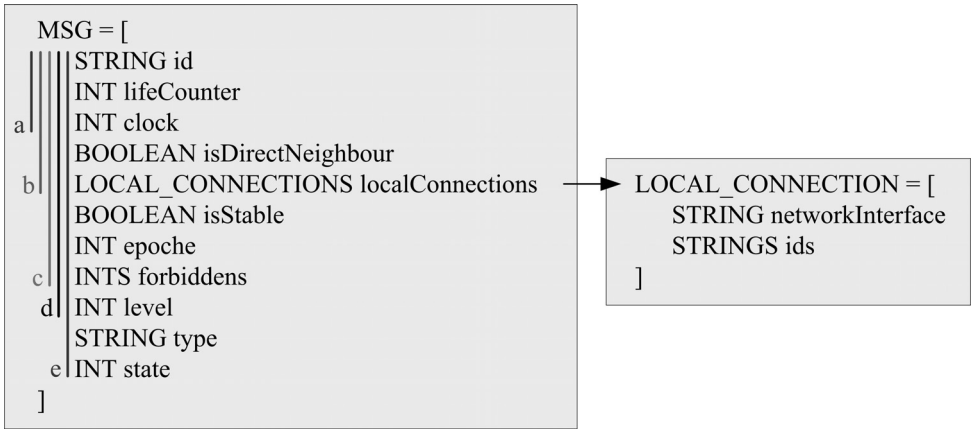


Abbildung 6.4: Nachrichten-Datenstruktur. a – Anwesenheitserkennung. b – Strukturerkennung (Kap. 6.4, S. 81). c – Adressvergabe (Kap. 6.7, S. 88). d – Hierarchiebildung (Kap. 6.6, S. 83). e – Typisierung von Entitäten (Kap. 7.7, S. 113 und Kap. 9.1.2, S. 135).

Abbildung 6.4 (a) zeigt die Datenstruktur der Nachrichten. Durch ihre Erweiterung ist der Austausch von Informationen für weiterführende Aufgaben (z.B. Organisation von Stoffflüssen) möglich. Den Algorithmus zum zyklischen Nachrichtenversand zeigt Abbildung 6.5, den zur Verarbeitung empfangener Nachrichten Abbildung 6.6. Letzterer wird von jeder Entität getrennt für jede vorhandene Kommunikationsschnittstelle ausgeführt.

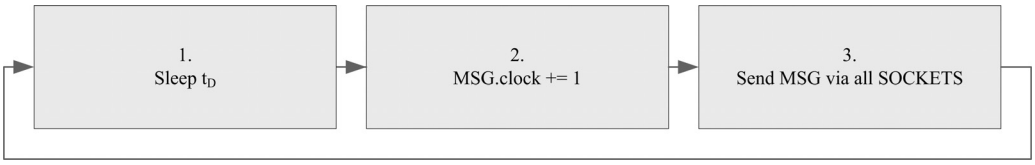


Abbildung 6.5: Zyklische Sendefunktion der Anwesenheitserkennung.

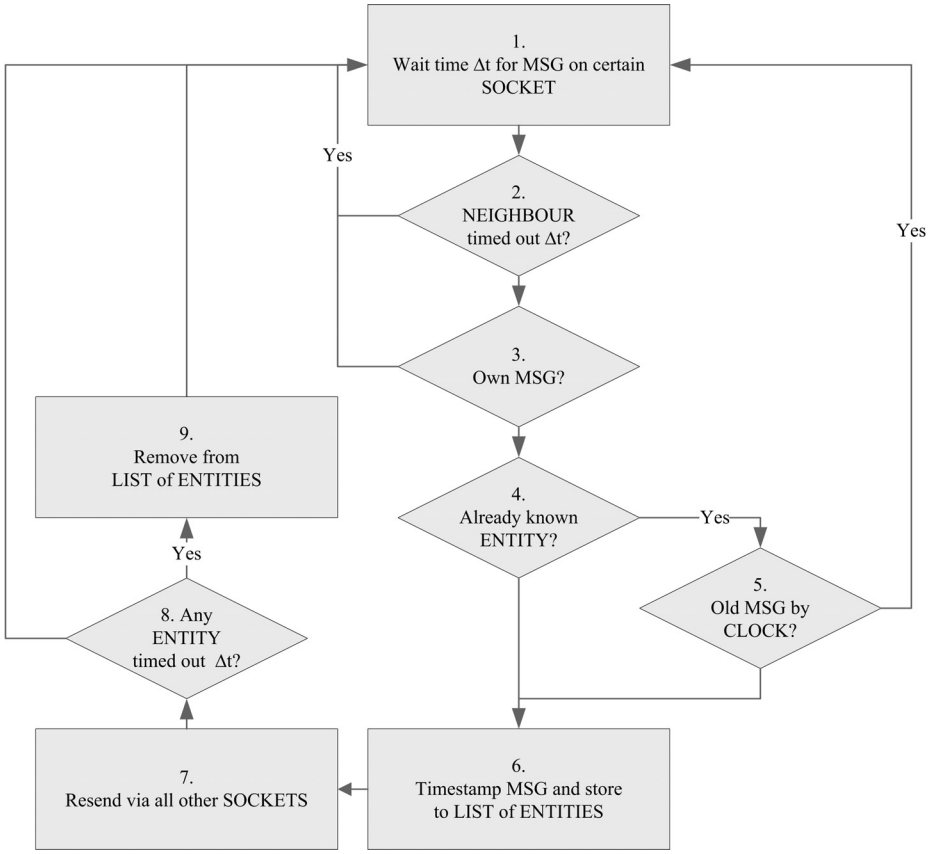


Abbildung 6.6: Zyklische Empfangsfunktion der Anwesenheitserkennung.

Neu hinzu gekommene oder nicht mehr aktive Entitäten werden nach diesem Verfahren unmittelbar (nach einem Sendeintervall bzw. Timeout-Zeit) erkannt, so dass höhere Systemfunktionen umgehend auf Veränderungen reagieren können. Durch das Verfahren können auch mehrere Teilnetze miteinander verschmolzen werden bzw. ein Netz in mehrere Teile partitioniert werden. Alle Entitäten innerhalb der jeweils resultierenden (Teil-)Netze erlangen Kenntnis voneinander.

6.4 Dezentrales Verfahren zur Ermittlung der strukturellen Beziehungen

Das Verfahren zur Anwesenheitserkennung von Entitäten wird derart erweitert, dass die strukturellen Beziehungen zwischen Entitäten dezentral ermittelt werden können. Jede Entität besitzt dadurch Kenntnis von der Gesamtstruktur. Wie in Kapitel 6.1 erläutert, lässt sich die isomorphe Anlagenstruktur durch übereinander gelagerte *Mikronetze* ermitteln, indem jede Entität mit ihren lokalen Nachbarn ein eigenes Mikronetz bildet. Werden Informationen zu solchen Mikronetzen versendet, kann jede Entität die Anlagenstruktur rekonstruieren.

Um dies zu erreichen, müssen zwei Voraussetzungen erfüllt werden. Erstens muss eine Entität die Nachricht eines Nachbarn als „nachbarschaftlich“ identifizieren können, d.h. Nachbarn von Nicht-Nachbarn unterscheiden können. Zweitens müssen die Informationen zu Mikronetzen – wie bereits erwähnt – mit versendet werden.

Für die Identifikation eines Nachbarn wird die Datenstruktur der Nachricht (Abb. 6.4, b) um ein Lokales-Nachbar-Flag (*isDirectNeighbour*) erweitert. Jede Entität versendet ihre eigenen originären Nachrichten stets mit gesetztem Flag. Empfängt eine Entität eine solche Nachricht, erhält sie Kenntnis über einen lokalen Nachbarn. Bei allen *replizierten* Nachrichten, wird das Flag zurück gesetzt. Für die Übertragung des Mikronetzes einer Entität wird die Datenstruktur der Nachricht zusätzlich um eine Liste (*localConnections*) erweitert, die zu jeder Schnittstelle eine Liste von Identitäten direkt angeschlossener Entitäten speichert. Ist die Entität über eine Schnittstelle an einen unstrukturierten Bereich gekoppelt, werden die Identitäten aller Teilnehmer dieses Bereichs dort gespeichert. Ist eine Schnittstelle unbesetzt, so bleibt die Identitätenliste leer.

Gegenüber dem Verfahren zur Anwesenheitserkennung bleibt der Sendealgorithmus gleich. Im Gegensatz dazu ist allerdings die Empfangsfunktion anzupassen, um die Nachbarn zu erkennen (Abb. 6.7).

Neue oder nicht mehr anwesende Nachbarn verändern das Mikronetz einer Entität. Da die aktuellen Informationen zu dem Mikronetz zyklisch versendet werden, werden alle Entitäten zeitnah davon in Kenntnis gesetzt.

6.5 Ansätze zur Reduktion der Nachrichten

Die Verfahren zur Anwesenheits- und Strukturerkennung beruhen auf der Replikation von Nachrichten, die das Kommunikationsnetz fluten. Durch diese Vorgehensweise ist einerseits die dezentrale

Strukturermittlung für alle Entitäten möglich, andererseits steigt die Nachrichtenzahl n in Abhängigkeit der Anzahl der Entitäten e und der maximalen Anzahl an Schnittstellen i einer Entität nach:

(6.1) $n \leq (i + (i-1)(e-1))e \approx O(e^2)$

Da die Nachrichten zyklisch mit Wiederholfrequenz f_s repliziert werden, ergibt sich eine Gesamtzahl von Nachrichten pro Sekunde nach:

(6.2) $n_{sek} \leq f_s\,[(i + (i-1)(e-1))e] \approx O(e^2)$

Angemerkt sei, dass die genannten oberen Grenzen sehr grob sind, da angenommen wird, dass alle Schnittstellen aller Entitäten verbunden sind. Da dies in der Praxis in der Regel nicht der Fall ist (z.B. sind nicht immer alle physischen Schnittstellen der Weichen eines Materialflusssystems verbunden), liegt das tatsächliche Nachrichtenaufkommen in Abhängigkeit der Anlagenstruktur deutlich unter den angegebenen Grenzen. Sie sind ggf. fallabhängig zu bestimmen und lassen sich daher an dieser Stelle nicht angeben. An der grundsätzlichen Komplexität $O(e^2)$ ändert dies allerdings nichts.

Mit zunehmender Größe des Kommunikationsnetzes belegen diese Nachrichten eine zunehmende Bandbreite der Kommunikationstechnik, was durch zwei einfache Ansätze reduziert werden kann. Die erste Möglichkeit besteht offenkundig in der Verringerung der Wiederholfrequenz, in der Entitäten zyklisch originäre Nachrichten erzeugen. Dies geht allerdings unmittelbar mit einer sinkenden Erkennungsrate struktureller Wandlungen einher.

Die zweite Möglichkeit besteht darin, die Replikation von Nachrichten einzuschränken. Indem eine Entität nur Nachrichten repliziert, deren enthaltene Mikronetzinformation von der Information abweicht, die in ihrer Teilnehmerliste gespeichert ist – ein Vergleich ergibt eine strukturelle Wandlung oder eine neue Entität – kann die Nachrichtenzahl stark reduziert werden. Dieser Ansatz birgt jedoch die Gefahr, dass nicht alle Entitäten Änderungen empfangen, da in paketorientierten Kommunikationssystemen Nachrichten verloren gehen können [103]. Durch Einführung eines Faktors r, mit der eine Entität die Replikation von Nachrichten wiederholt, kann dieses Risiko reduziert werden.

Ein weiteres Problem Nachrichten nur noch von strukturell veränderten Mikronetzen zu versenden entsteht, wenn verschiedene Teilnetze mit jeweils mehreren Entitäten verschaltet werden, da nur an den Kopplungspunkten strukturelle Änderungen geschehen. Allerdings müssen alle Entitäten auch von allen Teilnetzen Kenntnis erlangen. Das Problem kann dadurch gelöst werden, dass die Nachbarn einer neuen Entität unmittelbar alle verfügbaren Information zusenden. Da zwei neu gekoppelte Entitäten sich gegenseitig als „Neu“ identifizieren, schicken sie sich dadurch gegenseitig alle Informationen zu.

Durch den zweiten Ansatz sinkt die Anzahl zyklischer Nachrichten pro Sekunde auf:

(6.3) $n_{sek} \leq f_s i \cdot e \approx O(e)$

Die Anzahl an Nachrichten pro strukturell verändertem Mikronetz beträgt:

(6.4) $n \leq (i + (i-1)(e-1))r \approx O(e)$

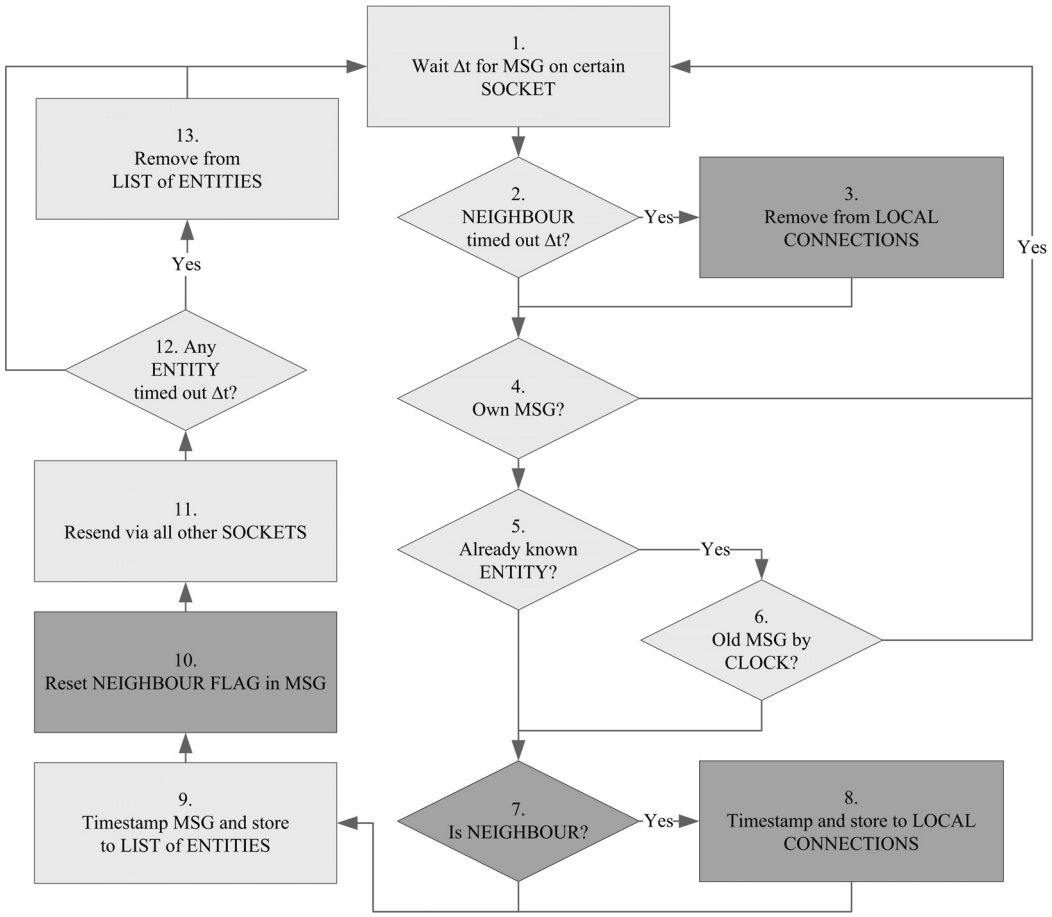


Abbildung 6.7: Zyklische Empfangsfunktion der Strukturerkennung. Dunkler dargestellt sind die Erweiterungen bezogen auf die Empfangsfunktion der Anwesenheitserkennung (Abb. 6.6, S. 80).

Das Nachrichtenaufkommen sinkt durch diesen Ansatz zwar deutlich, allerdings gehen mit sehr großen Kommunikationsnetzen auch steigende Nachrichtenlaufzeiten durch zeitbehaftete Replikation einher. Bis alle Entitäten über eine Änderung informiert sind, muss, im ungünstigsten Fall einer linearen strukturellen Verkettung aller Entitäten, eine Nachricht $e - 1$ mal nacheinander repliziert werden. Wächst der Zeitbedarf für die Benachrichtigung aller Entitäten über den Timeout Δt zur Erkennung von Entitätenausfällen (Kap. 6.3) kann es passieren, dass sich zu einer Entität gleichzeitig mehrere gültige Mikronetzinformationen in der Verbreitung befinden. Es ist davon auszugehen, dass sich Nachrichten überholen können [103]. Allerdings ist dies nicht weiter problematisch, da die logische Uhrzeit einer Nachricht ihre Aktualität widerspiegelt und veraltete Nachrichten verworfen werden.

6.6 Automatische Hierarchiebildung von Entitäten

Damit Steuerungsmodelle von Entitäten miteinander interagieren können, ist die Anwendung des in Kapitel 4.4.1 dargestellten Adressierungsschemas notwendig. Dazu sind die Entitäten in eine sinnvolle hierarchische Beziehung zu bringen, indem jeder Entität ein Vater und jedem Vater seine Kinder

zugeordnet werden. Dabei wird der Wurzelknoten eines Entitätensubbaums mit keinem oder einem übergeordneten Wurzelknoten und keinem, einem oder mehreren untergeordneten Wurzelknoten anderer Entitätensubbäume verknüpft. Die Aufgabe besteht also in dem Aufspannen eines Baums auf Grundlage der strukturellen Beziehungen der Entitäten (Kap. 5.7).

6.6.1 Separatoren zur Hierarchiebildung

Mit dem vorangegangenen dezentralen Verfahren zu Strukturerkennung, das auf dem dezentralen Verfahren der Anwesenheitserkennung fußt, sind die ersten Schritte zur Hierarchiebildung getan. Abbildung 6.8 zeigt dies in den Schritten a) und b). Die nächste Aufgabe besteht in der Zuordnung der Entitäten zu Gruppen, also zu Linien, Inseln, Zellen, Geräten, Modulen, Sensoren und Aktoren. Diese ist nicht ohne Zusatzwissen zu realisieren, denn die Gruppeninformation ist nicht Bestandteil der Strukturinformation. Eine Möglichkeit besteht darin, jeder Entität manuell die Gruppenzugehörigkeit vorzugeben. Allerdings werden bei jeder Positionsänderung oder dem Hinzufügen einer Entität manuelle Konfigurationsschritte erforderlich. Deshalb scheidet dieser Ansatz aus. Stattdessen wird das Konzept der *Separatoren* eingeführt.

Ein Separator ist ein Marker, der anzeigt, dass eine Entität, über ihre Hauptfunktion hinaus, auch eine Grenze zweier Hierarchiestufen repräsentiert. Auf die Struktur bezogen, teilen Separatoren diese in Zonen ein. Diese Zonen sind ineinander geschachtelt (aufgrund der Hierarchie von Produktionsanlagen – zur Verdeutlichung siehe Abb. 5.24, S. 74). Ein Separator allein ist jedoch ungenügend, da für die Schachtelung eine Zusatzinformation fehlt – die Angabe einer numerischen Hierarchiestufe h. Den Schlussfolgerungen zur Gestaltung der Wandelbarkeit (Kap. 3.4) ist zu entnehmen, dass eine Produktionsanlage über beliebig viele Hierarchiestufen verfügen kann. Deshalb wird als Wertebereich für die Hierarchiestufen $h \in \mathbb{N}_0^-$ gewählt. Die Schritte c) und d) der Abbildung 6.8 verdeutlichen die Notwendigkeit der Separatoren.

Die Angabe der Hierarchiestufe ist ein manueller Schritt, der durch Erzeugung eines Stufensystems in vielen Fällen auf die Entwicklungszeit verschiedener Entitätentypen vorgelagert werden kann, so dass eine Konfiguration während des Betriebs entfällt. Ein solches System wird dadurch realisiert, dass der Wertebereich in Intervalle zerlegt wird und jedem Intervall bestimmte Entitätentypen zugeordnet werden. Indem bspw. für den Übergabebahnhof einer Produktionslinie zur Transportfahrzeuganbindung fest die Stufe *-1* vergeben wird, markiert dieser Bahnhof in unterschiedlichen Linien stets die Grenze zwischen Produktionsanlage und Produktionslinie.

6.6.2 Verfahren zur Ermittlung der Zonenzugehörigkeit

Indem sich jede Separatorentität über die eigenen Nachrichten zur Strukturerkennung als Separator erkennbar macht und ihre Hierarchiestufe mit versendet, sind die Separatoren in der Gesamtstruktur verzeichnet. Hierzu ist eine Erweiterung der Datenstruktur der Nachrichten um ein Stufenfeld (*level*) erforderlich (Abb. 6.4, d, S. 79). In Nachrichten von Separatoren sind hier Werte $level \leq -1$ und von Entitäten $level = 1$ eingetragen. Die Aufgabe jeder Entität besteht nun darin, aus diesen Informationen selbstständig die Zonenzugehörigkeit aller Entitäten, d.h. deren jeweiligen Väter, zu ermitteln.

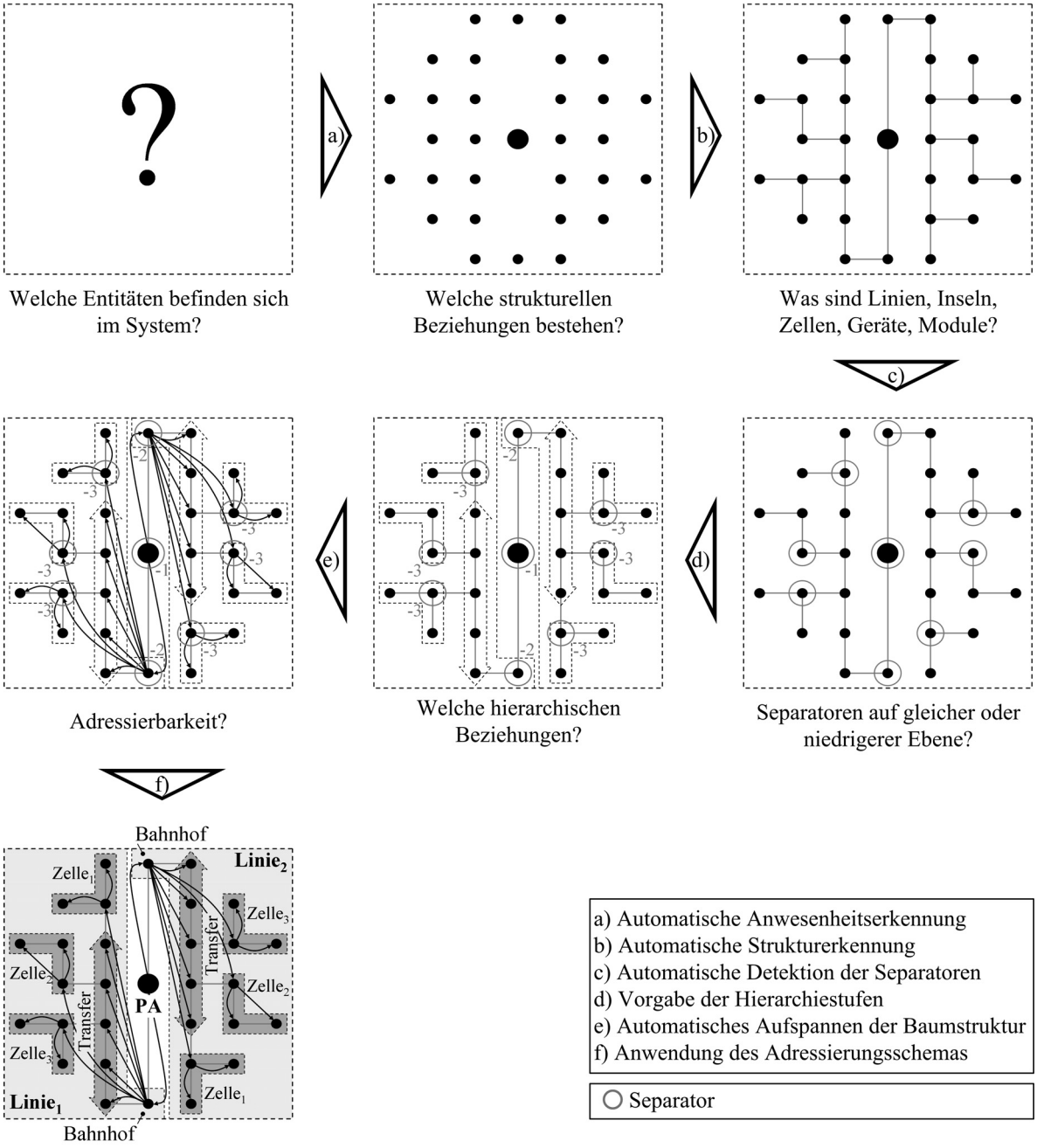


Abbildung 6.8: Schrittweise Vorgehensweise der Selbsterkennung und -konfiguration. PA = Produktionsanlage.

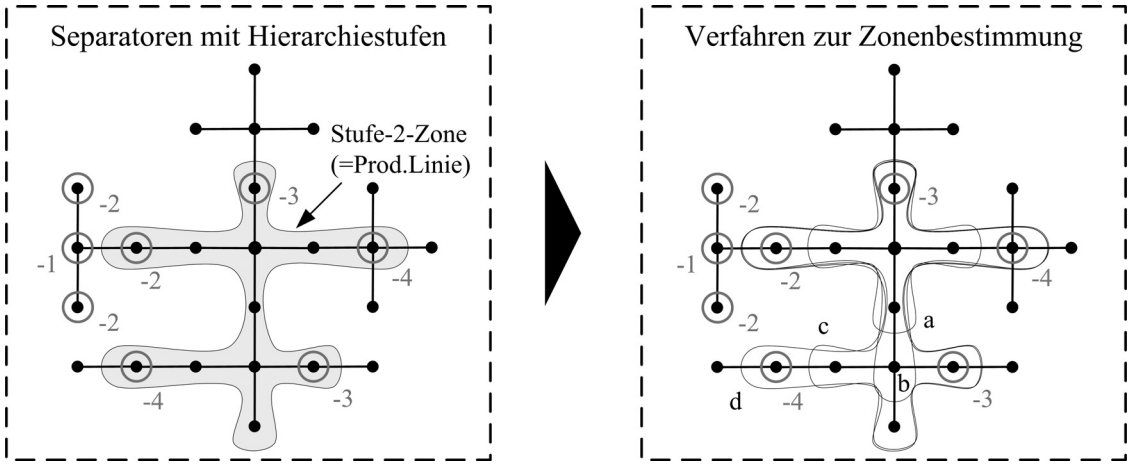


Abbildung 6.9: Lösungsprinzip zur selbstständigen Erkennung der Entitäten einer Zone und des Vaters. Die Kinder eines Separators sind alle anderen Entitäten der Zone.

Für Separatoren sind darüber hinaus die Kinder zu bestimmen, zu denen alle Elemente einer Zone zählen, der ein Separator vorsteht.

Zonen können dadurch erkannt werden, dass ausgehend von einer gegebenen Entität in einer sich wellenförmig ausbreitenden Suche auf der Anlagenstruktur, alle direkt und indirekt benachbarten Entitäten gesucht werden. Die Welle wird entweder durch Separatoren gestoppt oder hält an, wenn eine Entität keine Nachbarn besitzt. Abbildung 6.9 zeigt ein Szenario auf Ebene der Produktionslinien. Indem sukzessive für alle Entitäten, deren Zonenzugehörigkeit unbekannt ist, die Zugehörigkeit ermittelt wird, können alle Zonen bestimmt werden. Abbildung 6.10 zeigt hierzu einen Algorithmus, der als Funktion aus der Strukturerkennung heraus aufgerufen wird. Da einer Zone mehrere Entitäten vorstehen können (z.B. mehrere Übergabebahnhöfe an einer Linie) wird in einem solchen Fall die Entität mit der niedrigsten Identität Vater, da aufgrund der Eindeutigkeit der Identitäten eine Ordnungsrelation besteht. Mit diesem Algorithmus ist der in Abbildung 6.8 dargestellte Schritt e) durchführbar.

6.6.3 Umsetzung des Adressierungsschemas

Die so realisierte Hierarchie der Entitäten ist für die Anwendung des Adressierungsschemas geeignet. Allerdings sind hierzu noch zwei Erweiterungen erforderlich.

Eine Separatorentität kann neben der Separatorfunktion eine Hauptfunktion ausüben. Beide müssen semantisch nicht deckungsgleich sein, wie das bereits genannte Beispiel des Übergabebahnhofs einer Produktionslinie verdeutlicht. Als Separator repräsentiert er eine Linie, in der Hauptfunktion schlägt er Stoffe um, realisiert also lediglich eine Teilfunktionalität der Linie. Auf der anderen Seite zeigt das Beispiel eines Roboters semantische Deckungsgleichheit. Dieser kann über eine Hauptsteuerung verfügen, die ihn als Separator nach außen repräsentiert und in ihrer Hauptfunktion auch steuert, indem sie die einzelnen Achsen koordiniert. Damit sich Separatorfunktion und Hauptfunktion in der Adressierungshierarchie spiegeln, werden bei semantischer Ungleichheit zwei verschiedene Namen (z.B. „Linie_1“ und „Übergabebahnhof_1“) benötigt. Dadurch ist im Bedarfsfall eine zweifache Einordnung der Entität in das Adressierungsschema möglich (Abb. 6.11). So ist der letzte Schritt f) aus

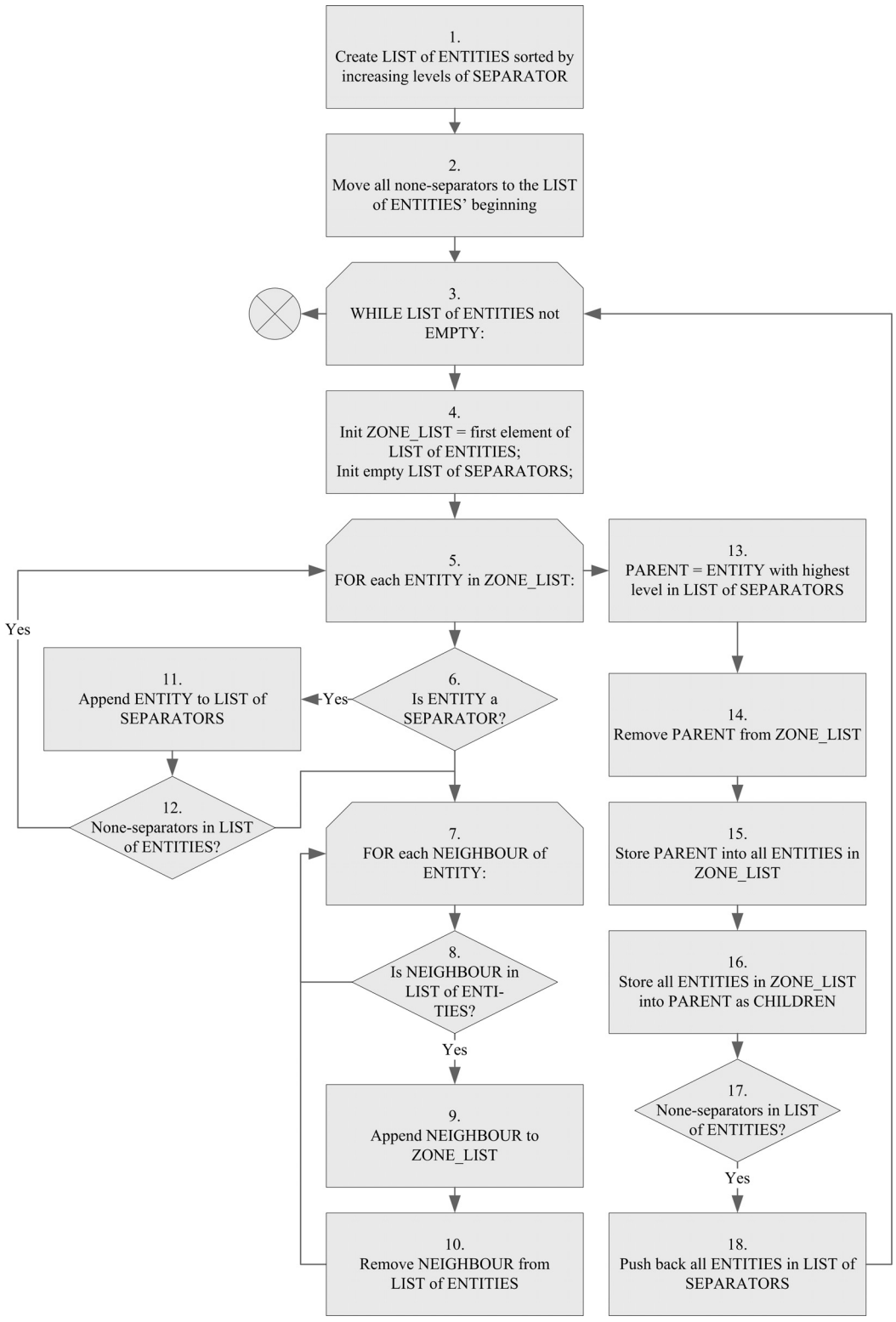


Abbildung 6.10: Algorithmus zur Zuordnung aller Entitäten in Zonen. Jede Entität vermerkt ihren Vater, jeder Separator darüber hinaus seine Kinder. Anmerkung: Durch das Anhängen neuer Entitäten in Schritt 9 wächst die Liste der Zonenentitäten.

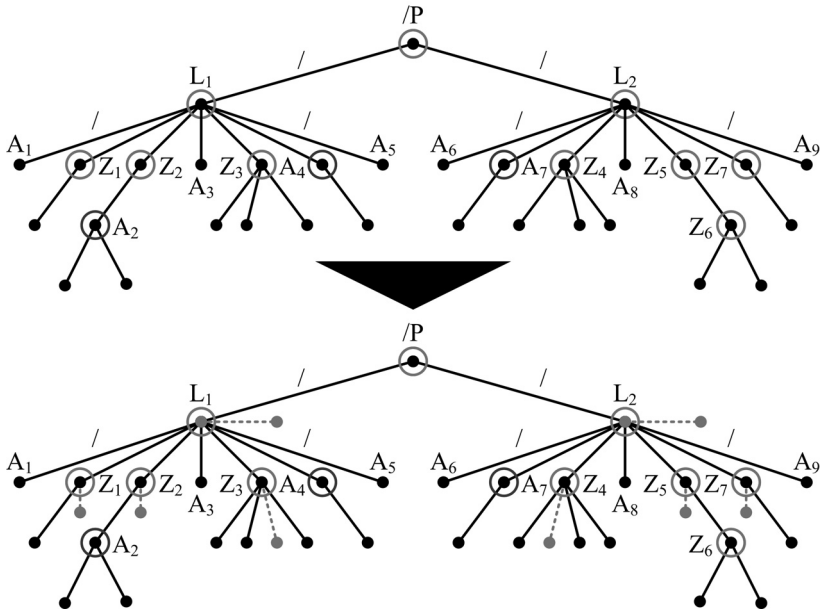


Abbildung 6.11: Zweifache Einordnung von Separatorentitäten nach Separator- und Hauptfunktion in die hierarchische Adressierungsstruktur. A_l = Arbeitsstation; L_m = Linie; P = Produktionsanlage; Z_n = Zelle.

Abbildung 6.8 vollzogen und die vollständige Adressierung aller Steuerungsmodelle einer Produktionsanlage möglich.

6.7 Selbstkonfiguration des Kommunikationsnetzes

Ein wesentlicher Bestandteil für die Funktionsbereitschaft einer Produktionsanlage ist ein konfiguriertes Kommunikationsnetzwerk. Das Verfahren zur Strukturerkennung erkennt die kommunikationstechnischen Beziehungen zwischen Entitäten und liefert damit wichtige Informationen für die Konfiguration. Die Aufgaben sind dabei an jeder Entität Adressen zu vergeben und Routentabellen zur Nachrichtenweiterleitung zu erstellen. Im Folgenden wird das Verfahren zur Strukturerkennung (Kap. 6.4) derart erweitert, dass diese Aufgaben dezentral und deterministisch erfüllt werden.

Adressen können nicht fest an Entitäten vergeben werden, da der zu Verfügung stehende Adressraum als begrenzt angesehen werden muss. Es sind Szenarien denkbar, in denen in einer Fabrik eine größere Anzahl von Entitäten vorhanden ist, *als* Adressen verfügbar sind. Maximal können sich allerdings nur so viele Entitäten in Benutzung befinden, *wie* Adressen verfügbar sind. Durch Umrüstvorgänge kann es zu einer Fluktuation von Entitäten-Adressen-Zuordnungen kommen, so dass eine Entität im Laufe ihres Lebenszyklus verschiedene Adressen erhält.

Hauptproblem der Adressvergabe ist die Vermeidung von Ambivalenzen – zwei Entitäten dürfen *nie* die gleiche Adresse besitzen. Da das Verfahren dezentral funktionieren muss, ist ein deterministischer Ansatz erforderlich, nachdem eine Entität ihre und auch die Adressen aller anderen Entitäten im System ermitteln kann. Der Lösungsansatz beruht darauf, dass zwischen den Identitäten (*Ids*) der

Entitäten aufgrund ihrer Eindeutigkeit eine Ordnungsrelation besteht. Eine Identität ist zu allen anderen Identitäten immer entweder größer oder kleiner. Dadurch kann eine Liste mit Identitäten immer eindeutig in aufsteigender Reihenfolge sortiert werden. Indem jeder Identität aus einer sortierten Liste anhand ihres Listenindex eine Adresse aus einer ebenfalls sortierten Adressliste zugeordnet wird, ist eine eindeutige Adressvergabe dezentral möglich.

Ein Phänomen resultierend durch Wandlung ist, dass sich der Identitätsindex einer Entität ändert, wenn eine oder mehrere Entitäten mit niedrigerer Id ausfallen oder neu hinzukommen. Im ersten Fall erniedrigt sich der Index, im zweitem Fall erhöht er sich. Daraus folgt, dass sich die Adresse einer Entität über die Zeit mehrfach ändern kann. Dies hat Auswirkungen auf die Kommunikation. Kommunikationspartner müssen eine sich ändernde Adresse ermitteln können und ihre Verbindungen dann selbstständig entsprechend der neuen Situation anpassen. Diese Umstellung der Kommunikation führt zu Unterbrechungen in der Kommunikationsfähigkeit der Fabrik. Da Kommunikation für die Produktion in rechnergestützten Anlagen essenziell ist, müssen Häufigkeit und Dauer der Unterbrechungen möglichst minimiert werden.

Zur Lösung dieses Problems wird ein Phasenmodell mit zwei Zuständen eingeführt. Jedes Kommunikationsnetz befindet sich in einer der beiden Phasen *stabil* oder *instabil*. Es ist stabil, wenn keine Umbaumaßnahmen erfolgen und jede Entität (kommunikationstechnisch) funktionsbereit ist. Es wird instabil sobald eine Entität ausfällt, entfernt oder hinzugefügt wird. Jede Entität erkennt autonom durch die Erkennung von Strukturveränderungen in Mikronetzen diesen Zustandswechsel und markiert für sich eine instabile Phase.

Die Erkennung eines wieder stabilen Kommunikationsnetzes erfolgt durch den Einsatz eines Timers, der bei Wechsel in den instabilen Zustand von jeder Entität autonom gestartet wird und rückwärts abläuft. Bei jeder neu erkannten Strukturveränderung wird der Timer zurückgesetzt und erneut gestartet. Erst wenn sich über einen gewissen Zeitraum keine Änderungen mehr ergeben, läuft der Timer ab und die Entität markiert für sich einen Wechsel in die stabile Phase. Dadurch werden mehrere Veränderungen innerhalb eines freiwählbaren Zeitraums als ein einziger Wandlungsvorgang betrachtet.

Alle Entitäten wechseln innerhalb einer gewissen Zeitspanne, die von der Länge des Timers abhängig ist, „quasi-synchron“ zurück in den stabilen Zustand. Bei diesem Wechsel berechnet jede Entität für sich die neuen Adressen aller Teilnehmer und konfiguriert ihre Netzwerkschnittstellen um. Zusätzlich werden Routing-Tabellen für die Paketweiterleitung neu erstellt, so dass sich die Entität wieder in einem kommunikationsbereiten Zustand befindet. Dadurch ist die Zeitdauer, in der keine Kommunikation möglich ist, auf die oben genannte Zeitspanne begrenzt.

Bisher gewährleistet der Ansatz, dass in jeder stabilen Phase Adressen konfliktfrei vergeben sind. Er verhindert allerdings nicht, dass die Adresse einer Entität während der (kurzen) Zeitspanne des Wechsels in den stabilen Zustand doppelt genutzt wird (eine Entität ist bereits wieder mit neuer Adresse stabil, eine zweite Entität mit bisher der gleichen Adresse noch nicht). Ein Sender kommuniziert dann womöglich mit dem falschen Empfänger, was je nach Protokoll auf ISO-OSI-Ebene-7 (Anwendungsschicht) zu unerwünschten Effekten führen kann. Um diese Situation zu vermeiden wird der zur Verfügung stehende Adressraum in disjunkte Intervalle gleicher Größe zerlegt und sog. *Epochen* eingeführt. Eine Epoche repräsentiert eine stabile Phase und ein Epochenwechsel markiert den Übergang von einer stabilen Phase zur nächsten. Jeder Epoche wird ein eigenes Adressintervall zugeteilt, wobei die Regel gilt, dass in zwei aufeinander folgenden Epochen nie das gleiche Adressintervall

verwendet wird. Dadurch ist es möglich, die geschilderte Problematik der kurzzeitigen Ambivalenz zu umgehen.

Das Epochenkonzept ist jedoch nicht ohne weiteren Aufwand umzusetzen. Es muss möglich sein, mehrere Teilnetze, die sich in unterschiedlichen Epochen befinden, zusammenzuschalten, so dass sich das daraus resultierende Gesamtnetz in einer einheitlichen Nachfolgeepoche befindet. Die Anzahl der pro Epochenwechsel maximal konfliktfrei zusammenschaltbaren Teilnetze (eine einzelne Entität ist ebenfalls ein eigenes Teilnetz) ist gleich der Anzahl möglicher Adressintervalle minus eins. Um die jeweiligen Epochen der Teilnetze gegenseitig erkennen zu können, ist es erforderlich, die Nachrichtenstruktur um einen Epocheneintrag (*epoche*) zu erweitern (Abb. 6.4, c, S. 79).

Um mehrere Teilnetze erfolgreich zusammenschalten zu können, werden, sobald das Netz instabil wird, alle aktuellen Epochen der verschiedenen Teilnetze von den Entitäten in einer Liste verbotener Nachfolgeepochen gesammelt. Sobald das neue Gesamtnetz wieder in die stabile Phase zurückkehrt, wird die erste nicht verbotene Epoche durch Abgleich mit dieser Liste ausgewählt. Allerdings wechseln die Entitäten nicht vollkommen zeitsynchron zurück. Während ein Teil der Entitäten bereits stabil ist, ist der Rest noch instabil. Es muss vermieden werden, dass instabile Entitäten, aufgrund empfangener Nachrichten bereits stabiler Entitäten, diese stabilen und neuen Epochen in die Verbotsliste eintragen. Andernfalls würden die neuen Epochen der einzelnen Entitäten auseinander driften und infolgedessen die Adressvergabe scheitern. Deshalb wird die Nachrichtenstruktur um ein Ist-Stabil-Flag (*isStable*) ergänzt (Abb. 6.4, c, S. 79), so dass nun eine Unterscheidung vorgenommen werden kann. Nur wenn eine empfangene Nachricht als instabil registriert wird, erfolgt eine Aufnahme der mitgesendeten Epoche in die Verbotsliste.

In diesem Zusammenhang gibt es ein weiteres Phänomen – die *Epochenfalle*. Wird bspw. eine Entität in Epoche Ep' zunächst mit einer zweiten Entität in Epoche Ep'' gekoppelt, Letztere während der daraus resultierenden instabilen Phase wieder abgekoppelt und in der gleichen instabilen Phase mit einer dritten Entität in Epoche Ep''' verbunden, so laufen die Epochen der zweiten und dritten Entität zwangsweise auseinander (Abb. 6.13). Die Ursache liegt in unterschiedlichen Verbotslisten. Dies kann vermieden werden, wenn jede Entität ihre Verbotsliste in der Nachricht mitschickt und die Verbotslisten empfangener Nachrichten mit der eigenen vereinigt. Die Datenstruktur der Nachrichten wird deshalb um eine Verbotsliste (*forbiddens*) erweitert (Abb. 6.4, c).

Zur algorithmischen Umsetzung der Selbstkonfiguration sind mehrere Erweiterungen bei der Strukturerkennung erforderlich. Bei der Empfängerfunktion umfassen sie die Detektion instabiler Phasen sowie die Aktualisierung der Epochenverbotsliste (Abb. 6.12). Die Initiierung bzw. die Verlängerung der instabilen Phase erfolgt über die Instabilfunktion (Abb. 6.14). Das Berechnen der nächsten Epoche sowie der Entitäten-Adressen, das Konfigurieren des Routers und der Netzwerkschnittstellen und das Zurücksetzen des Ist-Stabil-Flags der Entität erfolgt in der Stabilfunktion (Abb. 6.15).

6.8 Integration in das übergeordnete Kommunikationsnetz der Fabrik

Zur kommunikationstechnischen Integration der Produktionsanlage in das übergeordnete Kommunikationsnetz der Fabrik ist der Einsatz eines Gateways erforderlich. Dabei ist ein – für Entwickler, wie Betreiber - möglichst transparenter Datenaustausch zu realisieren. Grundsätzlich kann dies nach den

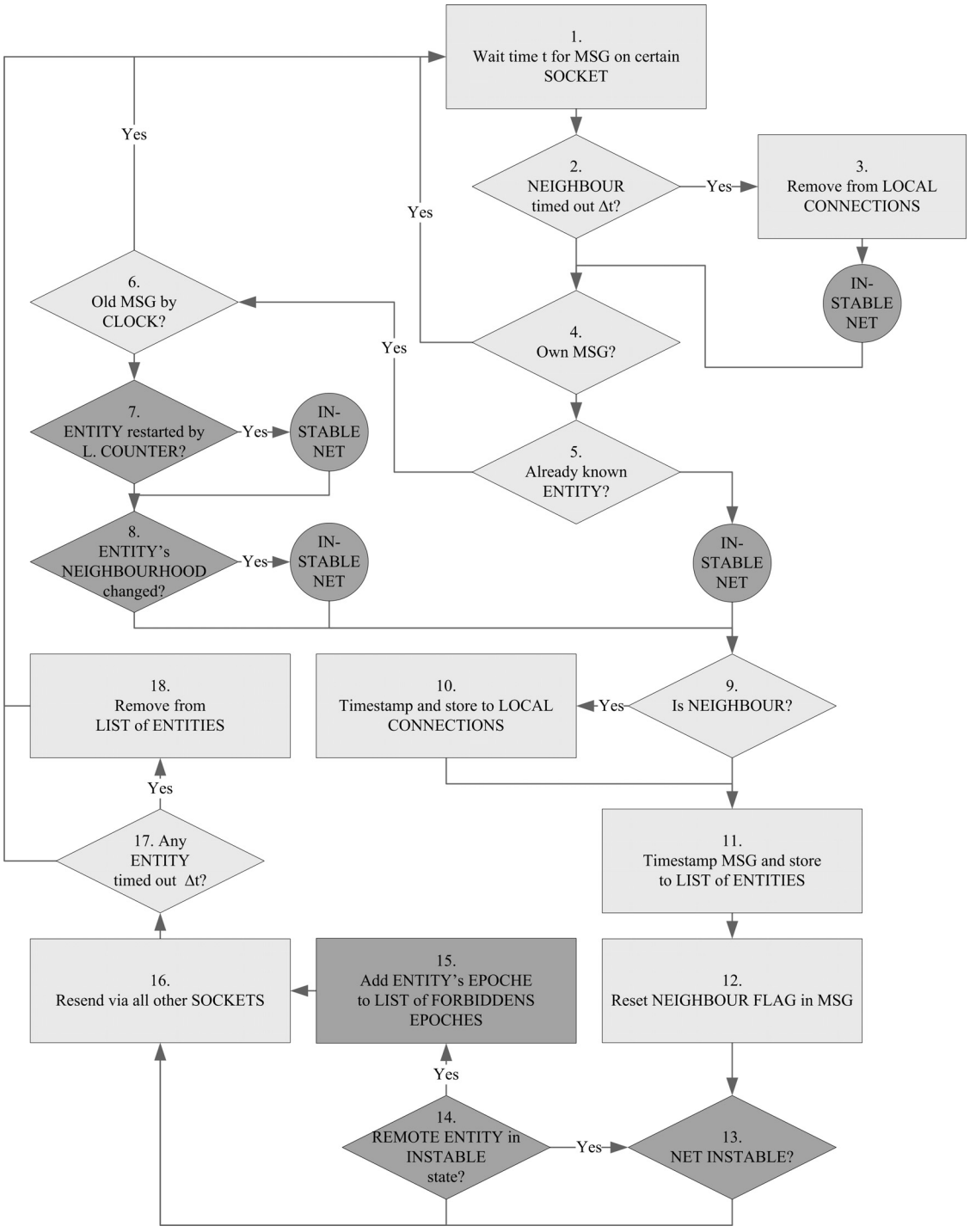


Abbildung 6.12: Zyklische Empfangsfunktion der Strukturerkennung mit Adressvergabe. Dunkler dargestellt sind die Erweiterungen bezogen auf die Empfangsfunktion der Strukturerkennung (Abb. 6.7, S. 83).

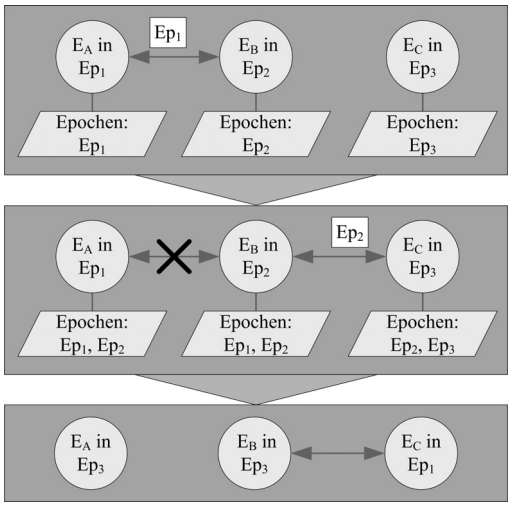


Abbildung 6.13: Epochenfalle – Wird nur die eigene Epoche versendet, divergieren im ungünstigsten Fall die Epochen. Dieser Effekt wird erst durch das Mitversenden einer Epochenverbotsliste vermieden.

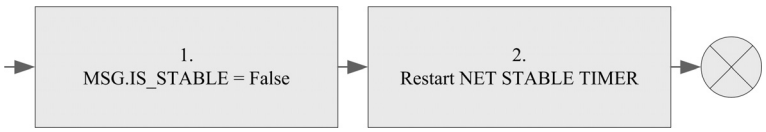


Abbildung 6.14: Funktion zur Initiierung oder Verlängerung der instabilen Phase.

Prinzipien „Briefkasten“ (*Mailboxing*) und „Nachrichtenweiterleitung“ (*Packet Forwarding*) auf den Ebenen drei und sieben des ISO-OSI-Modells [130, 102] erfolgen. Das an dieser Stelle favorisierte Prinzip ist die für Entwickler und Betreiber transparent gestaltbare Nachrichtenweiterleitung auf ISO-OSI-Ebene-3.

Forwarding basiert auf dem transparenten Umadressieren von Nachrichtenpakten auf ISO-OSI-Ebene-3 [52]. Hierzu wird im Folgenden das Kommunikationsnetz der Produktionsanlage als *intern* und die übergeordneten Netze als *extern* bezeichnet. Das Gateway legt für jede Entität eine statische *externe Adresse* an und führt zwei Verzeichnisse (Lookup-Table) mit Abbildungen von externen nach internen Adressen bzw. umgekehrt, die nach Wandlungsvorgängen aktualisiert werden. Zusätzlich verfügt das Gateway über mindestens eine interne Adresse.

Ein externer Teilnehmer adressiert die Nachricht an die externe Entitätenadresse. Treffen Pakete am Gateway ein, ändert diese die Empfängeradresse auf die interne Entitätenadresse und schickt sie ins interne Netz weiter. Die Routing-Tabellen sind so zu konfigurieren, dass Antwortpakete der Entität über die interne Adresse wieder am Gateway eintreffen. Daraufhin ändert das Gateway die Absenderadresse auf die externe Entitätenadresse und leitet das Paket weiter zum externen Teilnehmer. Auf diese Art kann auch die Entität die Initiative ergreifen und mit externen Teilnehmern proaktiv in Verbindung treten. Abbildung 6.16 verdeutlicht den Vorgang der Umadressierung.

Zwei Fragen ergeben sich bei der Nachrichtenweiterleitung. Erstens, wie erhalten externe Teilnehmer die externen Adressen der Entitäten? Zweitens, wie ist mit einem begrenzten Vorrat an externen Adressen umzugehen? Als Antwort auf die erste Frage besitzt das Gateway eine Informationsschnittstelle

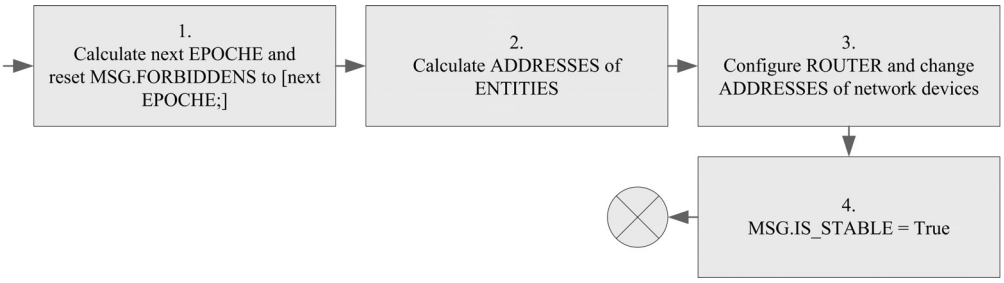


Abbildung 6.15: Funktion zur Behandlung eines Phasenwechsels in den Zustand *stabil*.

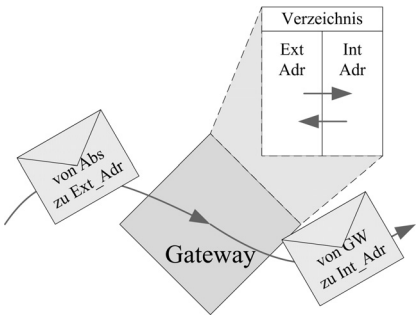


Abbildung 6.16: Transparente Nachrichtenweiterleitung auf ISO-OSI-Ebene-3.

zur Auskunft über Entitäten, die von externen Teilnehmern abgefragt werden kann. Das Problem, welches sich mit der zweiten Frage ergibt, besteht darin, dass Entitäten mit der Zeit der Produktionsanlage hinzugefügt oder aus dieser entfernt werden können, so dass externe Adressen bei begrenztem Vorrat mehrfach verwendet werden müssen. Dies kann gelöst werden, indem Adressen von nicht mehr vorhandenen Entitäten für eine gewisse Zeitspanne gesperrt werden, um lediglich temporäre Ausfälle zu überbrücken. Erst nach Verstreichen dieses Zeitraums werden Adressen bei Bedarf neu vergeben. Über die Informationsschnittstelle des Gateways sind diese Änderungen von Außen abfragbar.

Kapitel 7

Selbstorganisation von Stoffflüssen

Für die Organisation von Stoffflüssen sind je nach Art des zu transferierenden Stoffs unterschiedliche Informationen erforderlich. Für Werkstücke sind Struktur, Arbeitspläne und Wissen über die Fähigkeiten der Produktionseinrichtungen, für die restlichen Stoffe Struktur und Zielort von Bedeutung. Anders ausgedrückt „stellen sich“ Werkstücke die Fragen „was ist zu tun?“ (Arbeitsplan), „wer kann es?“ (Fähigkeiten) und „wie kommt man dort hin?“ (Struktur) und der Rest „wer ist der Empfänger?“ und „wie kommt man dort hin?“ (Struktur).

Mit der Kenntnis über die Anlagenstruktur, die Entitäten mit Hilfe der vorgestellten Verfahren zur Strukturerkennung (Kap. 6.4) selbstständig erhalten, ist ein grundlegender Schritt in Richtung Selbstorganisation vollzogen.

Unter den Annahmen, dass eine Arbeitsplanverwaltung (Kap. 4.6.3) vorhanden ist, die Informationen zur Werkstückbearbeitung bereitstellt, Werkstücke identifizierbar sind und dadurch Produkt, Variante und Bearbeitungszustand abgeleitet werden können, sowie Produktionseinrichtungen Auskunft über ihre Fähigkeiten geben, sind alle Informationen im Gesamtsystem für eine Selbstorganisation der Werkstückflüsse vorhanden. Unter den weiteren Annahmen, dass die restlichen Stoffflüsse zwischen Lagern und Produktionseinrichtungen bestehen und Lagerverwaltungen vorhanden sind, mit denen Produktionseinrichtungen zum Zwecke der Ein- und Auslagerungen interagieren können, sind auch hier alle Voraussetzungen für eine Selbstorganisation gegeben.

Nachfolgend werden Verfahren zur Selbstorganisation von Stoffflüssen vorgestellt, die für den Einsatz in strukturierten und unstrukturierten Bereichen (Kap. 4.2) effektiv geeignet sind. Sie werden von einem generischen Steuerungsmodell zur Ablaufsteuerung von Verzweigungen ausgeführt, das in realen Förder- aber auch Transportsystemen eingesetzt werden kann. Um bei der Steuerung Generizität zu erzielen und damit die Wiederverwendbarkeit zu erhöhen, geht der Modellbeschreibung eine geeignete Klassifikation von Transfersystemen, eine Erläuterung des Funktionsprinzips der Wegfindung sowie eine Erläuterung des Funktionsprinzips von Verzweigungen voraus. Die Verfahren zur Selbstorganisation von Stoffflüssen wurden in einer Simulationsstudie untersucht, deren Experimentgestaltung und Ergebnisse vorgestellt werden. Abschließend wird die am besten geeignete Variante ausgewählt und ihre Integration in das Konzept der wandelbaren Produktionsanlage beschrieben.

7.1 Klassifikation von Transfersystemen

In der Produktion sind eine Vielzahl von Transfertechniken (Kap. 2.1.1) etabliert, etwa das Fördern durch Reibschluss, in Form von gurt-, ketten- oder rollengetriebenen Fördereinrichtungen, oder das

Transportieren mit Hilfe von Transporteuren und Fahrzeugen wie Hubwagen, Ameisen, Gabelstaplern, Wagen, usw. Aus technologischer Sicht können diese Prinzipien in die beiden Hauptkategorien *spurgebundene* und *freinavigierende* Transfersysteme eingeteilt werden [34].

Spurgebundene Systeme sind durch Transfernetze, bestehend aus Transfersegmenten, Verzweigungen und Arbeitsstationen (= Produktionseinrichtungen möglicherweise im Verbund) charakterisiert. Transfernetze entsprechen damit einem Graph mit Verzweigungen und Arbeitsstationen als *Knoten* sowie Transfersegmenten als *Kanten*. Neben den Reibschlussfördersystemen zählen zu dieser Kategorie alle Formen schienen- bzw. leitliniengebundener Fahrzeuge.

Freinavigierende Systeme bestehen aus mobilen autonomen Einheiten, die sich auf einer Wegeinfrastruktur bewegen, die ebenfalls einen Graph darstellt. Kennzeichnend ist dabei, dass Wege eine horizontale Ausdehnung bezogen auf die Längsachse besitzen und Einheiten autonom über ihre Bahn unter Zuhilfenahme von Umgebungsinformationen entscheiden. Neben freinavigierenden autonomen Fahrzeugen kommt dem Mensch die größte Bedeutung zu, unabhängig davon, ob er selbst transportiert oder Fahrzeuge bzw. Hilfsmittel steuert.

Aus steuerungstechnischer Sicht sind *aktive* und *passive* Transfersysteme unterscheidbar [34]. Aktive Systeme verfügen über Rechen- und Kommunikationstechnik und sind in die logistischen Entscheidungsprozesse involviert. Passive Systeme verfügen dagegen über keine Steuerungstechnik oder sind nicht an den Entscheidungsprozessen beteiligt. Die Zugehörigkeit spurgebundener Transfersysteme tendiert zu den passiven, freinavigierende zu den aktiven Systemen. Allerdings ist keine allgemeingültige Aussage möglich, da die Klassifizierung vom Einzelfall abhängt. Zur Vereinfachung wird dennoch im Folgenden passiv mit spurgebunden und aktiv mit freinavigierend gleichgesetzt.

7.2 Funktionsprinzip der Wegfindung

Die Wegfindung von Flussgegenständen ist ein rein graphentheoretisches Problem und kann daher losgelöst von Fördern und Transport sowie Spurbindung und Freinavigation betrachtet werden.

Es wird angenommen, dass für alle herstellbaren Produkt-Varianten-Kombinationen $(P_A, V_B)\ mit\ P_A \in \{P\}\ und\ V_B \in \{V_{P_A}\}$ eigene Arbeitspläne existieren bzw. variantenübergreifende Arbeitspläne in variantenspezifische Arbeitspläne übergeführt werden können. Des Weiteren wird vorausgesetzt, dass Vorerzeugnistypen aller Produkt-Varianten-Kombinationen insofern als eigenständige Produkte wahrgenommen werden, als dass für sie ebenfalls eigene Arbeitspläne existieren. Grundsätzlich stellt jeder Arbeitsplan einen unidirektionalen, zyklenfreien Graph dar, wobei jeder Knoten eine durchzuführende Tätigkeit T aus der abgeschlossenen Menge an Tätigkeiten $\{T\}$, die insgesamt für die Erzeugung aller Produkt-Varianten-Kombinationen benötigt wird, beschreibt. Über Vorgänger-Nachfolger-Beziehungen stehen diese in der zur Erzeugung richtigen Reihenfolge (Abb. 7.1 – Arbeitspläne).

Jeder durchführbaren, qualitativ unterscheidbaren Tätigkeit wird eine eindeutige Kennung (eben T) zugeordnet, unter der alle *gleichgearteten Tätigkeiten* zusammengefasst sind. Zwei Tätigkeiten werden als gleichartig angenommen, wenn sie aufeinander folgend an einer Arbeitsstation *keinerlei Umrüstaufwand* verursachen und die Ergebnisse die *gleichen Qualitätsmerkmale* aufweisen. Die Durchführbarkeit einer Tätigkeit an einer Arbeitsstation hängt von den Einflussgrößen *Werkzeug, technologi-*

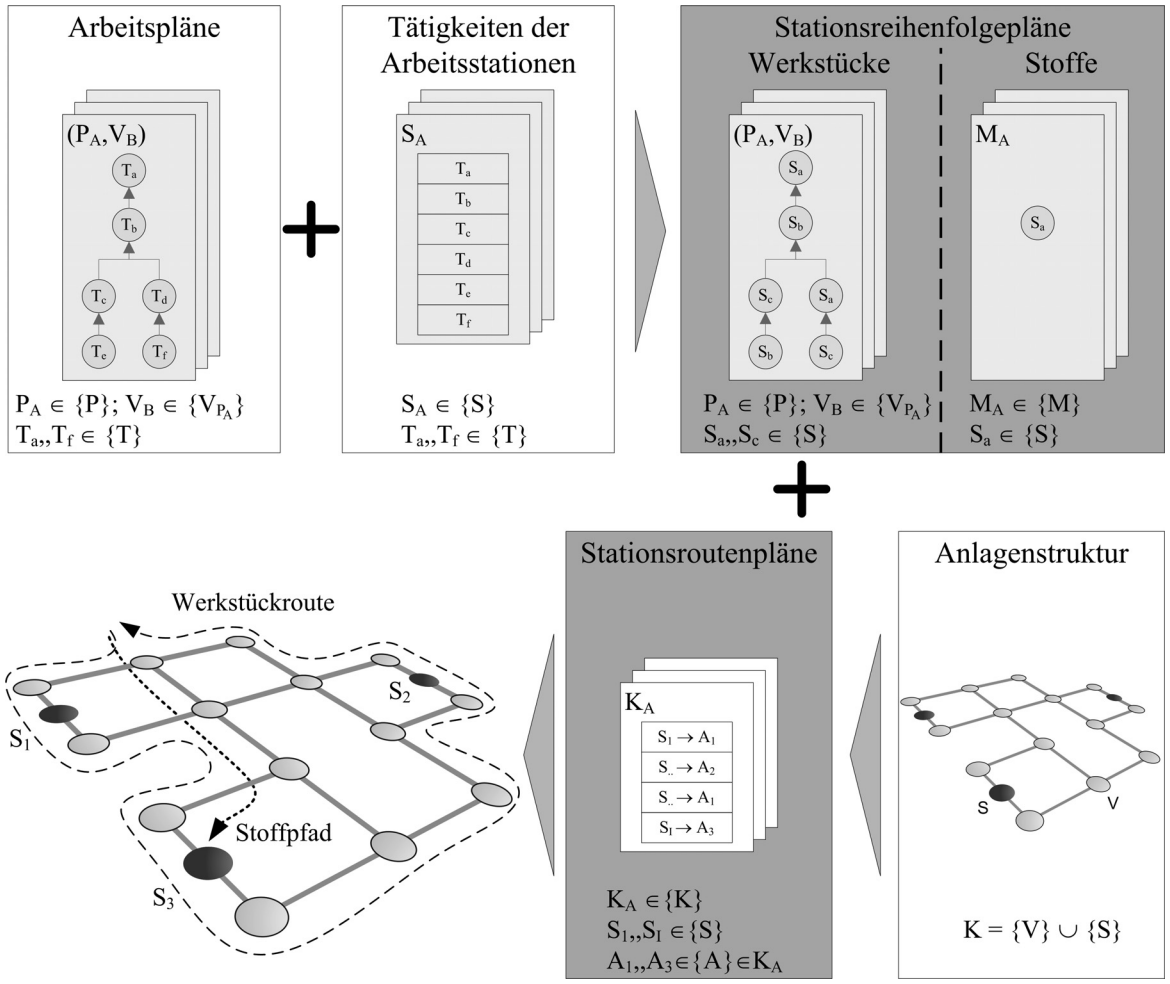


Abbildung 7.1: Stoffffluss mit Hilfe abgeleiteter Stationsreihenfolge- und Stationsroutenpläne. A = Ausgang; K = Knoten; M = Material / Stoffe; P = Produkt; S = Station; T = Tätigkeit; V = Verzweigung; V_{PA} = Variante.

sche Dimensionierung, Arbeitsraum sowie *der örtlichen und zeitlichen Auflösung der Prozessregelung* ab.

Als weitere Annahme existiert zu jeder Arbeitsstation $S_A \in \{S\}$ eine Liste der durchführbaren Tätigkeiten $\{T_{S_A}\} \subseteq \{T\}$ (Abb. 7.1 – Tätigkeiten der Arbeitsstationen). Über einen Abgleich der durchzuführenden mit den an den Stationen durchführbaren Tätigkeiten können die Arbeitspläne in Stationsreihenfolgepläne übergeführt werden (Abb. 7.1).

Für den Transfer der restlichen Stoffe $M_A \in \{M\}$ können während des Betriebs gleichgeartete Stationsreihenfolgepläne erzeugt werden, wobei diese lediglich einen einzigen Knoten enthalten. Allerdings setzt dies noch eine Zuordnung von Stoffen zu Tätigkeiten in den Arbeitsplänen voraus. Durch diese Vorgehensweise ist eine einheitliche Behandlung von Werkstück- und Stoffflüssen möglich, so dass neben *stoffreinen* Flusssystemen auch *Mischflusssysteme* durch Zusammenfassung verschiedener Flüsse realisierbar sind.

Um Werkstücke durch ein Transfernetz schleusen zu können, ist eine Kenntnis der Anlagenstruktur

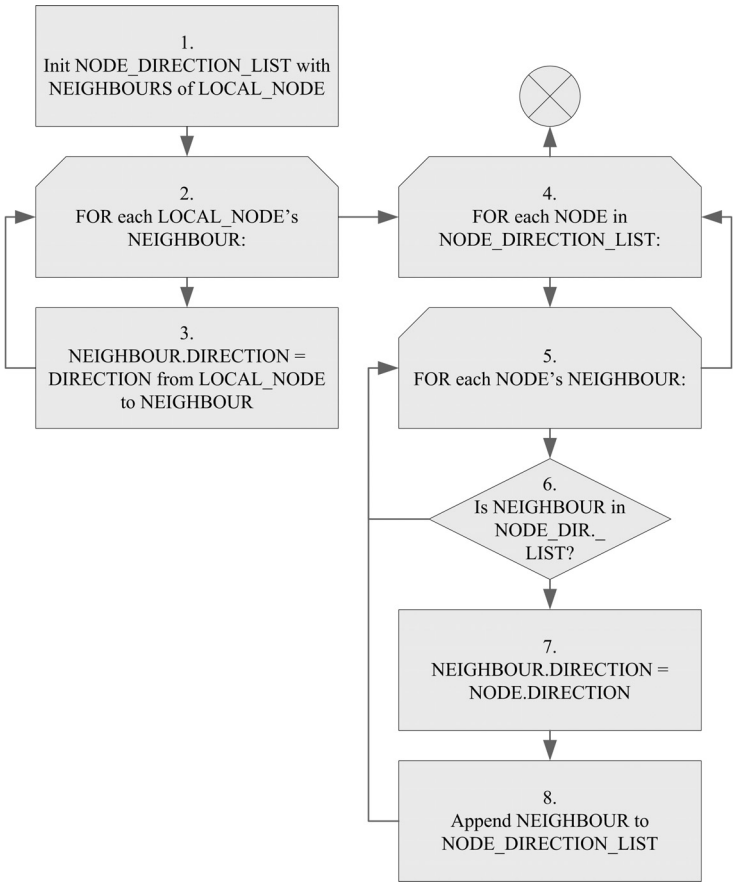


Abbildung 7.2: Algorithmus zur Erzeugung einer Routentabelle (*NODE_DIRECTION_LIST*) für einen Knoten auf Grundlage von Mikronetzen. Anmerkung: Durch Anhängen neuer Knoten wächst die Tabelle.

erforderlich. Liegt diese vor, kann für jeden Knoten $K_A \in \{K\}$ ein Stationsroutenplan erstellt werden. Darin ist für jede Arbeitsstation $S \in \{S\}$ der Ausgang mit dem kürzesten Weg verzeichnet. Als Knoten zählen hierbei Arbeitsstationen und Verzweigungen $\{V\}$, so dass $\{K\} = \{V\} \cup \{S\}$. Da jeder beliebige Graph in eine Liste von Mikronetzen zerlegbar ist, kann zur Routenplanerzeugung auf einem Knoten der in Abbildung 7.2 dargestellte Algorithmus angewendet werden. Indem jeder Knoten über einen eigenen Stationsroutenplan verfügt, können Flussgegenstände geroutet, d.h. von Knoten zu Knoten zum Zielort befördert werden.

7.3 Funktionsprinzip von Verzweigungen

Der Transfer durch das Transfernetz erfolgt über das Durchleiten der Flussgegenstände über mehrere Verzweigungen. Eine Verzweigung besteht aus einer Menge gepufferter Eingänge $\{E\}\ mit\ |\{E\}| \geq 1$, einer Einrichtung zur Richtungsänderung V und einer Menge an Ausgängen $\{A\}\ mit\ |\{A\}| \geq 1$ (Abb. 7.3). Je nach Aufbau verfügt die Verzweigung über eine maximale Kapazität $K \geq 1$ aufnehmbarer Flussgegenstände.

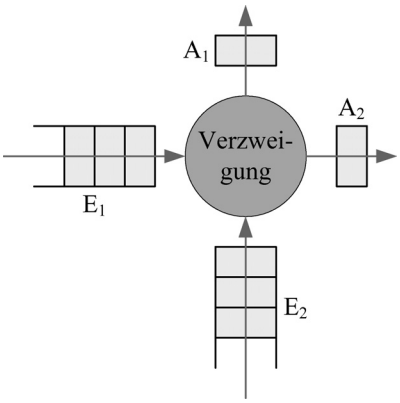


Abbildung 7.3: Schematischer Aufbau einer Verzweigung. Im Beispiel mit zwei Ein- und Ausgängen. A_n = Ausgang; E_m = Eingang.

Zwar unterscheiden sich aus technischer Sicht Verzweigungen realer Transfersysteme deutlich, allerdings lassen sie sich in die beiden Grundformen *Weiche* und *Kreuzung* einteilen. Wesentliches Merkmal einer Weiche ist, dass Ein- *und* Ausgang miteinander korrekt verschaltet werden *bevor* ein Flussgegenstand sie betritt. Zudem muss dieser auf der Weiche zur Richtungsänderung nicht gestoppt werden. Im Gegensatz dazu betritt ein Gegenstand eine Kreuzung *nachdem* ein Eingang, jedoch *bevor* ein Ausgang korrekt geschalten wurde. Zumeist wird zur Richtungsänderung gestoppt. Während bei spurgebundenen Systemen beide Formen zu finden sind, kommen bei freinavigierenden Systemen zumeist nur Weichen vor (Abb. 7.4).

Für Kreuzungen gilt als weiteres Unterscheidungskriterium die Position, an der eine Identifizierung des Flussgegenstands erfolgt. Sie ist entweder an allen Eingängen oder zentral auf der Kreuzung möglich. Bei Weichen erfolgt dieser Vorgang notwendiger Weise immer an den Eingängen.

Aus Sicht der Ablaufsteuerung ergeben sich dadurch die drei unterschiedlichen Abläufe *Identifizierung vor Weiche*, *Identifizierung vor Kreuzung* und *Identifizierung auf Kreuzung*, die einander ähnlich sind. Sie entstehen durch eine Verkettung der Funktionen *Zugangsmeldung* zur Signalisierung ankommender Flussgegenstände, *Vereinzelung* zur Trennung am Eingang, *Eingangsauswahl* und *Einlassen* eines Objekts, *Stoppen* auf einer Kreuzung, *Identifizierung* eines Objekts, *Ausgangsauswahl* und *Auslassen* sowie *Abgangsmeldung* zur Signalisierung einer freien Verzweigung (Abb. 7.4).

Der Steuerungsablauf bei der *Identifizierung vor Weiche* vollzieht sich wie folgt: Jeder ankommende Flussgegenstand wird am Eingang zunächst als ankommend erkannt, identifiziert und am Einlassen auf die Verzweigung gehindert. Die Anwesenheit und Identität wird einer *Eingangskontrolle* gemeldet, die unter den, an verschiedenen Eingängen wartenden, Gegenständen auswählt. Die Auswahl eines Eingangs signalisiert sie an den *Router*, der anhand der Identität des dort wartenden Flussgegenstands den besten Ausgang ermittelt. Sind Eingang und Ausgang bestimmt, wird die *Ausführungssteuerung* der Verzweigung mit dem Einlass, der physischen Durchführung der Richtungsänderung und dem anschließenden Auslass beauftragt. Das erfolgreiche Verlassen der Verzweigung wird über eine Ausgangsmeldung an die Einlasskontrolle quittiert, so dass der nächste Flussgegenstand bearbeitet werden kann. Abbildung 7.5 zeigt das Beispiel einer Ablaufsequenz.

Die beiden anderen Steuerungsabläufe unterscheiden sich im Wesentlichen dadurch, dass der Flussgegenstand auf der Kreuzung gestoppt wird, bevor eine Richtungsänderung und der Auslass erfolgt.

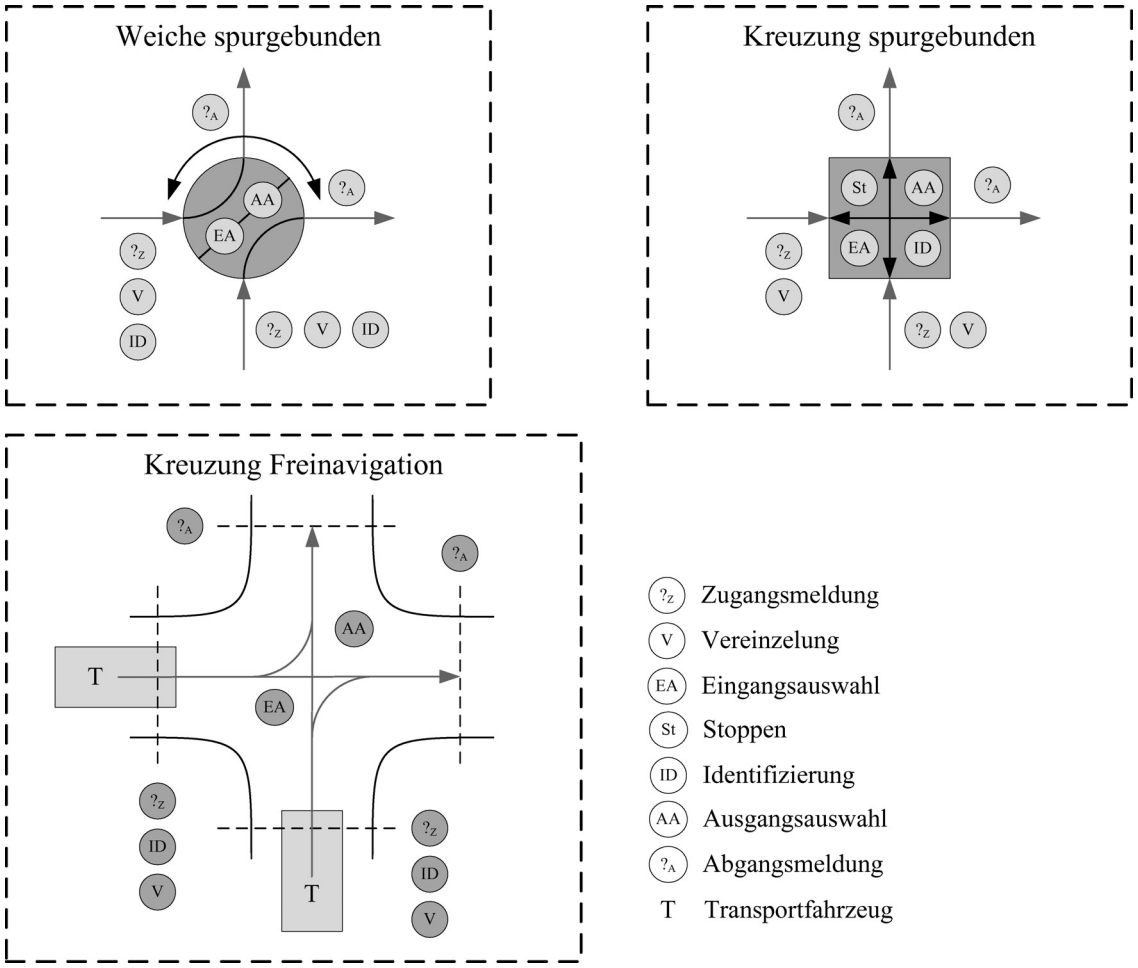


Abbildung 7.4: Oben links – Schematischer Aufbau einer spurgebundenen Weiche. Oben rechts – Schematischer Aufbau einer spurgebundenen Kreuzung. Unten – Schematischer Aufbau einer Weiche für Freinavigation. Die Funktionen des Einlassens und Auslassens sind zur Vereinfachung der Eingangs- bzw. der Ausgangsauswahl zugeordnet.

7.4 Generisches Steuerungsmodell für Verzweigungen

Das Steuerungsmodell basiert auf der auf Wandelbarkeit ausgerichteten Softwarearchitektur (Kap. 5) und nutzt teilweise die in Kapitel 5.6 vorgestellten Grundfacetten. Es vereint die drei Steuerungsabläufe *Identifizierung vor der Weiche*, *Identifizierung vor der Kreuzung* und *Identifizierung auf der Kreuzung*, so dass eine alternative Applikation auf verschiedene reale Gerätetypen möglich ist. Die Ausdifferenzierung des Verhaltens erfolgt ausschließlich durch die Ankopplung entsprechender Sensoren und Aktoren und der Ausführungssteuerung. Dies ist möglich, da die einzelnen Teilabläufe als Reflexe konzipiert sind, die nur durch angeschlossene Sensorik aktiviert werden können (Kap. 5.4).

Das Verzweigungsmodell ist aus verschiedenen Bausteinen aufgebaut. Jeder Eingang E stellt einen eigenen Knoten (Kap. 5.3.1) dar, mit jeweils einer Digital-In- $?_Z$, Digital-Out- V und String-In-Facette ID_1. Für jeden Ausgang ist eine Digital-In-Facette $?_A$ und für die eigentliche Verzweigung VZW jeweils eine Digital-Out- St, String-In- ID_1, Digital-In- $?_R$, Einlasskontroll- EK, Router- R und

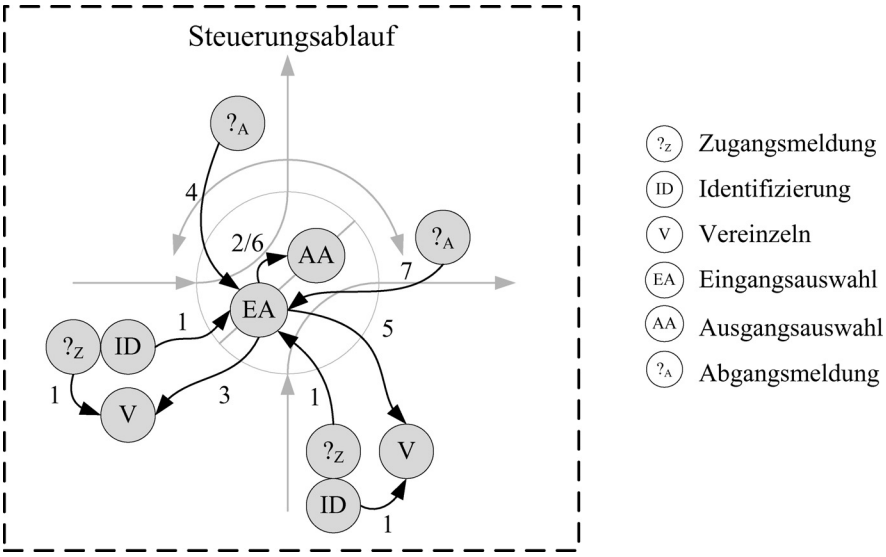


Abbildung 7.5: Nummerierte Sequenz des Steuerungsablaufs einer Verzweigung am Beispiel einer spurgebundenen Weiche mit Identifizierung an jedem Eingang. An beiden Eingängen treffen gleichzeitg Flussgegenstände ein.

Ausführungssteuerungsfacette $Strg$ vorhanden. Von der Einlasskontrolle, dem Router und der Ausführungssteuerung abgesehen, handelt es sich bei den restlichen Facetten um Grundfacetten (Kap. 5.6).

Abhängig von der Sensorikanschaltung werden Teile der Facetten genutzt. Die drei Variationsmöglichkeiten der Steuerungsabläufe sind in Abbildungen 7.9, 7.10, 7.11 – zur besseren Übersicht getrennt – dargestellt. Die schwarz eingefärbten Facetten sind jeweils ungenutzt.

7.4.1 Aufbau der spezifischen Facetten

Einlasskontrolle (Router Entrance Control): Die Struktur der Facette ist in Abbildung 7.6 gezeigt. Zur Erfüllung ihrer Funktion interagiert die Facette aktiv und passiv mit anderen. Dazu hält sie auf die Vereinzeler der Eingänge Referenzen in einer Liste (*vs*) sowie jeweils eine Referenz auf den Stopper (*st*) der Verzweigung und die Ausführungssteuerung (*crtl*). Um Interessenten über die Auswahl eines Flussgegenstands zum Einlass zu benachrichtigen, verfügt die Facette über den Publisher *RouteObj(id)*. Damit sie als Kreuzung mit Identifikation *am* Eingang oder als Weiche einsatzfähig ist, besitzt sie den Callback *EntranceNewId(id)*. Für den umgekehrten Fall der Identifizierung *auf* der Kreuzung steht der Callback *EntranceRisingEdge* zur Verfügung, über den eine bloße Anwesenheit mitgeteilt wird. Das vollständige Verlassen eines Flussgegenstands von der Verzweigung wird über den dritten Callback *ExitRisingEdge* mitgeteilt, so dass die Einlasskontrolle einen neuen Gegenstand auswählen kann.

Router (Router): Abbildung 7.7 stellt die Struktur der Facette dar. Der Router hält eine Referenz (*crtl*) auf die Ausführungssteuerung, um ihr den gewählten Ausgang mitzuteilen. Die Ermittlung des Ausgangs wird über den Callback *RouteObj(id)* veranlasst.

```
FACET_ROUTER_ENTRANCE_CONTROL = [
  Facets = [
      FACET_DIGITAL_OUT st
      FACET_DIGITAL_OUTS vs
      FACET_ROUTER_PHYSICAL_CONTROL ctrl
  ]
  Publishers = [
      RouteObj(STRING id)
  ]
  SubscriptionCallbacks = [
      EntranceRisingEdge()          //Identität des Eingangs ist über den mitgesendeten
      EntranceNewId(STRING id)      //Locator des Publishers ermittelbar
      ExitRisingEdge()              //Gilt analog für Ausgang
  ]
]
```

Abbildung 7.6: Facette zur Eingangsauswahl.

```
FACET_ROUTER = [
  Facets = [
      FACET_ROUTER_PHYSICAL_CONTROL ctrl
  ]
  SubscriptionCallbacks = [
      RouteObj(STRING id)
  ]
]
```

Abbildung 7.7: Facette zur Ausgangsauswahl.

Ausführungssteuerung (Router Physical Control): Abbildung 7.8 zeigt die Struktur der Facette, die die Verbindung zur tatsächlichen Gerätesteuerung darstellt. Die Aufgabe der Ausführungssteuerung ist es, physische Zustandswechsel durchzuführen, um Flussgegenständen ein Einlassen in die Verzweigung oder ein Auslassen aus dieser zu ermöglichen. Handelt es sich um eine Kreuzung, sind Einlassen und Auslassen zwei unterschiedliche Zustände. Bei einer Weiche handelt es sich beim Einlassen und Auslassen um einen einzigen Zustand.

Diese Aufgabe wird durch die zwei Prozeduren *FromEntrance(entrance)* und *ToExit(exit)* sowie den Callback *LetOutWhenPossible* erfüllt. Die semantische Bedeutung der ersten Prozedur unterscheidet sich abhängig davon, ob es sich um eine Kreuzung oder Weiche handelt. Bei einer Kreuzung wird unmittelbar die physische Bereitschaft des, als Parameter übergebenen, Eingangs zum Einlassen veranlasst. Die Prozedur kehrt erst zurück, wenn dieser Zustand erreicht ist. Bei einer Weiche wird lediglich der Eingang übergeben und dadurch signalisiert, dass die physische Bereitschaft veranlasst werden kann, sobald auch der Ausgang der Ausführungssteuerung bekannt ist. Dieser semantische Unterschied ist von der tatsächlichen Gerätesteuerung abzufangen. Bei einer Kreuzung wird die physische Bereitschaft zum Auslassen hergestellt, indem dies über den Callback indirekt veranlasst wird (je nachdem, ob die Identifizierung *vor* oder *auf* der Kreuzung erfolgt). Zunächst wird intern nur die Erlaubnis zur Zustandsveränderung vermerkt. Durchgeführt wird sie erst dann, wenn auch der Ausgang bekannt ist.

```
FACET_ROUTER_PHYSICAL_CONTROL = [
   Publishers = [
   ]
   SubscriptionCallbacks = [
        LetOutWhenPossible()
   ]
   FromEntrance(LOCATOR entrance)
   ToExit(LOCATOR exit)
]
```

Abbildung 7.8: Facette als Schnittstelle zur physischen Ausführungssteuerung.

7.4.2 Informationelle Verknüpfung und Steuerungsablauf als Kreuzung

Wie bereits eingangs erwähnt, implementiert das Steuerungsmodell drei verschiedene Abläufe, wobei die Auswahl implizit durch unterschiedliche Ankopplung von Sensorik geschieht. Die dahinter liegenden Reflexketten sind fest „verdrahtet“, indem die Facetten untereinander über Prozeduren, Publisher und Callbacks informationell verknüpft sind. Beim Entwurf wurde darauf geachtet, dass sich die Reflexe nicht gegenseitig stören und kein „Übersprechen“ stattfinden kann.

Zur Vereinfachung erfolgt die Darstellung der Reflexe und Verknüpfungen für Kreuzungen und Weichen getrennt, wobei Teile, die für beide gelten, bereits an dieser Stelle beschrieben werden.

Die Zugangsmeldung: ist abhängig vom Ort der Identifizierung. Erfolgt diese auf der Verzweigung, findet die Zugangserkennung mittels $?_Z$ über den Publisher *RisingEdge* statt, andernfalls mittels ID_1 über den Publisher *NewString*. An beiden ist EK mit den Callbacks *EntranceRisingEdge* für Meldungen von $?_Z$ und *EntranceNewId(id)* für Meldungen von ID_1 registriert, so dass sie über Neuzugänge informiert wird (Abb. 7.9, 7.10, 7.11, 1a bis 2a).

Die Vereinzelung: erfolgt über V, der hierzu am Publisher *RisingEdge* von $?_Z$ und *NewString* von ID_1 registriert ist. Sobald eine Zugangsmeldung eintrifft, blockiert er den Eingang auf die Verzweigung (Abb. 7.9, 7.10, 7.11, 1b bis 3b).

Die Eingangsauswahl: geschieht durch EK, wenn mindestens ein Zugang an den Eingängen verzeichnet wurde und die Verzweigung frei ist (Abb. 7.9, 7.10, 7.11, 1c).

Das Einlassen: findet nach der Auswahl des Eingangs statt. Über den Publisher *RouteObj(id)* signalisiert EK an R – über dessen, daran registrierten, gleichnamigen Callback *RouteObj(id)* – die Identität des Flussgegenstands. Anschließend informiert EK die $Strg$ mittels ihrer Prozedur *FromEntrance(entrance)* über den ausgewählten Eingang. Die Kreuzung wird dadurch unmittelbar in einen betretbaren Zustand versetzt – die Prozedur kehrt erst nach Erreichen des Zustands zurück. Nach Rückkehr wird V von EK über die Prozedur *SetState(False)* dazu veranlasst, den Eingang zu entblockieren, so dass der Flussgegenstand in die Verzweigung eingelassen wird (Abb. 7.9, 7.10, 6c bis 8c).

Die Identifizierung: erfolgt sobald ein Flussgegenstand in Reichweite des Sensors gelangt. Befindet sich der Sensor auf der Kreuzung, wird die Identität R über den Publisher *StringChanged(string)* signalisiert, der dazu mit seinem Callback *RouteObj(id)* an ID_2 registriert ist (Abb. 7.9, 1d bis 2d). Befindet sich dagegen der Sensor vor der Kreuzung, erfolgt die Identifizierung an ID_1. Dieser publiziert ebenfalls über *StringChanged(string)*, wodurch die Informationen von EK über den Callback

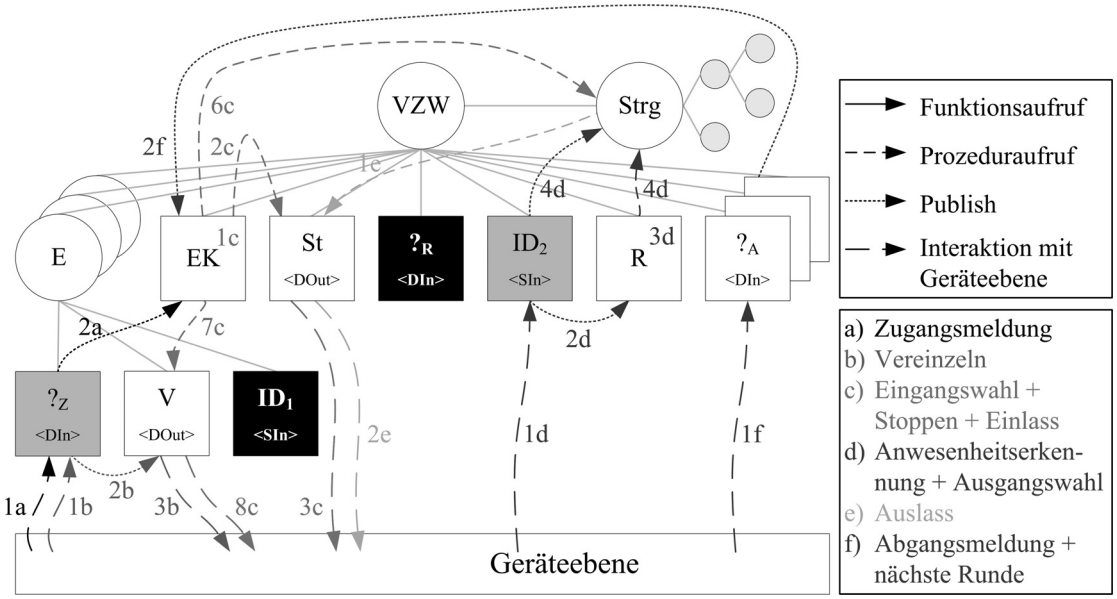


Abbildung 7.9: Generische Ablaufsteuerung einer spurgebundenen Kreuzung mit Identifikation von Flussgegenständen *auf* der Kreuzung. Aus Gründen der Übersichtlichkeit ist die Facette der Ausführungssteuerung nicht explizit dargestellt und der gerätespezifische Steuerungsbaum nur angedeutet. $?_A$ = Ausgangsmelder; $?_R$ = Anwesenheitsmelder; $?_Z$ = Zugangsmelder; E = Eingang; EK = Eingangskontrolle; ID_n = Identifizierer; R = Router; St = Stopper; $Strg$ = Ausführungssteuerung; V = Vereinzeler; VZW = Verzweigung.

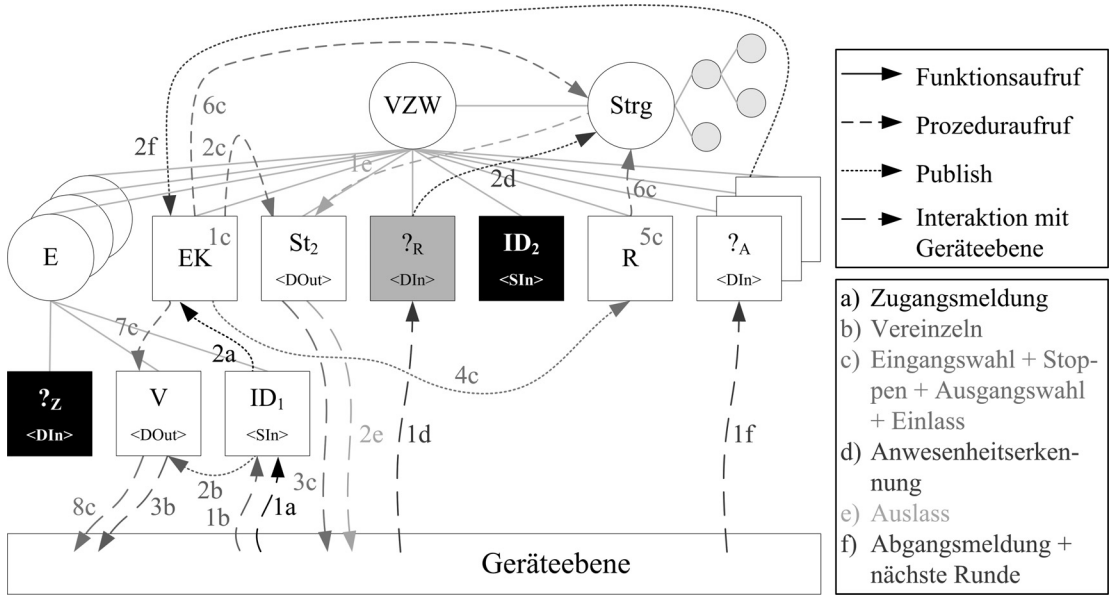


Abbildung 7.10: Generische Ablaufsteuerung einer spurgebundenen Kreuzung mit Identifikation von Flussgegenständen *vor* der Kreuzung. Aus Gründen der Übersichtlichkeit ist die Facette der Ausführungssteuerung nicht explizit dargestellt und der gerätespezifische Steuerungsbaum nur angedeutet. $?_A$ = Ausgangsmelder; $?_R$ = Anwesenheitsmelder; $?_Z$ = Zugangsmelder; E = Eingang; EK = Eingangskontrolle; ID_n = Identifizierer; R = Router; St = Stopper; $Strg$ = Ausführungssteuerung; V = Vereinzeler; VZW = Verzweigung.

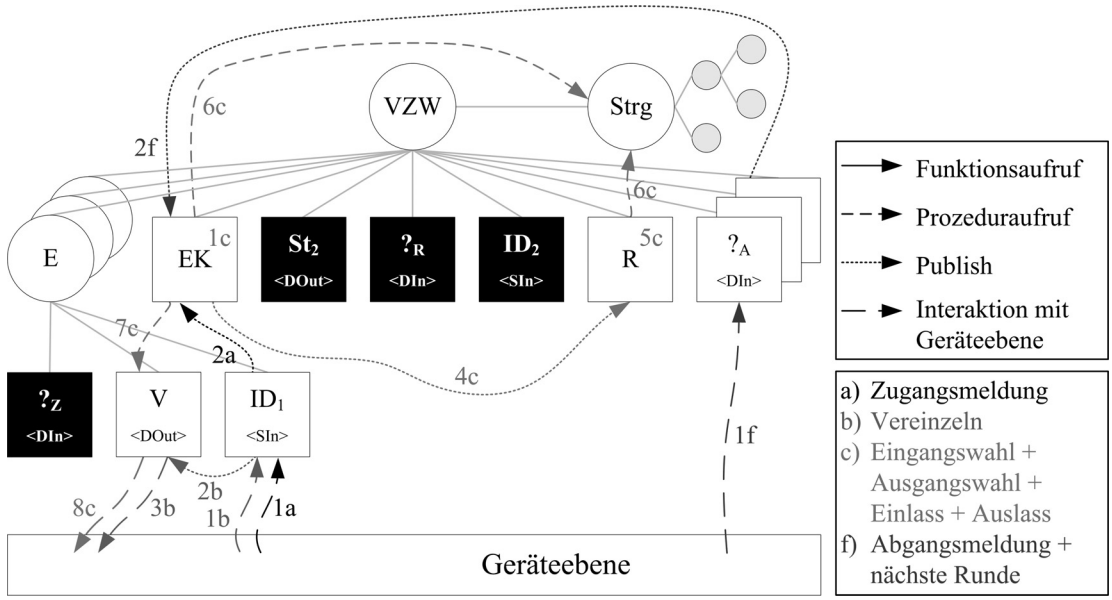


Abbildung 7.11: Generische Ablaufsteuerung einer spurgebundenen Weiche. Aus Gründen der Übersichtlichkeit ist die Facette der Ausführungssteuerung nicht explizit dargestellt und der gerätespezifische Steuerungsbaum nur angedeutet. $?_A$ = Ausgangsmelder; $?_R$ = Anwesenheitsmelder; $?_Z$ = Zugangsmelder; E = Eingang; EK = Eingangskontrolle; ID_n = Identifizierer; R = Router; St = Stopper; $Strg$ = Ausführungssteuerung; V = Vereinzeler; VZW = Verzweigung.

EntranceNewId(id) empfangen werden. EK publiziert die Identität über den Publisher *RouteObj(id)* weiter und wird schließlich von R über den gleichnamigen Callback empfangen (Abb. 7.10, 7.11, 1a bis 2a, 4c).

Das Stoppen: des Gegenstands auf einer Kreuzung wird durch EK vorbereitet, indem sie nach Auswahl eines Eingangs die Prozedur *SetState(True)* an St aufruft (Abb. 7.9, 7.10, 2c bis 3c).

Eine Anwesenheitserkennung: ist auf einer Kreuzung erforderlich, um $Strg$ zu signalisieren, dass sich der Flussgegenstand vollständig auf der Kreuzung befindet. Hierzu ist $Strg$ über den Callback *LetOutWhenPossible* sowohl am Publisher *RisingEdge* von $?_R$ als auch am Publisher *NewString* von ID_2 registriert. Sobald der Ausgang bekannt ist, kann mit der physischen Zustandsänderung zum Auslassen begonnen werden (Abb. 7.9, 7.10, 1d bis 2d).

Die Ausgangsauswahl: geschieht durch R, sobald die Identität des Flussgegenstands seinem Callback *RouteObj(id)* – entweder mittels des gleichnamigen Publishers *RouteObj(id)* über EK oder des Publishers *StringChanged(string)* über ID_2 – signalisiert wurde (Abb. 7.9, 3d, Abb. 7.10, 7.11, 5c).

Das Auslassen: findet, nachdem die Ausführungssteuerung den Ausgang eingestellt hat, statt. Hierzu ruft $Strg$ an St die Prozedur *SetState(False)* auf, worauf dieser entblockiert und der Flussgegenstand die Kreuzung verlassen kann (Abb. 7.9, 7.10, 1e bis 2e).

Die Abgangsmeldung: erfolgt über den Publisher *RisingEdge* von $?_A$. Mit ihr wird ein freier Zustand der Verzweigung signalisiert, weshalb EK an diesem Publisher mit dem Callback *ExitRisingEdge* registriert ist. Bei Zustellung eines Ereignisses erfolgt bei wartenden Flussgegenständen die Auswahl des nächsten Eingangs (Abb. 7.9, 7.10, 7.11, 1f bis 2f).

7.4.3 Informationelle Verknüpfung und Steuerungsablauf als Weiche

Zugangsmeldung, Vereinzelung, Identifizierung, Stoppen, Anwesenheitserkennung, Ausgangsauswahl sowie *Abgangsmeldung* bei einer Weiche sind identisch zu den oben beschriebenen Reflexen. Der wesentliche Unterschied zur Kreuzung besteht darin, dass die Einlass- und Auslassbereitschaft immer miteinander einhergehen, also einen gemeinsamen Zustand darstellen.

Nach der Auswahl des Eingangs signalisiert EK über den Publisher *RouteObj(id)* an R – über dessen, daran registrierten, gleichnamigen Callback *RouteObj(id)* – die Identität des Flussgegenstands. Anschließend informiert EK die $Strg$ mittels ihrer Prozedur *FromEntrance(entrance)* über den ausgewählten Eingang. Der Prozeduraufruf signalisiert $Strg$, dass, unmittelbar nach Vorliegen des Ausgangs – über die von R aufgerufene Prozedur *ToExit(exit)* –, der physische Zustandswechsel durchgeführt werden darf. Allerdings kehrt *FromEntrance(entrance)* nicht unmittelbar zurück. Stattdessen blockiert $Strg$ die Rückkehr solange, bis der Ausgang bekannt und der physische Zustandswechsel erfolgt ist. Im Anschluss daran wird dem Flussgegenstand der Einlass und Auslass ermöglicht, indem V über den Aufruf der Prozedur *SetState(False)* entblockiert wird.

7.4.4 Applikation auf freinavigierende Transportsysteme

Im Folgenden wird kurz eine beispielhafte Anwendung auf freinavigierende Transportsysteme skizziert. Es wird vorausgesetzt, dass an zentraler Stelle ein Softwaresystem Steuerungsmodelle für die Verzweigungen der Wegeinfrastruktur implementiert. Diese Verzweigungssteuerungen sind derart reduziert, dass V, ID_1, $?_A$ und $Strg$ nicht in dem Softwaresystem selbst implementiert sind, sondern dazu entsprechende Facetten auf den Transporteinheiten genutzt werden (Abb. 7.12). Die Ortstransparenz der auf Wandelbarkeit ausgerichteten Softwarearchitektur erlaubt dies.

Eine sich einer Verzweigung nähernde Transporteinheit registriert sich an dem Softwaresystem für diese Verzweigung unter Angabe der Locator ihrer Sensorfacetten. Das Softwaresystem beschafft daraufhin die Facettenreferenzen von der Transporteinheit und verknüpft alle Bausteine informationell, so dass die erforderlichen Reflexketten entstehen.

Hierbei kann für ID_1 und $?_A$ ein System zur Positionserkennung zugrunde liegen, das geometrisch *vor* und *nach* einer Weiche entsprechende Ereignisse erzeugt, wobei im Falle von ID_1 die Identität des Flussgegenstands mitgegeben wird. Hinter V steht ein Start- und Stoppbefehl an die Ausführungssteuerung der Transporteinheit.

7.4.5 Ansatzpunkte zur effizienten Steuerung der Stoffflüsse

Mit den bisherigen Konzeptionen ist aus Sicht der Ablaufsteuerung die Infrastruktur geschaffen, so dass *effektive Stoffflüsse* realisiert werden können. Da jedoch für eine leistungsstarke Produktion *Effizienz* benötigt wird, müssen geeignete Flussstrategien realisiert werden können. Ansatzpunkte für die Beeinflussung von Stoffflüssen finden sich an zwei Stellen – bei der Auswahl des Eingangs und des Ausgangs. Da für eine Auswahl unterschiedliche Strategien – unter Einbeziehung verschiedener Informationen – denkbar sind, verfügen Eingangskontrolle und Router über eine Funktionsschnittstelle,

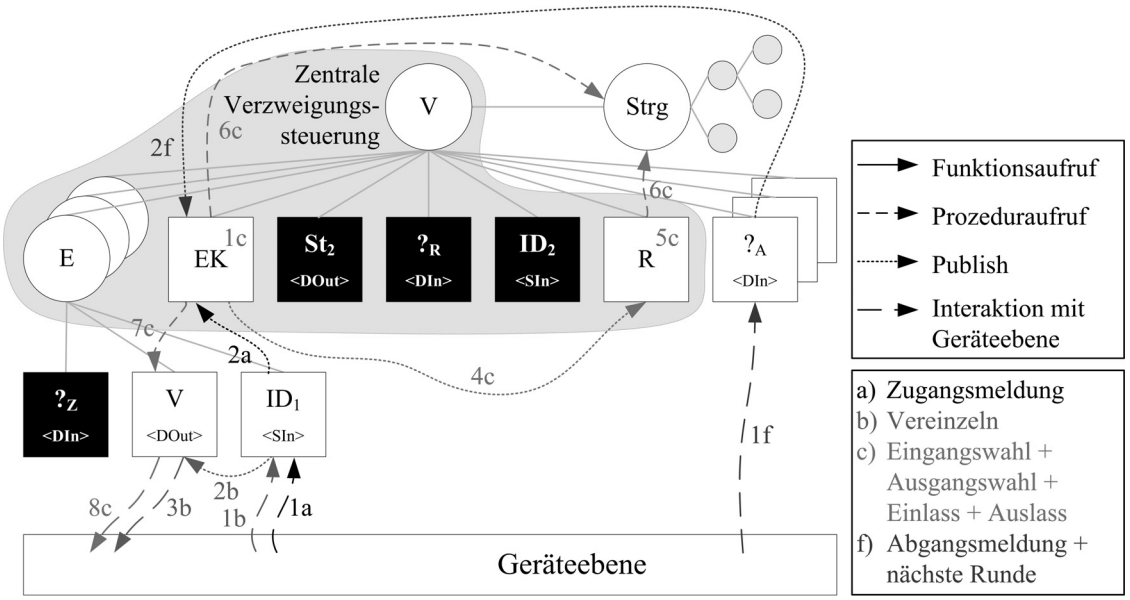


Abbildung 7.12: Generische Ablaufsteuerung einer Transportkreuzung. Aus Gründen der Übersichtlichkeit ist die Facette der Ausführungssteuerung nicht explizit dargestellt und der gerätespezifische Steuerungsbaum nur angedeutet. $?_A$ = Ausgangsmelder; $?_R$ = Anwesenheitsmelder; $?_Z$ = Zugangsmelder; E = Eingang; EK = Eingangskontrolle; ID_n = Identifizierer; R = Router; St = Stopper; $Strg$ = Ausführungssteuerung; V = Vereinzeler; VZW = Verzweigung.

an die verschiedene Auswahlfunktionen gekoppelt werden können (Abb. 7.13). Dadurch ist ein flexibler Austausch und eine fallabhängige Integration von Strategien möglich, ohne entwicklungsseitig Veränderungen an den beiden Facetten vornehmen zu müssen. Als Eingangsgrößen wird jeweils die Referenz auf die Eingangskontroll- EK bzw. die Routerfacette R übergeben. Einlassfunktionen erhalten so Zugriff auf die Liste der wartenden Flussgegenstände, Auslassfunktionen können gezielt Informationen über den Anlagenzustand beziehen. Grundsätzlich steht den Funktionen der vollständige Zugriff auf das gesamte Steuerungsmodell der Anlage – und damit auf das gesamte „Anlagenwissen“ – frei. Als Rückgabewerte liefert eine Einlassfunktion einen Eingang und eine Auslassfunktion eine Liste priorisierter, möglicher Ausgänge zurück.

7.5 Dezentrale Strategien zur Selbstorganisation von Stoffflüssen

Flussstrategien wirken sich insofern auf die Leistungsfähigkeit einer Produktionsanlage aus, als dass sie für einen reibungsfreien Transport der Stoffe in den Transfersystemen verantwortlich sind. Hierbei gilt es, mehrere ungünstige Situationen zu vermeiden. Staus in Transportsystemen, durch Überlastung einzelner Transferstrecken oder ausgefallene Verzweigungen verursacht, vermindern den Stoffdurchsatz. Bei redundanter Verfügbarkeit von Tätigkeiten und geeigneter Struktur der Transfernetze dürfen Arbeitsstationen nicht einseitig bevorzugt werden, da dadurch einerseits der Gesamtdurchsatz der Produktionsanlage sinkt und anderseits Eingangspuffer überlaufen können. Letzteres erhöht die Gefahr von Rückstauungen von Flussgegenständen in die Transfersysteme. Ausfälle von Arbeitsstationen kompensiert das Transfersystem bei geeigneter Struktur der Transfernetze idealerweise selbstständig.

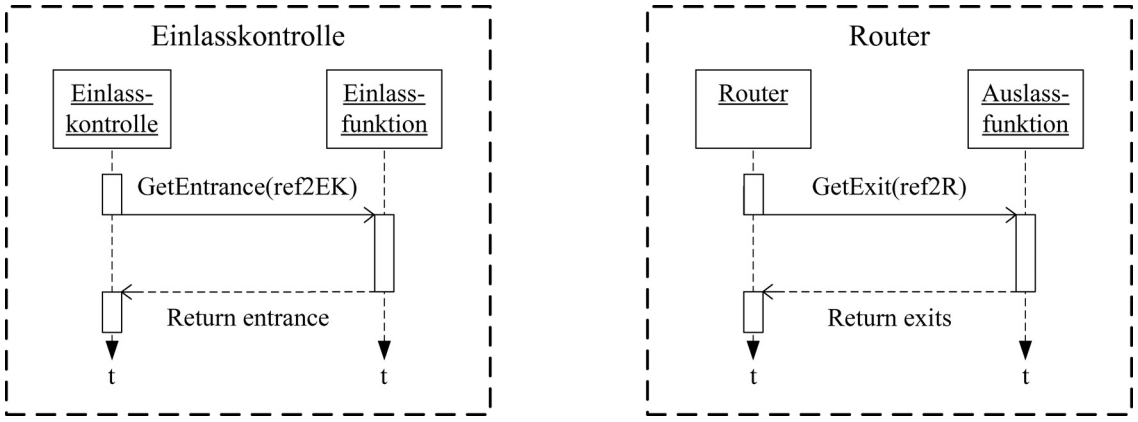


Abbildung 7.13: Funktionsschnittstellen für Einlass- und Auslassfunktionen. Verschiedene Strategien können integriert werden, ohne entwicklungsseitig die Einlasskontrolle oder den Router verändern zu müssen.

Zu den Ausfällen zählen hierbei Störungen, Pausen, Wartungen, Instandsetzungsarbeiten, etc.

Flussstrategien unterscheiden sich in ihrer Fähigkeit zur Vermeidung dieser Situationen und durch die Art und Menge der dazu verwerteten Informationen, ihrer Gewichtung sowie dem Informationsradius. Der Radius entspricht der Dimension des Umfelds einer Verzweigung, aus dem die Informationen für eine Auswahl von Ein- und Ausgängen bezogen werden. Dabei sind diese Informationen nicht auf die Transportsysteme beschränkt, sondern können auch Arbeitsstationen, Produktionsziele und Planungsstände betreffen. Alle Informationen sind grundsätzlich aus dem gesamten Steuerungsmodell der Produktionsanlage beziehbar.

Das an dieser Stelle verfolgte Ziel der vorliegenden Arbeit ist es, den einfachen Nachweis für die Machbarkeit selbstorganisierender Flüsse in wandelbaren Produktionsanlagen zu führen. Hierfür wurden die vier einfachen, dezentralen Auslassstrategien *stationsorientiertes Routing, tätigkeitsorientiertes Routing, tätigkeitsorientiertes Routing mit Lastausgleich* und *werkstückzentriertes, angebotsorientiertes Routing* entworfen. Während das erste Verfahren für den Transport aller Stoffe geeignet ist, dienen die anderen ausschließlich dem Transport von Werkstücken.

Von der Konzeption von Einlassstrategien wurde abgesehen, da es sich um das bekannte Problem der Reihenfolgebildung handelt. Zu diesem sind in der Literatur verschiedene Verfahren bekannt, wie z.B. *First-In-First-Out (FIFO), kürzeste Operationszeit (KOZ), längste Operationszeit (LOZ), frühester Plan-Endtermin (FPE), geringster Restschlupf (GR)* oder *Berücksichtigung von Prioritäten* [2, 44, 56, 88].

7.5.1 Stationsorientiertes Routing

Das einfachste aller denkbaren Verfahren ist das *stationsorientierte Routing*, das weitgehend dem bereits erörterten Funktionsprinzip der Wegfindung (Kap. 7.2) entspricht. Es wird dahingehend erweitert, dass mögliche Ausgänge bezüglich der Entfernung zur Zielstation priorisiert werden. Der Flussgegenstand verlässt die Verzweigung über den Ausgang mit der höchsten Priorität. Befindet sich

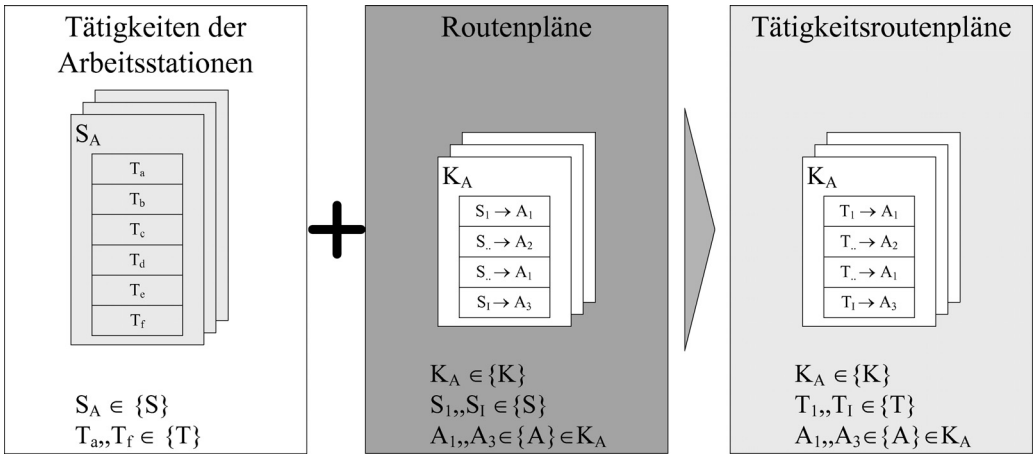


Abbildung 7.14: Tätigkeitsbasiertes Routing – Erzeugung von Tätigkeitsroutenplänen. A = Ausgang; K = Knoten; S = Station; T = Tätigkeit.

auf dem ausgehenden Transfersegment ein Stau, wird stattdessen der Ausgang mit der nächst niedrigeren Priorität gewählt, usw. Sind alle Ausgänge blockiert wartet der Flussgegenstand solange, bis sich der Stau bei irgendeinem Ausgang aufgelöst hat.

Zu jedem Flussobjekt existiert ein Stationsreihenfolgeplan (Kap. 7.2), der sukzessive abgearbeitet wird. Lediglich die spezifischen Wege zu den einzelnen Arbeitsstationen können im Verlauf varriieren. Im Falle von Stationsausfällen können Blockaden und Rückstauungen bis hin zum Anlagenstillstand entstehen, da keine Ersatzstationen vorgesehen sind.

7.5.2 Tätigkeitsorientiertes Routing

Beim *tätigkeitsorientierten Routing* sind auf den Verzweigungen Tätigkeitsroutenpläne hinterlegt. Sie entstehen durch Vereinigung von Stationsroutenplänen und den Tätigkeitslisten der Arbeitsstationen (Abb.7.14). Zu jeder in der Produktionsanlage verfügbaren Tätigkeit sind die jeweils geeigneten Stationen verzeichnet. Jeder Ausgang ist nach Entfernung zur nächsten geeigneten Station priorisiert.

Die Strategie reagiert durch die Priorisierung auf Staus an Ausgängen. Zudem entstehen keine Blockaden, solange sich funktionsbereite Arbeitsstationen im System befinden und erreichbar sind. Rückstauungen durch übergelaufene Eingangspuffer von Arbeitsstationen werden vermieden, da volle Eingangspuffer als Stau gewertet werden.

7.5.3 Tätigkeitsorientiertes Routing mit Lastausgleich

Dieses Verfahren ist eine Erweiterung des tätigkeitsorientierten Routings und beruht auf der Annahme, dass durch die starre Priorisierung der Ausgänge Arbeitsstationen ungleichmäßig ausgelastet werden, da immer die am nächsten liegende Station angesteuert wird. Dadurch besteht die Gefahr eines „Verhungerns“ von Arbeitsstationen, die weiter entfernt sind.

Deshalb werden die Flussgegenstände nun in unterschiedlich starken Strömen auf alle Ausgänge verteilt, die zu möglichen Stationen führen. Die Stromstärke wird nach der Kirchhoff'schen Regel für parallele Widerstandsnetze [75], der Ströme in elektronischen Schaltungen folgen, berechnet. Zwischen Wegnetzen und elektronischen Schaltungen besteht eine Analogie, wenn man die Distanz über einen Ausgang zur Zielstation dem Widerstand R_a gleichsetzt. Gleichung 7.1 zeigt eine Formel zur Berechnung des Gesamtwiderstands R_G. Dieser wird in Gleichung 7.2 zur Berechnung der Ströme an den einzelnen Ausgängen verwendet.

$$R_G = \left(\sum_{a=1}^{A} \frac{1}{R_a} \right)^{-1} \tag{7.1}$$

$$I_a = \frac{R_G}{R_a} \tag{7.2}$$

7.5.4 Werkstückzentriertes, angebotsorientiertes Routing

Diese Strategie ist eine Ergänzung zum stationsorientierten Routing und basiert darauf, dass nach jeder durchgeführten Tätigkeit die nächste Arbeitsstation, unter Berücksichtigung aktueller Zustandsinformationen, neu bestimmt wird. Das Verfahren beruht auf dem Prinzip verhandlungsführender Agenten, das aus Agentensystemen bekannt ist [10, 64, 84, 108].

Diese Systeme bestehen aus autonom agierenden Softwareeinheiten, die kooperativ Aufgaben lösen [10, 13, 87]. Sie führen Verhandlungen durch und stehen dazu untereinander in Verbindung. Hierbei sind verschiedene Verhandlungsformen, bspw. einfache Angebotsrunden (*Contract Net* [97]), englische Auktion oder holländische Auktion mit mehreren Angebotsrunden, möglich [84].

Contract Net ist eine Form, bei der Dienstnehmer Aufträge anbieten, für die Dienstgeber jeweils ein Angebot – ohne Kenntnis der anderen Angebote – abgeben. Der Dienstnehmer entscheidet sich für das Günstigste [13, 77, 88]. Die Anwendung auf Flusssteuerungen kann auf zweierlei Wegen erfolgen. Entweder initiieren Werkstücke Angebotsrunden und Arbeitsstationen bieten unterschiedlich schnelle Fertigstellungszeiten, oder Arbeitsstationen offerieren ihre Fähigkeiten und Werkstücke bieten kurze Bearbeitungszeiten. Der erste Fall trägt zur Absenkung der Gesamtdurchlaufzeiten bei. Allerdings können keine Prioritäten von Werkstücken berücksichtigt werden, da Werkstücke keine Kenntnis voneinander haben und sie Aufträge unabhängig der Priorität der anderen erteilen. Im anderen Fall können zwar Prioritäten einbezogen werden, allerdings trägt diese Vorgehensweise weniger zur Absenkung der Gesamtdurchlaufzeiten bei, da Arbeitsstationen bei der Erteilung der Aufträge nicht die Auslastungen der anderen berücksichtigen.

Für den einfachen Nachweis der Eignung eines angebotsorientierten Verfahrens werden Prioritäten vernachlässigt. Werkstücke initiieren nach jeder durchgeführten Tätigkeit eine Angebotsrunde und vergeben dann den nächsten Arbeitsschritt an eine Arbeitsstation. Jede Station gibt, sofern sie funktionsbereit ist und die Tätigkeit durchführen kann, ein Gebot in Form von Kosten ab (Abb. 7.15). Die Kosten berechnen sich über eine Kostenfunktion, die eine approximierte Transportzeit zur Zielstation, Bearbeitungszeiten und Rüstzeiten für das betreffende und alle im Eingangspuffer der Arbeitsstation befindlichen Werkstücke berücksichtigt.

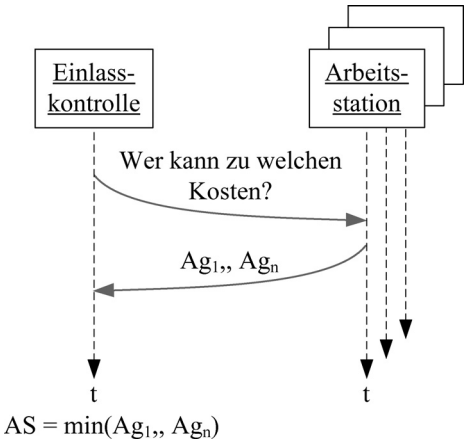


Abbildung 7.15: Sequenzdiagramm des angebotsorientierten Routings. Ag_n = Angebotsgeber; AS = Arbeitsstation.

Eine optionale Verfeinerung des Verfahrens besteht darin, die Angebotsrunde nach einer bestimmten Anzahl überschrittener Verzweigungen zu wiederholen, um auf dem Weg ein noch günstigeres Angebot zu ermitteln. Allerdings besteht die Gefahr des „Verhungerns“ von Werkstücken, indem sie ständig zwischen verschiedenen – abwechselnd günstigeren – Arbeitsstationen pendeln. Diese Gefahr lässt sich dadurch begrenzen, dass für jeden Arbeitsschritt nur eine maximale Anzahl an Angebotsrunden erlaubt wird.

Solange sich funktionsbereite Arbeitsstationen im System befinden, die die geforderte Tätigkeit ausüben können, kommt es zu keinen Blockaden. Die Ausgangswahl erfolgt auf Grundlage priorisierter Ausgänge durch das stationsorientierte Routing, so dass sowohl Staus als auch Rückstauungen durch übergelaufene Eingangspuffer von Arbeitsstationen vermieden werden.

7.6 Simulationsstudie zur Bewertung der Flussstrategien

Um die Leistungsfähigkeit der Flussstrategien miteinander zu vergleichen, wurde eine Ablaufsimulationsstudie anhand zweier allgemein gehaltener Produktionssysteme unter turbulenten Produktionsbedingungen durchgeführt. Als Simulator kam dabei *EM-Plant* von Tecnomatix [104] zum Einsatz.

Nachfolgend werden der Aufbau der Simulationsmodelle und die durchgeführten Experimente erläutert sowie die Ergebnisse diskutiert. Eine ausführlichere Darstellung findet sich in [30].

7.6.1 Simulationsmodelle

Die Simulationsmodelle der beiden Produktionssysteme sind eine einfachere Ring- sowie eine komplexere Matrixstruktur. Sie sind aus den Elementen *Arbeitsstation, Transfersegment, Kreuzung, Quelle* und *Senke* aufgebaut. Arbeitsstationen sind im Nebenschluss an ein Transfersystem angeschlossen, das sich aus den anderen Elementen zusammensetzt (Abb. 7.16). Werkstücke werden auf Werkstückträgern durch die Produktionssysteme geschleust.

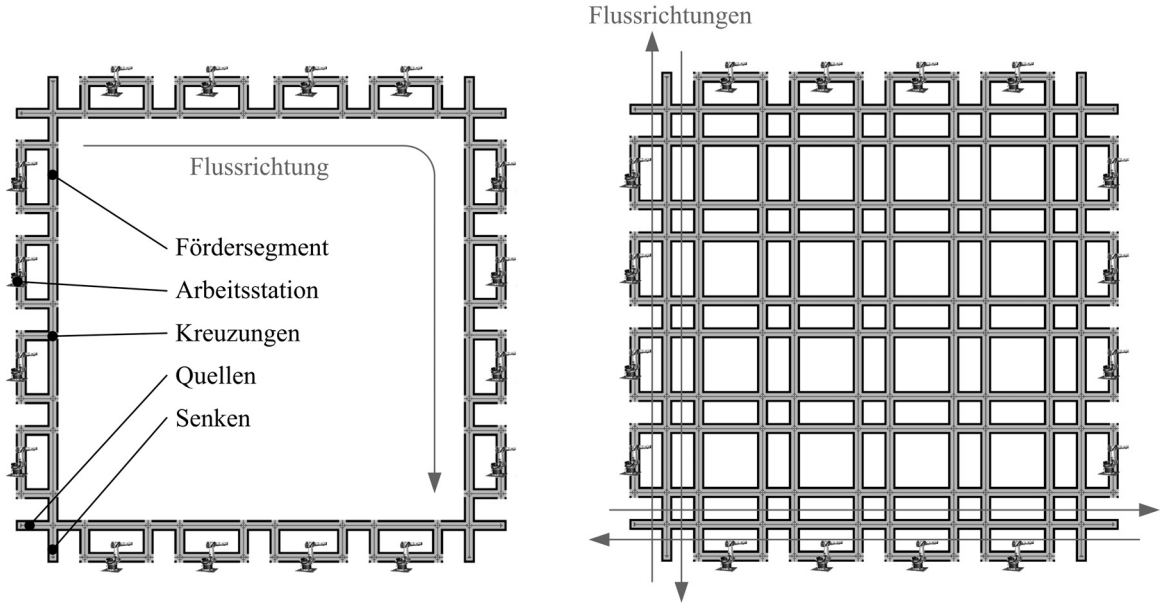


Abbildung 7.16: Simulationsmodelle. Links – Produktionssystem mit Ringstruktur. Rechts – Produktionssystem mit Matrixstruktur.

Die Arbeitsstationen entsprechen verschiedenen Maschinentypen und sind durch die Kapazität $K = 1$ und eine Liste ausführbarer Tätigkeiten charakterisiert. Für jede dieser Tätigkeiten sind zufallserzeugte Bearbeitungs- und Umrüstzeiten hinterlegt. Jede Arbeitsstation verfügt über FIFO-Eingangspuffer und -Ausgangspuffer in Form vorgeschalteter Transfersegmente. Alle Ursachen für die Nichtverfügbarkeit einer Arbeitsstation werden als Maschinenausfall aufgefasst, die zur Vereinfachung zyklisch und deterministisch mit konstanter Ausfalldauer auftreten. Ausgefallene Maschinen sind aus Sicht der Stoffflüsse unsichtbar, wodurch strukturelle Wandlungen der Produktionssysteme berücksichtigt werden.

Transfersegmente sind durch eine definierte Länge, Geschwindigkeit und Kapazität charakterisiert. Letztere hängt von der Segmentlänge und den Abmessungen der Werkstückträger ab. Für sie wurden mit 400x400mm typische Abmessungen gewählt. Jede Kreuzung verfügt über einen bis vier Eingänge bzw. Ausgänge, wobei mindestens ein Eingang und ein Ausgang vorhanden und belegt sind. Ihre Kapazität beträgt $K = 1$ und der Zeitbedarf zur Auswahl eines Ausgangs ist konstant.

Erzeugt werden verschiedene *Produkte* in mehreren *Varianten*, für die Arbeitspläne existieren. Ein Werkstück ist durch eine Produkt-Varianten-Kombination und einen Fortschrittszähler beschrieben, der den nächsten Arbeitsschritt indiziert. Alle Erzeugnisse sind zu typreinen Losen zusammengefasst und in einem Produktionsplan hinterlegt. Sie werden sukzessive in die Produktionslinie eingeschleust. Als Steuerungsprinzip kommt *Constant Work in Process (CONWIP)* zum Einsatz [45, 99], wobei als Obergrenze für die Anzahl der Werkstücke unterschiedliche WIP-Werte vorgegeben werden. So befinden sich zu jedem Zeitpunkt eine maximale Anzahl von Werkstücken in der Linie, um eine Überflutung der Produktionslinien zu verhindern. Andernfalls wäre ein Vergleich der unterschiedlichen Verfahren nicht möglich. Dazu wird ein Zähler bei jedem eingeschleusten Werkstück inkrementiert und bei jedem ausgeschleusten Werkstück dekrementiert. Nur wenn der Zähler kleiner der Obergrenze ist, werden neue Werkstücke eingelassen.

Um eine kurze Einschwingphase zu erzielen, befinden sich an allen vier Ecken Quellen und Senken für Flussgegenstände, so dass ein gleichmäßiger Zu- und Abfluss aus vier Richtungen möglich ist. Jedes Modell weist sechzehn Arbeitsstationen unterschiedlichen Typs auf. Jeder Maschinentyp kann zufallsgeneriert eine unterschiedliche Anzahl von Tätigkeiten teilredundant durchführen. Dabei sind jeweils ein Teil der Tätigkeiten exklusiv einem Maschinentyp und ein anderer Teil mehreren Maschinentypen zugeordnet. Jeder Maschinentyp kommt mindestens zweimal vor, um eine Redundanz aller Tätigkeiten sicherzustellen. Insgesamt können alle Maschinen alle erforderlichen Tätigkeiten durchführen. Um dies zu gewährleisten, erhält jeder Typ einen festen und einen variablen Anteil an Tätigkeiten. Insgesamt gibt es zehn Produkte mit fünf Varianten. Für die kleinste Variante sind fünf Tätigkeiten erforderlich, für die komplexeste Variante 50.

7.6.2 Simulationsexperimente

Insgesamt wurden 384 Einzelexperimente durchgeführt, bei denen jeweils 5000 Werkstücke produziert wurden. Die Experimente unterscheiden sich wie folgt:

- Matrix / Ring Struktur
- WIP-Wert 50 / 100 (maximale Werkstücke im System)
- 4 / 8 Maschinentypen mit zufallsgenerierten Tätigkeiten. Die Bearbeitungszeiten betragen tätigkeitsspezifisch zwischen 30 und 60 Sekunden und erfordern Rüstzeiten zwischen 120 und 300 Sekunden.
- Ohne / mit Maschinenausfällen. Diese treten initial für jeden Maschinentyp zufällig zwischen der ersten und fünften Stunde auf, danach deterministisch alle drei Stunden. Die Ausfalldauer beträgt jeweils konstant fünfzehn Minuten.
- Losgrößen 4 / 4 bis 12 / 4 bis 48 bei jeweils $1/3$ aller Experimente.
- 15m/min / 3,75m/min Geschwindigkeiten
- 4 Routing-Strategien

7.6.3 Simulationsergebnisse

Um die Strategien zu bewerten erfolgt der Vergleich anhand der Messgrößen *Gesamtbearbeitungszeit, durchschnittliche Durchlaufzeit pro Werkstück* und die *durchschnittliche Anzahl an Umrüstvorgängen*, unter Einbeziehung von Standardabweichungen.

Alle Einzelergebnisse weisen dieselbe Tendenz auf. Das stationsorientierte Routing unterliegt den anderen Verfahren. Die beiden tätigkeitsorientierten Strategien unterscheiden sich nicht wesentlich in ihrer Leistung. Dies liegt vermutlich daran, dass beim Lastausgleich der gleichmäßigeren Stationsauslastung längere Transferzeiten und eine größere Zahl von Umrüstvorgängen entgegen stehen. Das beste Resultat erzielt das angebotsorientierte Routing.

Beispielhaft soll dies anhand des Szenarios Matrixstruktur mit WIP-Wert 100, acht verschiedenen Maschinentypen ohne Stationsausfälle und einer Geschwindigkeit von 3.75 m/min gezeigt werden. Hierbei wurden die Ergebnisse der Einzelexperimente mit unterschiedlichen Losgrößen gemittelt (Abb.

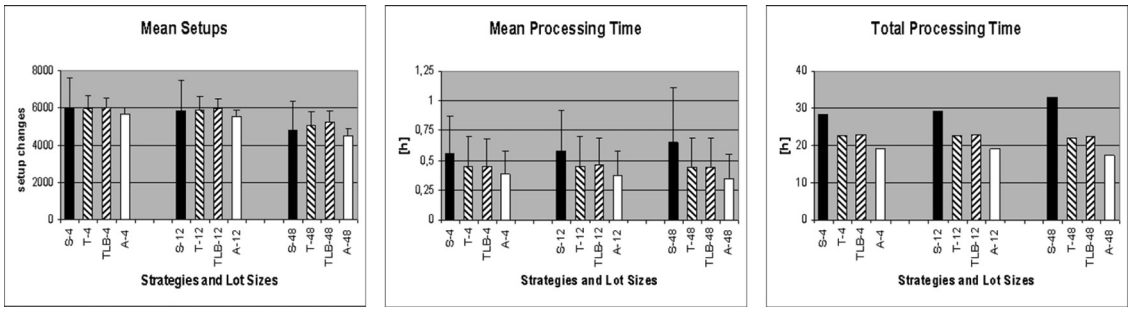


Abbildung 7.17: Einzelergebnisse bei verschiedenen Losgrößen. Links – Anzahl der Umrüstungen. Mitte – Mittlere Durchlaufzeit pro Werkstück. Rechts – Gesamtbearbeitungszeit aller Werkstücke. S = stationsorientiertes Routing; T = tätigkeitsorientiertes Routing; TLB = tätigkeitsorientiertes Routing mit Lastausgleich; A = angebotsorientiertes Routing. Losgrößen 4, 4 bis 12, 4 bis 48.

7.17). Es zeigt sich, dass beim stationsorientierten Routing die mittleren Bearbeitungszeiten, einhergehend mit größeren Losen, ansteigen und die Anzahl der Umrüstvorgänge dabei absinkt. Dies ist mit starren Stationsbelegungen und dadurch überlaufenden Eingangspuffern und Rückstauungen erklärbar. Das tätigkeitsorientierte Routing wird von unterschiedlichen Losgrößen wenig beeinflusst. Zwar sinken die Umrüstvorgänge, die mittleren Bearbeitungszeiten und die Gesamtdurchlaufzeit ändern sich jedoch wenig. Staus werden umgangen und bei vollen Eingangspuffern entweder Warteschleifen „gedreht“ oder alternative Arbeitsstationen angefahren. Dies geschieht auch beim angebotsorientierten Routing. Allerdings sinken hier alle drei Messgrößen bei steigenden Losgrößen kontinuierlich.

Um generelle Aussagen treffen zu können wurden alle Ergebnisse der Experimente – ohne Gewichtung Einzelner – gemittelt, wie in Abbildung 7.18 gezeigt. Bei der Darstellung sind die Werte auf die Ergebnisse des stationsorientierten Routings normiert und zeigen relative Veränderungen. Trotzdem sich – über alle Experimente gemittelt – eine steigende mittlere Anzahl von Umrüstungen zeigt, sinken die mittleren Bearbeitungszeiten deutlich. Es sei betont, dass die stationsorientierten Routen per Zufall generiert wurden und somit mögliches manuelles Verbesserungspotenzial nicht genutzt wurde. Daher sind die *absoluten Zahlen* nicht als absolut gültig anzunehmen. In ihrer Tendenz sind die *relativen* Ergebnisse jedoch schlüssig, da bei hochdynamischen Systemen starre, unflexible Verfahren der Vorabplanung (*PR – Production Reservation*) gegenüber Verfahren der Einzelschrittplanung (*SSPR – Single Step Production Reservation*) in jedem Fall im Nachteil sind [44, 64, 108].

7.7 Integration der besten Strategie

Die Wahl der besten Strategie zur Selbstorganisation von Stoffflüssen fällt nach Auswertung der Simulationsstudie auf das angebotsorientierte Routing. Neben den besten Ergebnissen verfügt sie über den Vorteil, das stationsorientierte Routing zu ergänzen. Dadurch ist ein effizienter Transport von Werkstücken und ein effektiver Transport der restlichen Stoffe gleichzeitig möglich. So besteht bei der Nutzung von Transfersystemen volle Flexibilität, so dass sowohl reine Werkstückflüsse, reine (Rest-)Stoffflüsse als auch Mischflüsse realisiert werden können.

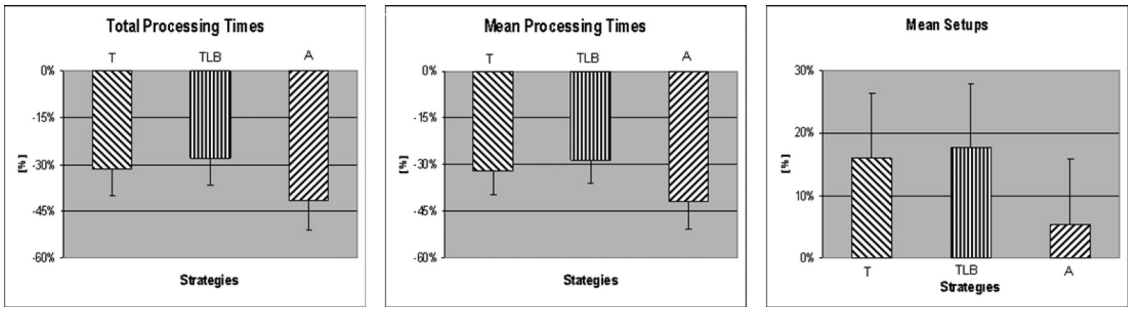


Abbildung 7.18: Gesamtergebnisse gemittelt über alle Einzelergebnisse und normiert auf stationsorientiertes Routing. Links – Gesamtbearbeitungszeit von 5000 Werkstücken unterschiedlicher Produkte und Varianten. Mitte – Mittlere Durchlaufzeit pro Werkstück. Rechts – Anzahl der Umrüstungen. T = tätigkeitsorientiertes Routing; TLB = tätigkeitsorientiertes Routing mit Lastausgleich; A = angebotsorientiertes Routing.

Ankopplung an die Strukturerkennung

Durch Ankopplung an die Selbsterkennung der strukturellen Beziehungen zwischen Entitäten (Kap. 6.4) und die Nutzung der dadurch verfügbaren Stationsroutenpläne für das stationsorientierte Routing, ist eine Integration der Strategie für *strukturierte Bereiche* (Kap. 4.2) einfach möglich. Bei *unstrukturierten Bereichen* ist nur eine Anwesenheitserkennung möglich, so dass das Wegenetz eines Transportsystems, bspw. aus Layout-Informationen der vorgelagerten Planungsphase abgeleitet werden muss.

Die reine Anwesenheitserkennung kann allerdings unmittelbar zur Kompensation von Störungen, Pausen, Wartungen und Instandsetzungsarbeiten genutzt werden, indem diese einheitlich als Ausfälle verstanden werden. Durch die Erweiterung der Datenstruktur der Nachrichten um ein weiteres Feld (*State*) zur Zustandsübertragung (Abb. 6.4, e, S. 79) sind alle Entitäten stets informiert. Echte Entitätenausfälle werden mittels Timeout festgestellt (Kap. 6.3). Durch eine Unterscheidbarkeit von Entitäten nach Verzweigungen, Quellen, Senken und Arbeitsstationen – Letztere darüber hinaus nach Gerätetypen – stehen weitere elementare Informationen zur Verfügung. Deshalb wird die Datenstruktur zusätzlich noch um ein Typfeld (*type*) erweitert. So können passgenaue Handlungsalternativen unmittelbar nach Erkennung von Zustandsänderungen eingeleitet werden.

Ermittlung der Arbeitspläne

Zur Durchführung der Angebotsrunden muss die nächste durchzuführende Tätigkeit eines Werkstücks bekannt sein. Die Beschaffung dieser Information hängt von der Informationsspeicherung am Werkstück(träger) ab. Entweder ist der Arbeitsplan des Werkstücks auf diesem gespeichert oder muss über die Arbeitsplanverwaltung (Kap. 9.3) bezogen werden.

Hierzu bietet diese die Funktionsschnittellen *GetWorkingPlan(prodVarId)* und *GetNextTaskId(prodVarId, index)* an (Abb. 9.12, S. 144). Die erste Funktion liefert unter Angabe der Identität *prodVarId* einer Produkt-Varianten-Kombination einen Arbeitsplan und die zweite Funktion unter Angabe von *prodVarId* und einem Fortschrittszähler *index* die Identität der nächsten Tätigkeit zurück.

```
FACET_BID = [
   GetBid(STRING  taskId): FLOAT
]
```

Abbildung 7.19: Facette zur Angebotsanfrage.

Durchführung von Angebotsrunden

Die eigentliche Angebotsrunde findet dadurch statt, dass eine Arbeitsstation stellvertretend für ein gerade fertig bearbeitetes Werkstück reihum Anfragen an Nachfolgestationen durchführt. Hierzu besitzen alle Arbeitsstationen eine Schnittstellenfacette (Abb. 7.19), die einzig die Funktion *GetBid(taskId)* als Schnittstelle anbietet. Durch Aufruf dieser Funktion liefert eine Arbeitsstation – unter Berücksichtigung ihrer Zustände – einen Kostenwert zurück. So ist das günstigste Angebot ermittelbar.

Informationsspeicherung auf Flussgegenständen

Für die Behandlung von Flussgegenständen durch Transport und Bearbeitung sind auf diesen Informationen zu speichern, woraus sich Zielstation und – bei Werkstücken – die nächste durchzuführende Tätigkeit ableiten lassen. Dabei sind unterschiedliche Varianten möglich.

Die *erste* Möglichkeit besteht darin, dem Gegenstand lediglich eine Identität mitzugeben. Allerdings wird dann eine Flussgegenstandsüberwachung erforderlich, die die Identität nach Zielstation und für Werkstücke darüber hinaus nach Produkt-Varianten-Kombination und Bearbeitungszustand auflöst. An jeder Verzweigung muss die Zielstation ermittelt werden. Die *zweite* Möglichkeit setzt beschreibbare Informationsspeicher voraus und erweitert die erste Variante, indem die Zielstation mit auf dem Flussgegenstand hinterlegt wird. So ist nur noch für Werkstücke an den Arbeitsstationen die nächste Tätigkeit zu ermitteln, indem die Identität nach Produkt-Varianten-Kombination und Bearbeitungszustand aufgelöst und im Anschluss die Arbeitsplanverwaltung befragt wird. Dadurch ist der gesamte Stoffffluss autonom durchführbar, d.h. vom Start bis zum Ziel ist keine Interaktion mehr mit zentralen Diensten erforderlich. In der *dritten* Variante werden Identität, Zielstation, Produkt-Varianten-Kombination und Fortschrittszähler auf dem Flussgegenstand abgespeichert. Dadurch kann die zentrale Flussgegenstandsüberwachung vermieden werden. Nur noch die Arbeitsstationen müssen mit der Arbeitsplanverwaltung interagieren, um die durchzuführende Tätigkeit zu bestimmen. Die *vierte* Möglichkeit schließlich setzt größere Informationsspeicher voraus und speichert den Arbeitsplan mit ab. Enthält dieser die notwendigen Rezepte (4.5.2) und Prozessparameter, ist auch an Arbeitsstationen keine Interaktion mehr mit der Arbeitsplanverwaltung oder Rezeptverwaltung erforderlich. Dies stellt jedoch hohe Anforderungen an Rezepte und deren Verarbeitung in Produktionseinrichtungen, da für jede Tätigkeit nur ein einziges Rezept existieren darf, das von allen Geräten gleichermaßen verstanden wird. Dieser Ansatz ist bei heutigem Stand der Technik in der Praxis nicht umzusetzen.

Aus praktischen Erwägungen wird die *dritte* Variante zur Informationsspeicherung auf Flussgegenständen favorisiert. Durch die Mitspeicherung der Zielstation eignet sie sich zum Transport aller Stoffe.

Kapitel 8

Selbstorganisation wandlungsfähiger Produktionsanlagen

Mit der Selbsterkennung von Entitäten und ihren strukturellen Verbindungen sowie der Selbstorganisation von Stoffflüssen sind wichtige Voraussetzungen für selbstorganisierende, wandelbare Produktionslinien und -inseln – im Weiteren vereinheitlichend „Produktionssystem“ – und darüber hinaus für selbstorganisierende, wandelbare Produktionsanlagen geschaffen.

Um Produktionssysteme aufbauen zu können sind als weitere Bestandteile Arbeitsstationen (prozessausführende Einheiten), eine Auftragsverwaltung und Lager erforderlich. Über Letztere werden Systeme und Arbeitsstationen mit Stoffen versorgt und fertige Erzeugnisse aus diesen entfernt. Sie stellen Bindeglieder zu vorhergehenden und nachfolgenden Produktionssystemen dar, wodurch die Herstellung komplexer Erzeugnisse möglich wird. Ein oder mehrere Produktionssysteme zusammen mit globalen Eingangs- und Ausgangslagern bilden eine Produktionsanlage, die mit der Produktionsplanung und -steuerung zum Zwecke der Auftragsfreigabe und der Materialwirtschaft in Verbindung steht.

Im Weiteren wird ein generisches, selbstadaptierendes Lagermodell mit einer steuerungsmodellbasierten Ablaufsteuerung vorgestellt, das die notwendigen Lagerformen abdeckt und auf den verschiedenen Ebenen eingesetzt werden kann. Anschließend erfolgt die Betrachtung von Produktionssystemen, einer Auftragsverwaltung sowie einem Grundmodell für Arbeitsstationen. Den Abschluss bildet schließlich die Beschreibung des Aufbaus von Produktionsanlagen.

Da Transfer das Bindeglied zwischen allen Bestandteilen der Produktionsanlage darstellt und Transferhilfsmittel darin eine tragende Rolle einnehmen, werden diese jedoch zuerst kurz beschrieben.

8.1 Transferhilfsmittel

Für den Transfer von Stoffen kommen Transferhilfsmittel zum Einsatz, worunter Werkstückträger, Paletten, Kisten, usw. verstanden werden (Kap. 2.1.1). Verallgemeinert handelt es sich um mobile Container mit Kapazität $K \geq 1$, die Stoffe (Kap. 2.1.1) in unterschiedlicher Anzahl sortenrein oder gemischt aufnehmen können. Jeder Stoff wird als einzeln handhabbar aufgefasst, d.h. Schüttgut, lose Teile, Flüssigkeiten oder Gase sind entsprechend zu einer Einheit verpackt. Stoffe können in Formnestern platziert, frei positioniert oder geschlichtet werden, wobei mehrere Schichten möglich sind (Abb. 8.1).

Container weisen aufgabenspezifisch unterschiedliche Formen und Abmaße auf und verfügen optional über Formnester, Fächer, etc. Das korrekte *Beladen* und *Entladen* hängt von diesen Eigenschaften ab.

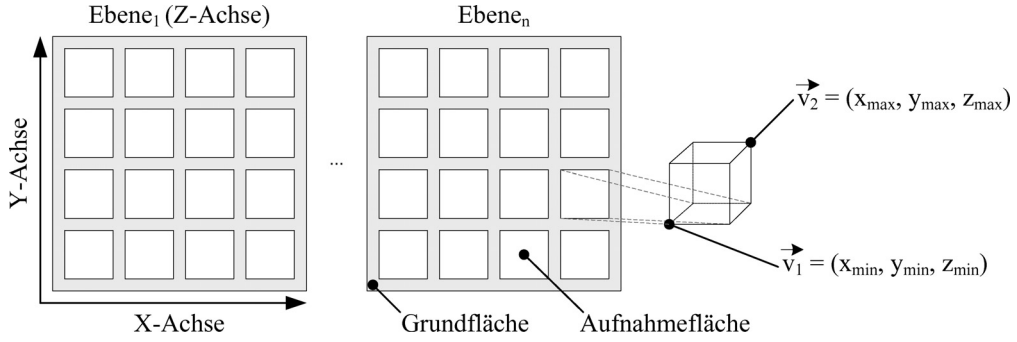


Abbildung 8.1: Allgemeines Modell eines Transferhilfsmittels zur Aufname von Stoffen in allen drei Dimensionen. Die Ebenen entsprechen verschiedenen Packschichten. Form und Abmaße der einzelnen Lagervolumina sind fallabhängig und können innerhalb eines Transferhilfsmittels variieren.

Für das *Beladen* eines Containers sind dessen Geometrie und Ablageplätze sowie die Geometrien der Stoffe von Bedeutung. Durch Typisierung von Containern und Stoffen (*carrierTypeId*, *typeId* bzw. *prodVarId* (für Produkt-Varianten-Kombinationen (siehe auch Kap. 7.2) und Stofftypen gilt: $\{$ *prodVarId* $\} \subseteq \{$ *typeId* $\}$, weshalb bei jeder Verwendung *typeId* auch *prodVarId* gemeint ist, sofern nicht explizit unterschieden wird) und Einsatz einer zentralen Geometriedatenverwaltung sind diese Informationen beziehbar. Für das *Entladen* eines Stoffs sind dessen Position im Raum und Geometrie von Bedeutung. Die Geometrie wird dabei vereinfacht als dreidimensionale Hüllbox beschrieben, die durch zwei räumliche Koordinaten $\vec{v_1} = (x_{min}, y_{min}, z_{min})$ und $\vec{v_2} = (x_{max}, y_{max}, z_{max})$ an den Enden der Raumdiagonalen angegeben wird (Abb. 8.1, rechts). Dadurch lässt sich der Mittelpunkt c eines Stoffs berechnen:

$$c = \vec{v_1} + \frac{\vec{v_2} - \vec{v_1}}{2} \tag{8.1}$$

Unter der Voraussetzung, dass sich ein Stoff immer in einer fest definierten räumlichen Ausrichtung befinden muss, ist die eindeutige Lage im Raum, bezogen auf den Ursprung des Containers, bekannt. Durch Angabe des Stofftyps *typeId* bzw. der Produkt-Varianten-Kombination *prodVarId* ist die tatsächliche Geometrie von der bereits postulierten Geometriedatenverwaltung zu ermitteln, wodurch ein dediziertes manuelles und automatisiertes Greifen möglich wird. An dieser Stelle sei angemerkt, dass die Position eines Stoffs in der Praxis Toleranzen unterworfen ist. Es ist jedoch Aufgabe der Greiftechnik, diese zu kompensieren.

Zusammen mit den benötigten Informationen für die Selbstorganisation der Stoffflüsse ergibt sich für Transferhilfsmittel die in Abbildung 8.2 dargestellte Datenstruktur, die jedem Container auf einem wiederbeschreibbaren Informationsspeicher beigefügt wird. Ergänzend zu den bisherigen Ausführungen verfügt jeder Container und jeder einzelne Stoff über eine Identität (*carrierTypeId, partId*).

Somit stehen alle Informationen zur Stoffidentifizierung und -entnahme von Transferhilfsmitteln in Lagern und Arbeitsstationen zur Verfügung.

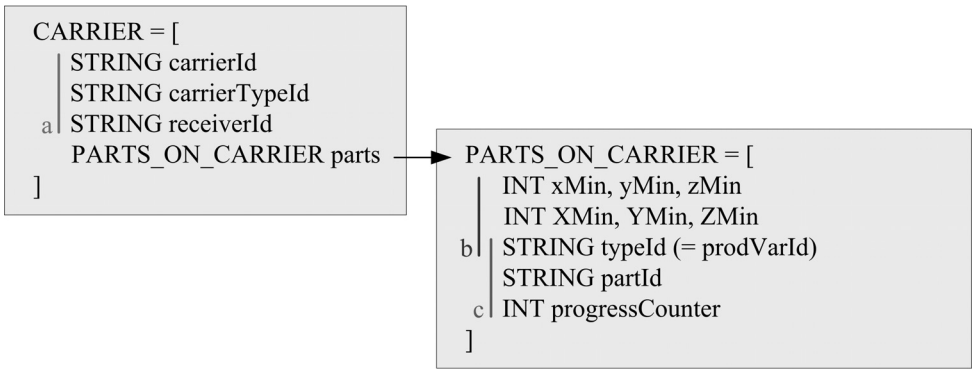


Abbildung 8.2: Datenstrukturen für Transferhilfsmittel. a – Informationen für Transport. b – Informationen zur Position eines Stoffs zur Entnahme. c – Informationen zur Werkstückbearbeitung bzw. zum Be- und Entladen.

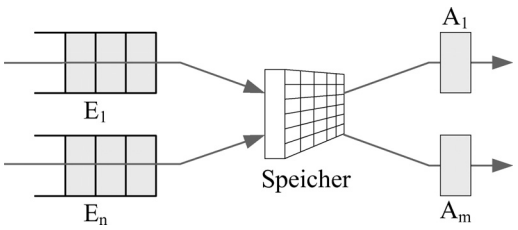


Abbildung 8.3: Schema eines Lagers mit Ein- und Ausgängen sowie dem Speicher. A_m = Ausgang; E_n = Eingang.

8.2 Generisches Lager

Zur Selbstorganisation der Produktionsabläufe leisten Lager einen wesentlichen Beitrag und werden zu diesem Zweck in Produktionslinien und -inseln wie auch auf Ebene der Produktionsanlage benötigt (Kap. 8.3, 8.4).

Grundsätzlich speichert ein Lager Gegenstände (Stoffe, Transferhilfsmittel) für spätere Transfers oder Bearbeitungsschritte zwischen. Es verfügt über einen Speicher S mit begrenzter Kapazität $K \geq 1$. Gegenstände gelangen über gepufferte Eingänge $\{E\}\ mit\ |\{E\}| \geq 1$ hinein und über Ausgänge $\{A\}\ mit\ |\{A\}| \geq 1$ hinaus (Abb. 8.3). Ein- und Auslagerungen werden von Ausführungseinheiten $\{AE\}\ mit\ |\{AE\}| \geq 1$ übernommen. Von der konkreten physischen Beschaffenheit des Lagers wird abstrahiert, da sie aus Sicht einer effektiven Selbstorganisation nicht von Bedeutung ist.

Ankommende Transferhilfsmittel werden im Lager ggf. von ihren geladenen Stoffen getrennt und alle Gegenstände gespeichert. Umgekehrt erfolgt bei Auslagerungen im Lager ggf. eine Kommissionierung auf Transferhilfsmittel sowie ein Beschreiben der Informationsspeicher mit Daten (8.1).

Das Lager verfügt über eine Verwaltung, die einerseits zur Überwachung der Lagerbestände und andererseits zur Veranlassung physischer Ein- und Auslagerungen mit Hilfe der Ausführungseinheiten dient. Für Einlagerungen ist sie mit den Eingängen verbunden und entscheidet bei mehreren anwesenden Transferhilfsmitteln über die Auswahlreihenfolge. Das Lager implementiert eine Schnittstelle zur Einlastung von Auslageraufträgen. Eingehende Stoffe werden nach erfolgter Speicherung als Lager-

bestand verbucht, ausgehende Stoffe erst aus dem Lagerbestand entfernt, wenn das Transferhilfsmittel das Lager über einen der Ausgänge verlassen hat.

Lager können als *Eingangs-, Zwischen-* und *Ausgangslager* eingesetzt werden. Eingangslager bilden eingehende und Ausgangslager ausgehende Systemgrenzen von Produktionslinien, -inseln sowie der übergeordneten Produktionsanlage. Zwischenlager gelangen innerhalb der entsprechenden Systeme zum Einsatz. Eingangs- und Ausgangslager können an anderen Lagern selbstständig Bestellvorgänge auslösen, um ihre Bestände aufzufüllen. Eingangs- und Ausgangslager können von Wandlung betroffen sein, da sich das Spektrum der zu lagernden Stofftypen verändern kann. Dieses Phänomen ist für die Selbstorganisation der Produktion von Bedeutung (Kap. 8.3.2, 8.4).

8.2.1 Modell zur Ablaufsteuerung

Das Steuerungsmodell basiert auf der auf Wandelbarkeit ausgerichteten Softwarearchitektur (Kap. 5). Es ist gleichermaßen für die drei Lagertypen ausgelegt und besteht aus Knoten, zwei Grundfacetten (5.6) und zwei spezifischen Facetten, die in dem Knoten L (Kap. 5.3.1) zusammengefasst sind (Abb. 8.6).

Jeder Eingang wird von einem eigenen Knoten repräsentiert, mit einer Digital-Out-Facette V zum Vereinzeln sowie einer String-In-Facette ID_E zur Erkennung der Anwesenheit einzulagernder Gegenstände und zum gleichzeitigen Auslesen der Informationsspeicher der Transferhilfsmittel. Dadurch sind die Art und jeweilige Anzahl der Stofftypen sowie die Identität und Typ (Kap. 8.1) der Transferhilfsmittel bestimmbar. Für jeden Ausgang ist eine String-In-Facette ID_A vorhanden, die den gleichen Zweck wie ID_E bei auszulagernden Gegenständen erfüllt. Die Lagerverwaltung ist in einer einzelnen Facette LV integriert. Als Schnittstelle zu den Ausführungseinheiten ist eine Ausführungssteuerungsfacette $Strg$ vorhanden.

Aufbau der spezifischen Facetten

Lagerverwaltung (Stock Manager): Die Struktur der Facette ist in Abbildung 8.4 gezeigt. Für die Interaktion mit der Ausführungssteuerung, d.h. zum Anstoßen von Einlager- und Auslagervorgängen, referenziert sie diese (*ctrl*). Die Eingänge müssen einerseits zur Erkennung ankommender Transferhilfsmittel überwacht und andererseits das Einlassen kontrolliert werden, weshalb zum einen der Callback *EntranceNewCarrier(data)* implementiert ist und zum anderen Referenzen auf die Vereinzeler aller Eingänge in einer Liste (*vs*) gespeichert werden. Um über das Lager verlassende Transferhilfsmittel informiert zu werden implementiert die Facette den Callback *ExitNewCarrier(data)*. Die Schnittstelle für die Einlastung von Auslageraufträgen stellt die Prozedur *ReleaseParts(parts, carrierTypeId, receiverId)* dar. Zum Zwecke der Nachbestellung verfügt die Facette über Referenzen (*smSALs*) auf die Ausgangslager aller anderen Produktionssysteme und zum Eingangslager der Produktionsanlage (Kap. 8.3, 8.4). Der Mechanismus der Nachbestellung beruht unter anderem auf dem Publisher-Subscriber-Prinzip, wozu ein Publisher *RequestForOrder(parts, carrierTypeId, receiverId)* sowie ein gleichnamiger Callback vorhanden ist. Einen weiteren Bestandteil der Nachbestellung bildet die Prozedur *OrderedPartsDeliverableBy(stockManager, parts)*.

```
FACET_STOCK_MANAGER = [
Facets = [
   FACET_DIGITAL_OUTS vs
   FACET_STOCK_PHYSICAL_CONTROL ctrl
   FACET_STOCK_MANAGERS smSALs
   Publishers = [
        RequestForOrder(PARTS parts, STRING carrierTypeId, LOCATOR receiver)
   ]
   SubscriptionCallbacks = [
        EntranceNewCarrier(STRING data)          //Identität des Eingangs ist über den mitgesendeten
        ExitNewCarrier(STRING data)              //Locator des Publishers ermittelbar
        RequestForOrder(PARTS parts, STRING carrierTypeId, LOCATOR receiver)
   ]
   ReleaseParts(PARTS parts, STRING carrierTypeId, LOCATOR receiver)
    ↳ THROWS NotInStock(PARTS parts)
   OrderedPartsDeliverableBy(LOCATOR stockManager, PARTS parts)

   SetStockType(STRING stockType)
   SetEntranceSelectFunction(FUNCTION func)
   SetReleaseSelectFunction(FUNCTION func)
   SetOrderFunction(FUNCTION func)
   SetStorables(STRINGS typeIds)
]
```

Abbildung 8.4: Facette zur Lagerverwaltung.

```
FACET_STOCK_PHYSICAL_CONTROL = [
   Facets = [
        FACET_STOCK_MANAGER sm
   ]
   PlaceInStock(LOCATOR entrance, PARTS parts)
   ReleaseFromStock(LOCATOR exit, PARTS parts, STRING carrierTypeId, STRING receiverId)
]
```

Abbildung 8.5: Facette als Schnittstelle zur physischen Lagersteuerung.

Die Lagerverwaltung implementiert darüber hinaus fünf Schnittstellen zur Konfiguration. Mit der ersten Prozedur *SetStockType(stockType)* wird einer der drei Lagertypen festgelegt. Die Prozeduren *SetEntranceSelectFunction(func), SetReleaseSelectFunction(func)* und *SetOrderFunction(func)* dienen zur Übergabe jeweils einer Funktion, die eine Strategie zur Auswahl von Einlageraufträgen, Auslageraufträgen sowie zur Auslösung von Nachbestellungen implementiert. Die letzte Funktion *SetStorables(typeIds)* ist für die Adaption eines Lagers an gewandelte Produktionsbedingungen von Bedeutung, da darüber festgelegt wird, welche Stofftypen zu speichern sind.

Ausführungssteuerung (Stock Physical Control): Abbildung 8.5 zeigt die Schnittstellenfacette zur physischen Ausführungssteuerung. Für die Veranlassung von Einlagerungen steht die Prozedur *PlaceInStock(entrance, parts)* zur Verfügung. Die Prozedur bewirkt durch die Zustandsänderung einer der Ausführungseinheiten die Bereitschaft zur physischen Übernahme des Transferhilfsmittels und kehrt erst danach zurück. Für die Durchführung von Auslagerungen implementiert die Facette die Prozedur *ReleaseFromStock(exit, parts, carrierTypeId, receiverId)*.

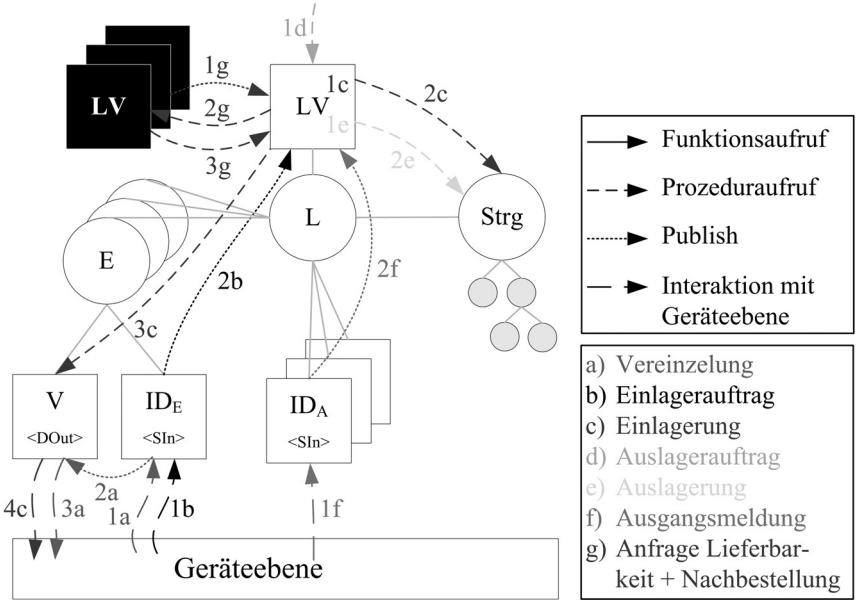


Abbildung 8.6: Generisches Modell der Lagersteuerung. Die invertierten Facetten sind Bestandteil anderer Lager. E = Eingangsknoten; ID_A = Identifizierung ankommender Gegenstände; ID_E = Identifizierung ausgehender Gegenstände; L = Lager; LV = Lagerverwaltung; $Strg$ = Ausführungssteuerung; V = Vereinzeler.

Informationelle Verknüpfung und Steuerungsablauf

Der Steuerungsablauf gliedert sich in verschiedene Reflexketten (Kap. 5.4) und Aktionsketten.

Ein Einlagerauftrag: entsteht, wenn die String-In-Facette ID_E die Anwesenheit eines Transferthilfsmittels registriert und dessen Informationsspeicher ausgelesen hat. Mittels des Publishers *StringChanged(string)* signalisiert ID_E die Zeichenkette über den Callback *EntranceNewCarrier(data)* an die Eingangsverwaltung. Aus dieser werden Art und Anzahl der einzulagernden Gegenstände ermittelt und mit dem Eingangslocator als Einlagerauftrag in einer Warteliste gespeichert (Abb. 8.6, 1b bis 2b).

Die Vereinzelung: erfolgt über V, der hierzu am Publisher *NewString* von ID_E registriert ist. Sobald eine Zugangsmeldung eintrifft, blockiert er den Eingang in das Lager (Abb. 8.6, 1a bis 3a).

Die Einlagerung: erfolgt, wenn Einlageraufträge vorhanden und nicht alle Ausführungseinheiten belegt sind und die Lagerverwaltung LV aufgrund ihrer Auswahlstrategien für Ein- und Auslagerungen einen entsprechenden Auftrag auswählt. LV lässt dann durch Aufruf der synchronen Prozedur *PlaceInStock(entrance, parts)* an der Ausführungssteuerung die Einlagerbereitschaft eines der Ausführungseinheiten herstellen und entblockiert danach den passenden Vereinzeler (*SetState(False)*) (Abb. 8.6, 1c bis 4c). Abschließend gleicht LV seine Lagerbestandsinformationen an.

Ein Auslagerauftrag: entsteht durch Aufruf einer der beiden Schnittstellen an der Lagerverwaltung (Abb. 8.6, 1d).

Eine Auslagerung: erfolgt, wenn Auslageraufträge vorhanden und nicht alle Ausführungseinheiten belegt sind und die Lagerverwaltung LV aufgrund ihrer Auswahlstrategien für Ein- und Auslagerungen

einen entsprechenden Auftrag auswählt. LV veranlasst den Vorgang an der Ausführungssteuerung durch Aufruf der Prozedur *ReleaseFromStock(exit, parts, carrierTypeId, receiverId)*, woraufhin das Lager ggf. die Kommissionierung, das Beschreiben des Informationsspeichers des Transferhilfsmittels veranlasst und den Gegenstand ausschleust (Abb. 8.6, 1e bis 2e).

Eine Ausgangsmeldung: wird über ID_A festgestellt. Dieser signalisiert den Vorgang der Lagerverwaltung, die ihre Lagerbestandsinformationen anpasst und ggf. einen nächsten Auftrag generiert (Abb. 8.6, 1f bis 2f).

Eine Teilebestellung: wird ausgelöst, wenn die Lagerverwaltung aufgrund ihrer Strategie zur Bestellauslösung einen Bedarf ermittelt. Über den Publisher *RequestForOrder(parts, carrierTypeId, receiverId)* publiziert sie eine Liste unterschiedlicher Stofftypen und Bedarfsmengen. Über den gleichnamigen Callback empfangen alle Lager dieses Ereignis und überprüfen daraufhin ihre Lagerbestände. Jedes Lager, das mindestens eine Teilmenge der benötigten Stoffe liefern kann, meldet dies über die Prozedur *OrderedPartsDeliverableBy(stockManager, parts)*. LV lastet dann gezielt an einem oder mehreren Lagern Bestellvorgänge über die Prozedur *ReleaseParts(parts, carrierTypeId, receiverId)* ein (Abb. 8.6, 1g bis 3g).

8.2.2 Eingriffspunkte zur effizienten Steuerung

Die Effizienz eines Lagers ist – neben seiner inneren Organisation – abhängig von den gewählten Strategien zur Auswahl von Einlager- und Auslageraufträgen. Darüber hinaus hat die Bestellstrategie eine noch größere Bedeutung, da von ihr die Lieferfähigkeit mit abhängt. Für alle drei Bereiche gilt, dass es allgemein keine optimalen Strategien gibt, sondern diese fallspezifisch auszuwählen sind. Deshalb werden auch keine Strategien fest vorgegeben. Stattdessen implementiert die Lagerverwaltung Schnittstellen, über die als Funktionen implementierte Strategien übergeben werden können. Diese Funktionen verfügen über eine einheitliche Schnittstelle und werden immer dann aufgerufen, wenn eine Entscheidung zu fällen ist (die Funktion zur Ermittlung von Bestellbedarfen wird bei jeder Auslagerung aufgerufen). Der Mechanismus ist identisch zur Auswahl von Eingängen an Verzweigungen, weshalb auf Abbildung 7.13 (S. 107) verwiesen sei. Um alle erforderlichen Informationen für eine fundierte Entscheidung treffen zu können, wird den Funktionen als Eingangsparameter eine Referenz auf die Lagerverwaltung mit übergeben. Darüber steht diesen der vollständige Zugriff auf das gesamte Steuerungsmodell der Anlage – und damit auf das gesamte „Anlagenwissen“ – frei.

Die Auswahl einer Einlagerung: kann bspw. Transferhilfsmittel bevorzugen, die Stoffe aus Bestellungen anliefern, um die Lieferfähigkeit zu unterstützen. Eine andere Strategie kann in einer umrüstminimalen Einlagerung bestehen, wenn das Lager über Kommissionierstellen verfügt und stoffspezifische Lagerhilfsmittel zu beladen sind (gilt analog für Auslagerungen). Weitere mögliche Ansätze sind *First-In-First-Out (FIFO)* und *kürzeste / längste Operationszeit (KOZ / LOZ)* [56].

Die Auswahl einer Auslagerung: kann ebenfalls nach *FIFO* oder *KOZ / LOZ* erfolgen. Es sind aber auch Strategien denkbar, die erzeugnisspezifisch auswählen, etwa nach *frühestem Plan-Endtermin* oder *geringstem Restschlupf* [56]. Allerdings sind für diese Strategien die Schnittstellen der Lagerverwaltung derart zu erweitern, dass zu jeder auszulagernden Stoffeinheit eine Zuordnung zu einem Erzeugnis möglich wird.

Die Auslösung von Bestellvorgängen: kann in Bestellzyklen mit variablen Bestellmengen ((T,S)-Regel) oder zu Bestellpunkten mit festen ((s, Q)-Regel) bzw. ebenfalls variablen Bestellmengen ((s, S)-Regel) erfolgen [2]. Hier sind beispielhaft das *Bestellbestandsverfahren (fixe / variable Bestellgrößen)* und das *Bestellrythmusverfahren* zu nennen [56].

8.2.3 Unterstützung der Wandelbarkeit

Zur erfolgreichen Nachbestellung muss die Lagerverwaltung Referenzen (*smSALs*) zu allen Zulieferlagern halten (Abb. 8.4, S. 120). Ausgangslager von Produktionssystemen können gemeinsam mit diesen aus der Produktionsanlage entfernt aber auch neu hinzugefügt werden. Über die Anwesenheitserkennung (Kap. 6.3) kann die Lagerfacette strukturelle Veränderungen der Produktionsanlage ermitteln und so die Referenzen abgleichen.

8.3 Produktionslinien und -inseln

Ein Aufbau selbstorganisierender, wandlungsfähiger Produktionslinien bzw. -inseln (= Produktionssysteme) gelingt mit Hilfe von Transfersystemen, selbstorganisierenden Stoffflüssen (Kap. 7) und generischen Lagern (Kap. 8.2). Zusätzlich sind als weitere Komponenten eine Auftragsverwaltung und Arbeitsstationen erforderlich.

Die Grenzen eines Produktionssystems bilden ein optionales Systemeingangslager SEL, ein obligatorisches Systemausgangslager SAL und optionale Eingangslager $ASEL$ als Bestandteil von Arbeitsstationen $\{AS\}\ mit\ |\{AS\}| \geq 1$ (Abb. 8.7). Über die Eingangslager werden Vorerzeugnisse, andere Stoffe und Transferhilfsmittel bereitgestellt, die von externen Transfersystemen angeliefert werden. Fertig bearbeitete (Vor-)Erzeugnisse treten auf der anderen Seite über SAL aus dem System aus. Eine Auftragsverwaltung AV verwaltet Produktionsaufträge, die ausschließlich von SAL generiert werden. Indem AV Aufträge freigibt, beginnt die Produktion durch Auslagerung eines *initialen Gegenstands*. Bei einem solchen handelt es sich immer um ein Transferhilfsmittel, das produkt-varianten-spezifisch gestaltet und mit einem oder mehreren Vorerzeugnissen beladen sein kann. Initiale Gegenstände und Transferhilfsmittel können in allen Eingangslagern gespeichert werden. Dagegen bleiben die restlichen Stoffe den Arbeitsstationseingangslagern vorbehalten.

Die Ziele der effektiven Selbstorganisation bestehen in der kontrollierten Auftragsfreigabe für die Produktion, unter Berücksichtigung verfügbarer Stoffe (Material, Hilfsstoffe und Vorerzeugnisse), sowie in der Aufrechterhaltung eines stetigen Werkstückflusses unter Einhaltung einer Obergrenze an Werkstücken im System (Steuerungsprinzip *CONWIP – Constant Work in Process* [45, 99]). Ausfälle von Systemelementen werden von der Auftragsverwaltung berücksichtigt, indem keine Aufträge eingelastet werden, die nicht aktuell produzierbar sind. Als Ausfälle gelten alle Störungen, Pausen, Wartungen, Instandsetzungsarbeiten, etc. Zudem werden Ausfälle durch die Selbstorganisation der Stoffflüsse kompensiert, sofern (Teil-)Redundanzen vorhanden sind.

Produktionssysteme können direkt durch Umbau von Zellen, Segmenten oder Arbeitsstationen gewandelt werden (Kap. 4.1). Dies hat möglicherweise Änderungen in der Menge der durchführbaren Tätigkeiten zur Folge, woraus sich zwangsläufig eine Veränderung des herstellbaren Produkt-Varianten-

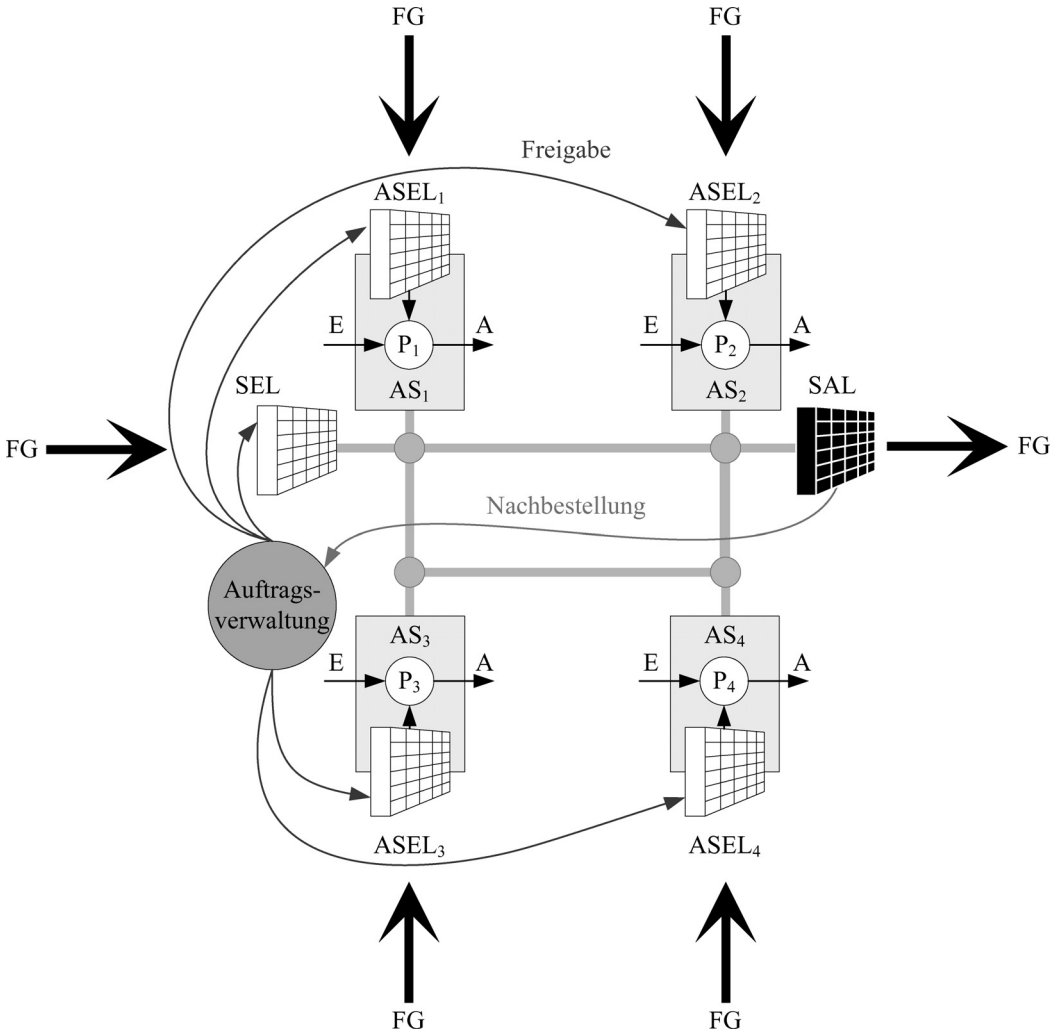


Abbildung 8.7: Generisches Modell für Produktionssysteme am Beispiel einer Produktionslinie. A = Ausgang; AS_l = Arbeitsstation; $ASEL_m$ = Arbeitsstationseingangslager; E = Eingang; FG = Flussgegentand; P_n = Prozess; SEL = Produktionssystemeingangslager; SAL = Produktionssystemausgangslager.

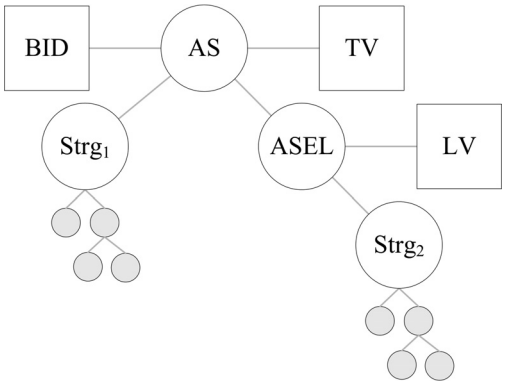


Abbildung 8.8: Grundmodell der Arbeitsstationssteuerung. AS = Arbeitsstation; $ASEL$ = Eingangslager; BID = Angebotsschnittstelle zur Auftragsbeschaffung (Kap. 7.7, S. 115); LV = Lagerverwaltung; $Strg_1$ = Ausführungssteuerung der Prozessausführung; $Strg_2$ = Ausführungssteuerung des Eingangslagers; TV = Tätigkeitsverwaltung.

```
FACET_WORKING_STATION_TASK_MANAGER = [
   Publishers = [
        TaskChanged(STRINGS taskIds)
   ]
   SetTasks(STRINGS taskIds)
   GetTasks(): STRINGS
]
```

Abbildung 8.9: Facette zur Tätigkeitsverwaltung einer Arbeitsstation.

Spektrums ergibt. Diese Veränderung wirkt sich wiederum auf benötigtes Material, Hilfsstoffe und Vorerzeugnisse aus, die in den Lagern vorrätig gehalten werden müssen. Umgekehrt können Änderungen im Produkt-Varianten-Spektrum auftreten, ohne dass ein Umbau der Produktionssysteme erfolgen muss. Allerdings kann sich das zu lagernde Stoffspektrum ändern. Das wandelbare Produktionssystem adaptiert sich automatisch an diese veränderten Situationen.

8.3.1 Arbeitsstationen

Arbeitsstationen sind die prozessausführenden Einheiten der Produktionssysteme. Es kann sich hierbei um einzelne Produktionseinrichtungen oder einen Verbund mit Zellencharakter handeln. Neben diesen ist eine Arbeitsstation AS mit einem Eingangslager $ASEL$ ausgestattet, sofern zur Durchführung von Tätigkeiten Stoffe benötigt werden.

Die Ablaufsteuerungen von Arbeitsstationen sind höchst fallspezifisch, weisen jedoch immer eine Grundstruktur auf, die für die Selbstorganisation von Bedeutung ist (Abb. 8.8). Neu ist die Schnittstellenfacette zur Tätigkeitsverwaltung TV (Abb. 8.9), in der eine Arbeitsstation immer eine Liste der verfügbaren Tätigkeiten vorhält. Diese Liste wird immer nach Wandlungsvorgängen über den Aufruf der Prozedur *SetTasks(taskIds)* angepasst und anschließend über den Publisher *TasksChanged(taskIds)* publiziert. Gegenstelle hierzu ist die Auftragsverwaltung AV, wie an nachfolgender Stelle erläutert wird.

```
FACET_ORDER_MANAGER = [
  Facets = [
      FACET_STOCK_MANAGERS smSELs
      FACET_STOCK_MANAGERS smASELs
      FACET_STOCK_MANAGER smSAL
      FACET_WORKING_PLAN_MANAGER wpm
  ]
  Publishers = [
      RequestForOrder(PARTS parts, STRING carrierTypeId, LOCATOR receiver)
  ]
  SubscriptionCallbacks = [
      RequestForOrder(PARTS parts, STRING carrierTypeId, STRING receiverId)
      TasksChanged(STRINGS taskIds)          //Identität der Arbeitsstation ist über den mitgesendeten
      ProdVarsChanged(STRINGS prodVarIds)    //Locator des Publishers ermittelbar
  ]
  ReleaseParts(PARTS parts, STRING carrierTypeId, LOCATOR receiver)
    ↳ THROWS NotInStock(PARTS parts)
  OrderedPartsDeliverableBy(LOCATOR stockManager, PARTS parts)

  SetOrderSelectFunction(FUNCTION func)
]
```

Abbildung 8.10: Facette zur Auftragsverwaltung. Die Verwaltung verhält sich wie ein fiktives Lager und implementiert dazu weitgehend die Schnittstellen der Lagerverwaltung.

8.3.2 Auftragsverwaltung

Die Auftragsverwaltung AV nimmt Produktionsaufträge entgegen und gibt sie kontrolliert, unter Berücksichtigung einer Obergrenze an Werkstücken, in das System frei. Dazu verfügt AV über einen internen WIP-Zähler (WIP – Constant Work in Process), mit dem die Anzahl der im System befindlichen Werkstücke konstant gehalten wird. Die Verwaltung ist eine einzelne Facette, die in einem Knoten gekapselt wird, und bildet mit einem eigenen Rechner eine Entität. Sie kann aber auch alternativ bei anderen Entitäten als zusätzlicher Nebenprozess ausgeführt werden.

Die Struktur der Facette ist in Abbildung 8.10 gezeigt. Sie tritt nach Außen als ein *fiktives Eingangslager* auf, welches das Ausgangslager beliefert. Dazu implementiert sie weitgehend die gleichen Lagerschnittstellen. Die Einlastung von Produktionsaufträgen erfolgt analog zu den Lagern über die Schnittstelle *ReleaseParts(parts, carrierTypeId, receiver)*. AV steht mit allen Verwaltungen LV der Eingangs- und Ausgangslager des Produktionssystems über Referenzen in Verbindung. Auftragsfreigaben werden als Nachbestellungen gehandhabt, weshalb der gleiche Publisher-Subscriber-Mechanismus verwendet wird. Publisher *RequestForOrder(parts, carrierTypeId, receiver)* und ein gleichnamiger Callback sowie die Prozedur *OrderedPartsDeliverableBy(stockManager, parts)* sind deshalb vorhanden. Zusätzlich hält AV eine Referenz auf die anlagenweite Arbeitsplanverwaltung APV, um eine geeignete Auftragsauswahl zur Freigabe durchführen zu können. Für die Adaption an Wandlungen verfügt AV über die Prozedurschnittstellen *TasksChanged(taskIds)* und *ProdVarsChanged*. Es ist möglich, AV fallspezifisch unterschiedliche Strategien zur Auswahl des nächsten Auftrags zu übergeben, wofür die Prozedur *SetOrderSelectFunction(func)* vorhanden ist.

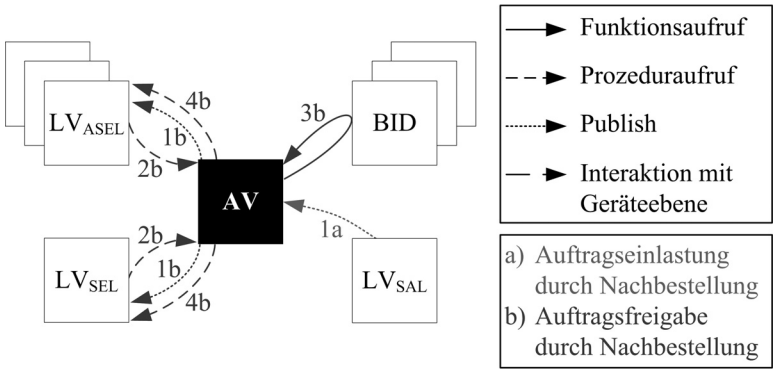


Abbildung 8.11: Produktionsliniensteuerung durch Zusammenwirken der Lager und Arbeitsstationen. Alle Facetten sind dezentralisiert in den einzelnen Steuerungen und über die Kommunikations-Middleware gekoppelt. Die invertiert dargestellte Facette verdeutlicht diese Verteilung. AV = Auftragsverwaltung; BID = Angebotsschnittstelle Arbeitsstation; LV_{ASEL} = Lagerverwaltung Arbeitsstationseingangslager; LV_{SAL} = Lagerverwaltung Produktionssystemausgangslager; LV_{SEL} = Lagerverwaltung Produktionssystemeingangslager.

8.3.3 Informationelle Verknüpfung und Steuerungsablauf

Die Ablaufsteuerung des Produktionssystems erfolgt durch ein Zusammenwirken der Auftragsverwaltung mit den Lagerverwaltungen aller Eingangslager und des Ausgangslagers (Abb. 8.11). Aufgabe ist die Erzeugung des konstanten Werkstückflusses unter Berücksichtigung der Verfügbarkeiten einerseits von Arbeitsstationen und ihrer Erreichbarkeit durch das Transportsystem, sowie andererseits von Stoffen. Der Steuerungsablauf gliedert sich dabei in verschiedene Teilabläufe.

Die Auftragseinlastung: erfolgt durch Aufruf der Schnittstelle *ReleaseParts(parts, carrierTypeId, receiver)* an AV, die den Auftrag in einer Warteschlange zwischenspeichert. Aufträge werden ausschließlich von der Lagerverwaltung des Systemausgangslagers SAL erzeugt und nur, wenn ein Erzeugnis dieses verlassen hat (Abb. 8.11, 1a).

Die Auftragsfreigabe: erfolgt, wenn ein Auftrag eingegangen ist, sich in der Warteschlange mindestens ein produzierbarer Auftrag befindet und die Grenze der maximalen Werkstücke im System nicht erreicht ist. Dazu müssen in einem *ersten Schritt* die produzierbaren Aufträge aus der Warteschlange gefunden werden, was durch eine *dreistufige Selektion* erfolgt. In der *ersten Stufe* werden zunächst diejenigen Aufträge ausgefiltert, die aktuell nicht verfügbare Tätigkeiten erfordern. Dazu hält AV zu jeder Arbeitsstation den Locator und eine Liste ihrer durchführbaren Tätigkeiten bereit. Die Vereinigungsmenge aller Listen ergibt das verfügbare Tätigkeitsspektrum. Die für ein Erzeugnis benötigten Tätigkeiten sind in dessen Arbeitsplan verzeichnet, der über die Referenz (*wpm*) zur Arbeitsplanverwaltung durch Aufruf der Funktion *GetWorkingPlan(prodVarId)* (Kap. 9.3) abgefragt werden kann. Durch Abgleich dieser Tätigkeiten mit der Vereinigungsmenge wird die Trennung vorgenommen. In der *zweiten Stufe* erfolgt ein erster Abgleich mit den Stoffen, wozu der Mechanismus für Nachbestellungen genutzt wird. Es wird angenommen, dass jeder zur Ausführung einer Tätigkeit benötigte Stoff ebenfalls in den Arbeitsplänen vermerkt ist. Indem AV für jeden der aus dem ersten Schritt verbliebenen Aufträge über den Publisher *RequestForOrder(parts, carrierTypeId, receiver)* eine Lieferanfrage publiziert, empfängt sie über die Prozedur *OrderedPartsDeliverableBy(stockManager, parts)* als Antworten von allen lieferfähigen Eingangslagern Listen vorhandener Stoffe in Art und Anzahl (Abb.

8.11, 1b bis 2b). Nach Bildung der Vereinigungsmenge aus allen Listen wird durch Abgleich festgestellt, ob *nicht alle* notwendigen Stoffe für einen Auftrag verfügbar ist. In der *dritten Stufe* muss abschließend für jede Tätigkeit überprüft werden, ob die dazu benötigten Stoffe auch wirklich vom Eingangslager der Arbeitsstation bereitgehalten werden. Da zu jeder Stoffliste auch der Locator (*stockManager*) des Eingangslagers übergeben wird, der den Locator der Arbeitsstation enthält (Kap. 4.4.1), kann dieser Abgleich problemfrei vorgenommen werden. Als *zweiter Schritt* wird nun aus den übrig gebliebenen Aufträgen ein geeigneter ausgesucht, für den im Anschluss ein passendes Eingangslager zur Auslagerung des initialen Gegenstands und eine erste Arbeitsstation gefunden werden muss. AV holt hierzu von allen Arbeitsstationen über die Funktion *GetBid(taskId)* (Abb. 7.19, S. 115) Angebote für die Durchführung des ersten Tätigkeitsschritts ein und wählt die günstigste Station aus (Kap. 7.7). Sofern dessen $ASEL$ den initialen Gegenstand lagert, erfolgt an diesem die Auslagerung, andernfalls am Systemeingangslager SEL (Abb. 8.11, 3b bis 4b). Als Empfänger wird die ermittelte Arbeitsstation angegeben.

Die Berücksichtigung: temporär nicht verfügbarer Arbeitsstationen erfolgt über die Anwesenheitserkennung, da Arbeitsstationen darüber ihren Zustand mitteilen (Kap. 7.7). Bei Erkennung eines Ausfalls markiert die Auftragsverwaltung die Tätigkeitsliste als inaktiv und berücksichtigt die Tätigkeiten solange nicht bei der Selektion von Aufträgen, bis die Arbeitsstation wieder verfügbar ist.

8.3.4 Identitäten initialer Gegenstände

Ein Problem der Auftragseinlastung und Nachbestellung besteht darin, fertige Erzeugnisse von initialen Gegenständen unterscheiden zu können. Produkte und ihre Varianten werden über die Produkt-Varianten-Kombination *prodVarId* identifiziert. Da Bestellvorgänge über den Publisher-Subscriber-Mechanismus initiiert werden, erhalten alle Eingangslager Anfragen zur Lieferbarkeit. Würden für die initialen Gegenstände die gleichen Identitäten wie für die fertigen Erzeugnisse verwendet, könnten Aufträge irrtümlich an den Eingangslagern und nicht an der Auftragsverwaltung eingelastet werden.

Durch Einführung eines *Initialpräfix*, das Produkt-Varianten-Kombinationen vorangestellt wird, kann eine Unterscheidbarkeit gewährleistet werden. Bestellungen des Ausgangslagers an der Auftragsverwaltung erfolgen weiterhin mit *prodVarId*, während die Freigabe mit *prefixProdVarId* erfolgt. Allerdings muss das Präfix im Eingangslager von der Lagerverwaltung wieder entfernt werden, so dass der Informationsspeicher des initialen Gegenstands mit *prodVarId* beschrieben wird. Dazu ist eine entsprechende Implementierung in der Prozedur *ReleaseParts(parts, carrierTypeId, receiver)* notwendig.

8.3.5 Eingriffspunkte zur effizienten Steuerung

Die Auftragsverwaltung nimmt eine zentrale Position beim Erreichen von Produktionszielen ein, da sie für die Reihenfolgebildung der Aufträge verantwortlich ist. Die Leistung eines Produktionssystems hängt demnach auch von der gewählten Auswahlstrategie ab. Wie schon bei den Strategien zu Einlager- und Auslagervorgängen sowie zum Auslösen von Bestellvorgängen bei Lagern, existiert auch in diesem Fall keine universelle, optimale Lösung. Stattdessen muss fallspezifisch eine geeignete

Regeln	Auswahl nach ...
FIFO	Ankunftsreihenfolge.
FPE	dem frühesten Plan-Endtermin.
KOZ / LOZ	kürzester / längster Gesamtbearbeitungszeit.
NAA / GAA	niedrigster / größter Anzahl durchzuführender Tätigkeiten.
MGK	meistgebundenem Kapital.
GS	aus Erfahrung größter Störanfälligkeit.
GUA	geringstem Umrüstaufwand.
SZ	Stufenzahl, die berücksichtigt, ob Aufträge in andere einmünden und ob diese wiederum in andere eingehen, usw. Je tiefer die Schachtelung, desto höher die Stufenzahl.
EPZ	Extern vorgegebener Prioritätsziffer.

Tabelle 8.1: Mögliche Regeln zur Reihenfolgebildung von Produktionsaufträgen für die Auftragsverwaltung. Regeln können miteinander kombiniert werden. Nach [24].

Strategie ausgewählt werden. Die Auftragsverwaltung verfügt hierzu über die Schnittstelle *SetOrderSelectFunction(func)*, über die ihr eine als Funktion implementierte Strategie übergeben werden kann. Diese Funktionen verfügen über eine einheitliche Schnittstelle und werden immer dann aufgerufen, wenn eine Entscheidung getroffen werden muss. Der Mechanismus ist zur Auswahl von Eingängen an Verzweigungen identisch, weshalb an dieser Stelle auf Abbildung 7.13 (S. 107) verwiesen sei. Um alle erforderlichen Informationen für eine fundierte Entscheidung treffen zu können, wird den Funktionen als Eingangsparameter eine Referenz auf die Auftragsverwaltung mit übergeben. Darüber steht diesen der vollständige Zugriff auf das gesamte Steuerungsmodell der Anlage – und damit auf das gesamte „Anlagenwissen“ – frei. Beispielhafte Auswahlregeln sind in Tabelle 8.1 dargestellt.

8.3.6 Unterstützung der Wandelbarkeit

Produktionssysteme können auf verschiedene Weise von Wandlungen betroffen sein. Es sind Modifikationen an Arbeitsstationen genauso möglich, wie strukturelle Veränderungen auf Systemebene, indem Arbeitsstationen hinzugefügt, ausgetauscht oder entfernt werden. Insgesamt können sich dadurch Veränderungen im Tätigkeitsspektrum ergeben, so dass die Menge bearbeitbarer Produkte und Varianten gewandelt wird. Umgekehrt ist dies auch von außen etwa durch die Produktionsvorbereitung möglich.

Die Aufrechterhaltung der Produktion ist unmittelbar von der korrekten Konfiguration des Ausgangslagers abhängig. Es muss jederzeit über das aktuell herzustellende Produkt-Varianten-Spektrum informiert sein, um korrekte Produktionsaufträge an der Auftragsverwaltung erzeugen zu können. Die Suche nach produzierbaren Aufträgen setzt wiederum bei der Auftragsverwaltung Wissen über die verfügbaren Tätigkeiten voraus. Damit eine Stoffverfügbarkeit gegeben ist, benötigen zudem alle Eingangslager Kenntnis über die entsprechend einzulagernden Stoffe, um ihrerseits Bestellvorgänge an externen Lagern tätigen zu können.

Zur Selbstadaption an Wandlungen sind deshalb die drei Fälle *Wandlung Arbeitsstation, Wandlung Produktionssystem* und *Wandlung Produkt-Varianten-Spektrum* zu berücksichtigen (Abb. 8.12).

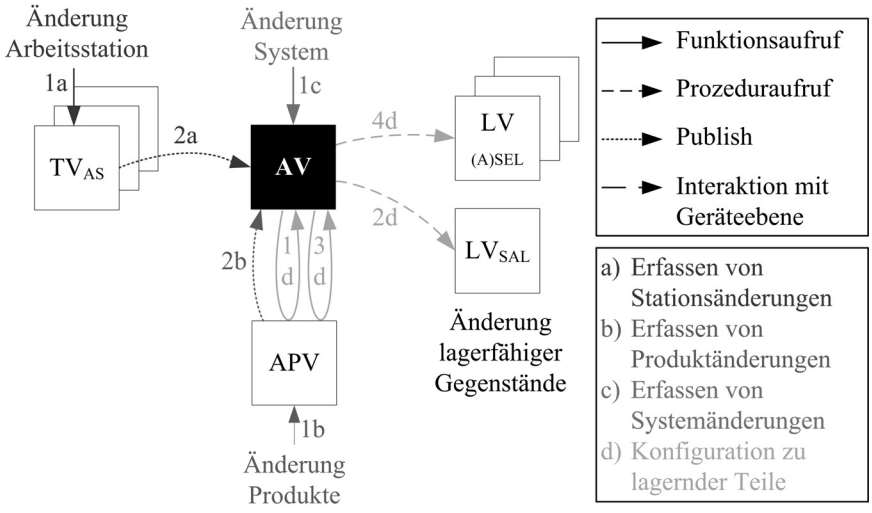


Abbildung 8.12: Selbstadaption von Produktionslinien und -inseln an Wandlungsvorgänge. Diese sind durch Änderungen in den Arbeitsstationen, der Systemstruktur oder anlagenweit durch Änderungen an Produkten verursacht. AV = Auftragsverwaltung; APV = Arbeitsplanverwaltung; $LV_{(A)SEL}$ = Lagerverwaltung Eingangslager; LV_{SAL} = Lagerverwaltung Produktionssystemausgangslager; TV_{AS} = Tätigkeitsverwaltung Arbeitsstation.

Führt die *Wandlung einer Arbeitsstation* zu einer Veränderung der durchführbaren Tätigkeiten, aktualisiert diese intern die Liste der verfügbaren Tätigkeiten über die Prozedur *SetTasks(taskIds)* der Tätigkeitenverwaltung TV. TV publiziert diese Liste anschließend mit Hilfe des Publishers *TaskChanged(taskIds)*. Die Auftragsverwaltung AV ist bei den Publishern aller Arbeitsstationen im Produktionssystem registriert und empfängt über den gleichnamigen Callback die Listen (Abb. 8.12, 1a bis 2a).

Führt die *Wandlung auf Ebene des Produktionssystems* systemweit zu einer Veränderung der durchführbaren Tätigkeiten, so stellt dies die Auftragsverwaltung AV in zwei Schritten fest. Zunächst ist AV an die Anwesenheitserkennung (Kap. 6.3) gekoppelt, so dass vor allem ausgefallene Arbeitsstationen erkannt werden. Da AV zu jeder Arbeitsstation eine Liste ihrer durchführbaren Tätigkeiten hält, wird die der ausgefallenen Einheit entfernt (Abb. 8.12, 1c).

Wird das *Produkt-Varianten-Spektrum* von außen geändert, signalisiert die Arbeitsplanverwaltung APV über den Publisher *ProdVarsChanged(prodVarIds)*, dass es Veränderungen gegeben hat. Die Auftragsverwaltung AV ist hierzu über den gleichnamigen Callback registriert und empfängt die Signale (Abb. 8.12, 1b bis 2b).

In *allen drei Fällen* erfolgt eine Neukonfiguration der Lager. Dazu beschafft die Auftragsverwaltung AV zunächst über die Funktion *GetProducables(taskIds)* an der Arbeitsplanverwaltung APV eine Liste der durch die Vereinigungsmenge der durchführbaren Tätigkeiten herstellbaren Produkt-Varianten-Kombinationen. Das Systemausgangslager SAS wird mit dieser Liste über die Prozedur *SetStorables(typeIds)* konfiguriert. Anschließend ermittelt AV die dazugehörigen Arbeitspläne über die Funktion *GetWorkingPlan(prodVarId)* und extrahiert aus diesen, unter Beibehaltung der Zuordnung zu den Tätigkeiten, alle erforderlichen Stoffe inklusive benötigter Vorerzeugnisse und initialer Gegenstände. Durch Aggregation der Stoffe in Abhängigkeit der Tätigkeiten einer jeden Arbeitssta-

tion können die Stofftypen für deren Eingangslager $ASEL$ ermittelt und über *SetStorables(typeIds)* gesetzt werden. Das Systemeingangslager SEL wird mit allen initialen Gegenständen konfiguriert, die *nicht* von den Eingangslagern gespeichert werden (Abb. 8.12, 1d bis 4d).

Alle drei Ursachen können gleichzeitig oder in kurzen Zeitabständen in Erscheinung treten. Dies stellt jedoch kein Problem dar, da sich die Vorgänge überlagern lassen.

8.4 Selbstorganisation der Produktionsanlage

Ein Aufbau selbstorganisierender, wandlungsfähiger Produktionsanlagen ist mit Produktionslinien bzw. -inseln (Kap. 8.3) sowie generischen Lagern (Kap. 8.2) möglich.

Eine Produktionsanlage besteht aus Produktionsanlageneingangslagern $\{PAEL\}\ mit\ |\{PAEL\}| \geq 1$ zur Speicherung extern zugelieferter Roh-, Kauf- und Normteile und Produktionsanlagen-Ausgangslagern $\{PAAL\}\ mit\ |\{PAAL\}| \geq 1$ zur Speicherung der fertig gestellten Produkte. Die Produktion erfolgt mit Hilfe von Produktionssystemen $\{\Sigma\}\ mit\ |\{\Sigma\}| \geq 1$, die untereinander über ein oder mehrere Transportsysteme vernetzt sind. Da Transportsysteme von Wandlung nicht direkt betroffen sind (Kap. 4.1), werden sie an dieser Stelle vernachlässigt.

8.4.1 Kopplung mehrerer Systeme

Komplexe Produkte benötigen in der Regel Vorerzeugnisse, die in gleichem oder anderen Produktionssystemen hergestellt werden. Aus diesem Grund sind diese Systeme über Transport auf Anlagenebene miteinander verbunden. Zwischen Produktionssystemen können $m : n$-Beziehungen bestehen, d.h. einem Systemeingangslager können mehrere Ausgangslager anderer Systeme vorgelagert und einem Systemausgangslager mehrere Eingangslager nachgelagert sein.

Es bestehen drei Möglichkeiten Systeme miteinander zu verbinden. Über eine sequenzielle Kopplung sind zwei Systeme Σ_1 und Σ_2 über das Ausgangs- SAL_{Σ_1} und Eingangslager SEL_{Σ_2} verknüpft (Abb. 8.13). Der Gegenstandsfluss wird durch rückwärtsgerichtete Bestellaufträge erzeugt, wobei sich systemexterne Nachbestellungen (blaue Pfeile) mit systeminternen Produktionsaufträgen (grüne Pfeile) abwechseln. Dadurch entstehen sukzessive Auslagerketten, die bis zu den Produktionsanlageneingangslagern zurückreichen. Die zweite Möglichkeit der Kopplung besteht über ein Zwischenlager (Abb. 8.14). Dieses kann von mehreren Systemen versorgt werden und mehrere Systeme beliefern. Eine Entnahme führt zu einer Nachbestellung. Die dritte Variante besteht in der Ankopplung von Systemen über die Eingangslager der Arbeitsstationen $ASEL$ (Abb. 8.15). Da die Lagersteuerung generisch ausgelegt ist, sind die Abläufe in allen Fällen identisch.

8.4.2 Ankopplung an die Produktionsplanung und -steuerung

Die Produktionsanlage ist an die Produktionsplanung und -steuerung PPS der Fabrik gebunden, wobei an dieser Stelle die Anbindung an die übergeordnete Auftragsfreigabe und Materialwirtschaft von

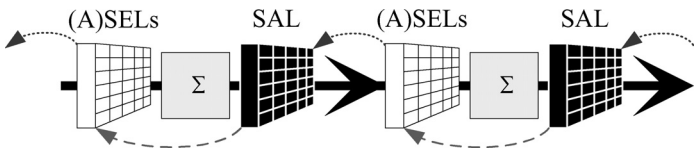


Abbildung 8.13: Sequenzielle Kopplung mehrer Linien. Die vorgelagerten Linien produzieren Vorerzeugnisse für die Nachfolger. $(A)SEL$ = Eingangslager; SAL = Produktionssystemausgangslager; Σ = Produktionssystem.

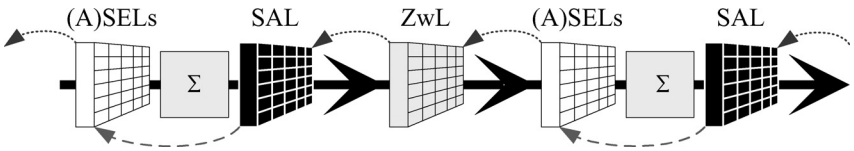


Abbildung 8.14: Kopplung zweier Linien über ein Zwischenlager. $(A)SEL$ = Eingangslager; SAL = Produktionssystemausgangslager; Σ = Produktionssystem; ZwL = Zwischenlager.

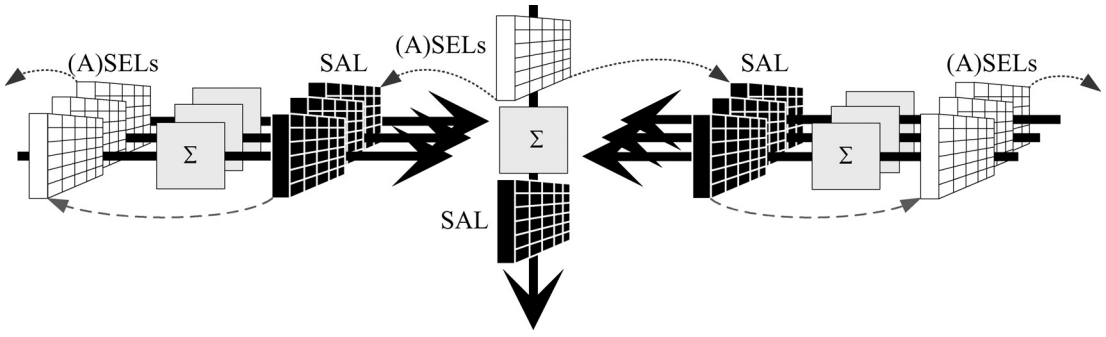


Abbildung 8.15: Kopplung mehrerer Linien zur Herstellung von Vorerzeugnissen an Eingangslagern von Arbeitsstationen einer Hauptlinie. $(A)SEL$ = Eingangslager; SAL = Produktionssystemausgangslager; Σ = Produktionssystem.

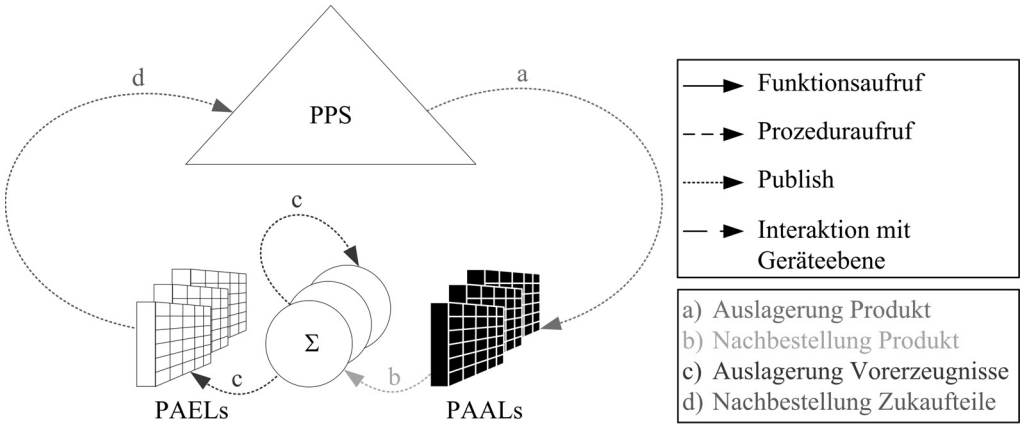


Abbildung 8.16: Verknüpfung der Produktionsanlage mit den Bereichen Auftragsfreigabe und Materialwirtschaft der übergeordneten Produktionsplanung und -steuerung (PPS). Das Resultat ist ein rückwärts gerichteter, ziehender Kreislauf. $PAEL$ = Produktionsanlagen-Eingangslager; $PAAL$ = Produktionsanlagenausgangslager; Σ = Produktionslinie bzw. -insel.

Bedeutung ist [89]. Indem diese als fiktive Lager aufgefasst und mit Facettenschnittstellen ausgestattet werden, lassen sich Auftragsfreigabe, Materialwirtschaft und Produktionsanlage zu einem einzigen rückwärts gerichteten, ziehenden Kreislauf verschalten (Abb. 8.16). Ein vom PPS freigegebener Produktionsauftrag wird durch eines der Produktionsanlagenausgangslager $PAALs$ aufgefangen und ein Produkt ausgeschleust. Durch Nachbestellungen an den Produktionssystemen werden die Lücken wieder aufgefüllt. Da Nachbestellungen von Produkten zu Nachbestellungen von Vorerzeugnissen führen, resultieren schließlich Bestellungen bei den Produktionsanlageneingangslagern $PAELs$. Indem diese nun wiederum an der Materialwirtschaft Bestellungen vornehmen schließt sich der Kreis.

8.4.3 Unterstützung der Wandelbarkeit

Auf Ebene der Produktionsanlage sind keine expliziten Mechanismen mehr zur Unterstützung der Wandelbarkeit erforderlich. Zwar können Teilsysteme und Produktionslager hinzukommen oder entfernt werden, allerdings basieren alle Kopplungen auf publizierten Ereignissen. Es gibt daher keine festen Bindungen, die explizit geschaffen werden müssten. Die Systeme sind aus Sicht der Selbstorganisation daher vollständig modular.

Kapitel 9

Selbstadaptierende generische Informationsdienste und allgemeine Softwaresysteme

Die in den vorangegangenen Kapiteln (6, 7, 8) erläuterten Ansätze zur Selbsterkennung, -konfiguration und -organisation reichen noch nicht aus, um wandelbare Produktionsanlagen erfolgreich zu betreiben. Informationssysteme sind dazu erforderlich, die wichtige Aufgaben zur Informationsverwaltung und -bereitstellung sowie zur Informationsverarbeitung übernehmen. Sie gliedern sich in *generische Informationsdienste* und *allgemeine Softwaresysteme* (Kap. 4.6).

Zu den *generischen Informationsdiensten* gehören Katalogdienst, Arbeitsplanverwaltung, Auftragsverwaltung von Produktionslinien und -inseln, Rezeptverwaltung, Modellverwaltung und Betriebsdatenverwaltung.

Zu den *allgemeinen Softwaresystemen* werden beispielhaft Informationsverarbeitungen, Steuerungen höherer Ordnung, Mensch-Maschine-Schnittstellen, Produktionsüberwachungen, Produkt-Tracking-Systeme, Qualitätssicherungssysteme oder Diagnosesysteme gezählt. Der Begriff „allgemein“ bezieht sich dabei nicht auf die spezifische Gestaltung dieser Systeme sondern soll anzeigen, dass aufgrund der hohen Vielfalt keine näheren Spezifikationen erfolgen können.

Im Folgenden werden für die erste Gruppe ausgehend von einem generischen Grundkonzept, konkretere Lösungsansätze für die einzelnen Verwaltungen benannt und für die zweite Gruppe aufgezeigt, wie eine wandelbare Gestaltung erreicht werden kann.

9.1 Generische Informationsdienste

Allen generischen Informationsdiensten ist gemein, dass sie im Zentrum eines jeweils eigenen *Informationsnetzes* stehen, in dem Entitäten oder Softwaresysteme als Informationserzeuger und -verbraucher interagieren. Informationen werden dabei zentral gebündelt. Unabhängig der Art der Informationen weisen alle Dienste deshalb einen gemeinsamen Grundaufbau auf.

Informationsnetze können in einem wandelbaren Umfeld Veränderungen unterworfen sein, wovon auch die Informationsdienste betroffen sind. Zusätzlich besteht die Gefahr eines Ausfalls der Dienste selbst, weshalb zur Absicherung die Möglichkeit einer redundanten Auslegung gegeben sein muss. Um Redundanz zu realisieren, bedarf es mehrerer Schritte. Zunächst muss die Erzeugung mehrerer Instanzen eines Diensts möglich sein. Aus diesen ist dann ein Hauptdienst (*Master*) auszuwählen, über den der Austausch zwischen Informationserzeugern und -verbrauchern erfolgt. Die restlichen Dienste agieren als Nebendienste (*Slaves*). Um Informationsverluste durch den Ausfall des Masters

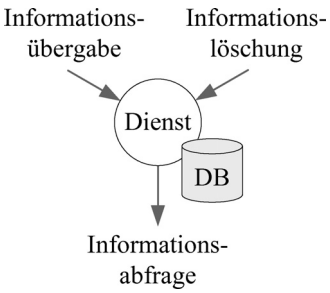


Abbildung 9.1: Dienst und seine Schnittstellen zum Informationsaustausch.

zu vermeiden ist eine Synchronisation der Informationen mit den Slaves erforderlich. Dies muss immer dann geschehen, wenn neue Informationen hinzugefügt oder bestehende verändert bzw. gelöscht werden. Fällt ein Master aus, muss einer der Nebendienste diesen ersetzen.

9.1.1 Grundaufbau der Informationsdienste

Der grundlegende Aufbau eines Informationsdiensts besteht im Kern aus einer Datenbank zur Persistierung von Informationen sowie Schnittstellen zur Informationsübergabe, -löschung und -abfrage (Abb. 9.1). Zur Interaktion mit Informationserzeugern und -verbrauchern dient eine Schnittstellenfacette. Zwar variiert diese bei den verschiedenen Diensten, ihr struktureller Aufbau ist jedoch gleich (Abb. 9.2). Die Informationsübergabe erfolgt über die Prozedur *AddInfo(id, ...)*, die Löschung über *DelInfo(id, ...)* und die Abfrage über die Funktion *GetInfo(...)*.

Informationen werden in der Datenbank unter einer eindeutigen Identität gespeichert. Diese Identität wird beim Hinzufügen von Informationen als Rückgabewert von *AddInfo(id, ...)* zurück geliefert und kann zur gezielten Aktualisierung oder Löschung von Informationen als Parameter übergeben werden. Zudem ist die Identität für die Synchronisation zwischen Haupt- und Nebendiensten von Bedeutung, die über die Publisher *InfoAdded(id, seqNo, info), InfoDeleted(id, seqNo)* und die gleichnamigen Callbacks erfolgt.

9.1.2 Redundanz zur Absicherung von Ausfällen

Die Absicherung von Ausfällen wird durch redundante Dienste erreicht, indem ein spezifischer Dienst mehrfach in eine Produktionsanlage integriert wird. Hierbei ist eine Anforderung, dass jedem Dienst ein eigener Rechner zugeordnet ist, da Rechnerstörungen eine Ursache für Ausfälle von Diensten sein können. Dienst und Rechner bilden dadurch eine eigene Entität. Damit Informationserzeuger und -verbraucher, also andere Entitäten, mit einer dedizierten Informationsverwaltung interagieren können, muss der Informationsaustausch über einen Hauptdienst (*Master*) gebündelt werden. Dazu ist ein Verfahren erforderlich, das den Entitäten dezentral und deterministisch die Auswahl des richtigen Masters ermöglicht. Um dies zu können, müssen Dienste als solche eindeutig erkennbar und verschiedene Dienstarten (z.B. Arbeitsplanverwaltung, Rezepteverwaltung) unterscheidbar sein. Ausfälle von Hauptdiensten bergen die Gefahr von Informationsverlusten. Diese müssen möglichst ver-

```
FACET_SERVICE = [
  Publishers = [
      InfoAdded(STRING id, INT seqNo, STRING info)
      InfoDeleted(STRING id, INT seqNo)
      Resync(STRINGS ids)
  ]
  SubscriptionCallbacks = [
      InfoAdded(STRING id, INT seqNo, STRING info)
      InfoDeleted(STRING id, INT seqNo)
      Resync(STRINGS ids)
  ]
  AddInfo(STRING id, ...): STRING THROWS SameAlreadyExistent
  DelInfo(STRING id, ...) THROWS Unknown
  GetInfo(...): ... THROWS Unknown
]
```

Abbildung 9.2: Allgemeines Facettenschema für Dienste. Je nach Dienst variieren die Informationsschnittstellen im Detail.

mieden werden, weshalb eine Synchronisation der Informationen mit den Nebendiensten erforderlich ist.

Auffinden redundanter Einheiten

Zum Auffinden der Dienste kann die, in Kapitel 6.3 vorgestellte, Anwesenheitserkennung genutzt werden. Da die Datenstruktur der Nachrichten zur Selbstorganisation von Stoffflüssen bereits um ein Typfeld (*Type*) erweitert wurde (Abb. 6.4, S. 79), kann dieses zur Erkennung von Diensten und zur Unterscheidung von Dienstarten genutzt werden.

Nicht alle Dienste gleichen Typs gehören zusammen. Beispielsweise verfügt jede Produktionslinie über eine eigene Auftragsverwaltung – die Verwechslung von Auftragsverwaltungen verschiedener Produktionslinien sollte vermieden werden. Deshalb müssen Gruppierungen von Diensten erkannt und auseinander gehalten werden können. Durch Berücksichtigung der hierarchischen Anlagenstruktur (Kap. 5.7), die selbstständig ermittelt werden kann (Kap. 6.6), ist dies möglich, da Anlagenbereiche in einzelne Zonen segmentiert werden (Abb. 5.24, S. 74). Dienste eines Typs, die sich in einer Zone befinden, bilden eine Gruppe.

Verfahren zur Wahl der Masterapplikation

Ein einfaches, dezentrales und deterministisches Verfahren zur Auswahl des Masters beruht auf der Tatsache, dass zwischen den Identitäten der Dienste (= id des Wurzelknotens des Steuerungsmodells, Kap. 5.3.1) eine Ordnungsrelation besteht, so dass sich diese aufsteigend sortieren lassen. Aus der Liste der sortierten Identitäten wird der Dienst an erster Stelle und damit mit niedrigster Identität zum Master ernannt (Abb. 9.3). Abbildung 9.4 zeigt den dazugehörigen Algorithmus, der, als Funktion implementiert, von jeder Entität aufgerufen wird, wenn die Strukturerkennung (Kap. 6.4) Veränderungen in der eigenen Zone feststellt. Dadurch erkennt jeder Dienst, ob er Master oder Slave ist, und jeder Informationserzeuger und Verbraucher, an welchen Dienst er sich wenden muss.

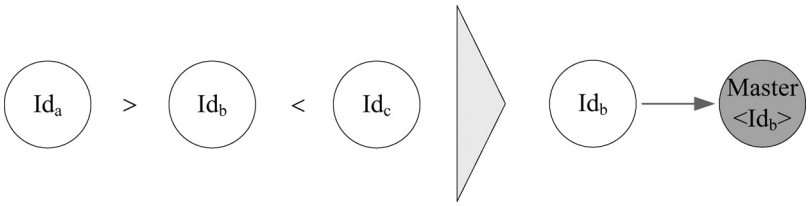


Abbildung 9.3: Auswahl des Master auf Grundlage der Ordnungsrelation zwischen den Identitäten der Dienste.

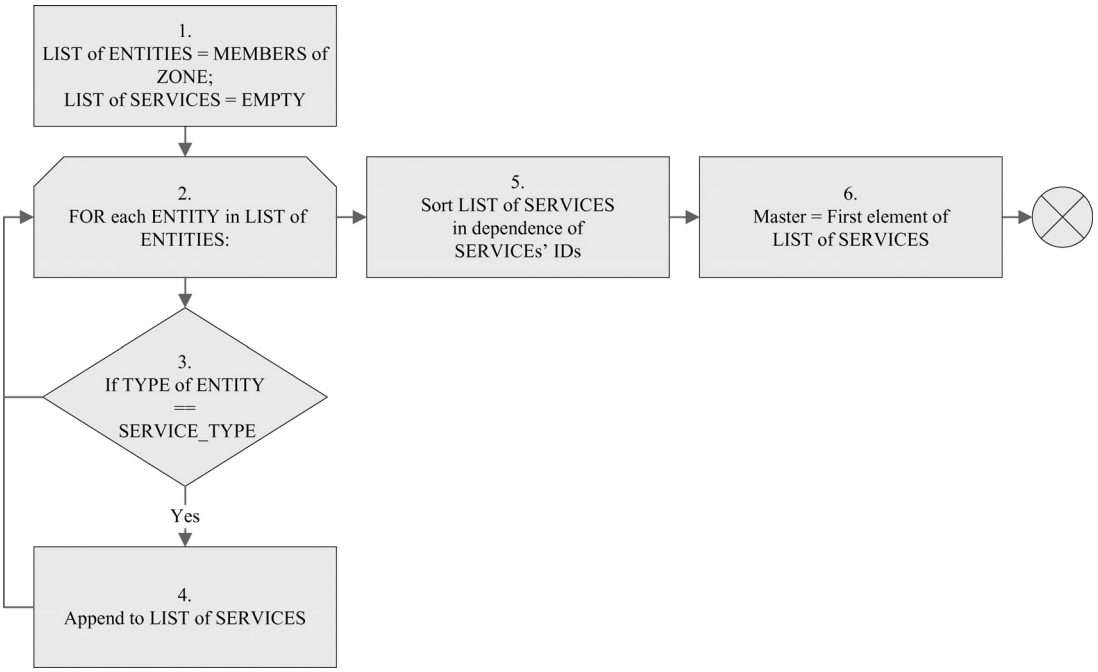


Abbildung 9.4: Funktion zur Auswahl des Masters einer Zone.

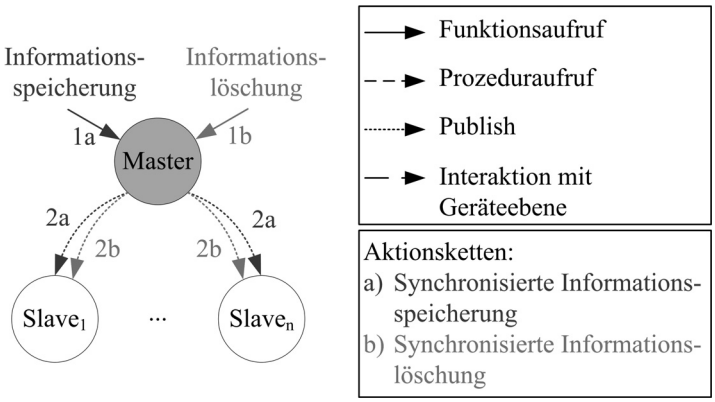


Abbildung 9.5: Synchronisationen von Informationsänderungen zwischen Master und redundanten Slaves.

Synchronisation von Informationen

Für die Synchronisation von Informationen zwischen Master und Slaves sind die Publisher *InfoAdded(id, info) und InfoDeleted(id)* sowie die gleichnamigen Callbacks der Schnittstellenfacette (9.2, S. 136) vorgesehen.

Die informationelle Verknüpfung der Dienste und der Synchronisationsablauf sind wie folgt. Wird dem Master eine neue Information hinzugefügt, speichert er sie zusammen mit einer Identität (*Id*) und einem Sequenznummernzähler in seiner Datenbank. Im Anschluss verteilt er die neue Information mit der Identität und dem Zählerstand (*seqNo*) des Sequenznummernzählers über den Publisher *InfoAdded(id, seqNo, info)*. Entsprechend verhält es sich, wenn eine Information aus der Datenbank gelöscht wird. In diesem Fall wird über *infoDeleted(id, seqNo)* publiziert. Alle Dienste einer Gruppe sind untereinander für diese Ereignisse über die passenden Callbacks registriert und werden daher entsprechend informiert, woraufhin sie ihre Datenbestände anpassen (Abb. 9.5).

Die Übermittlung der Sequenznummer ist erforderlich, da die Reihenfolge empfangener Ereignisse nicht festgelegt ist und so Vertauschungen entstehen können. Geschieht dies bei zwei Ereignissen, die gleiche Information betreffend, kann dadurch eine Verfälschung auftreten. Mit Hilfe der Sequenznummern ist ein solcher Vorgang ausgleichbar, da darüber Reihenfolgefehler erkannt und Ereignisse solange von der Bearbeitung zurück gestellt werden können, bis alle vorausgegangenen Ereignisse eingetroffen sind.

9.1.3 Partitionierung und Verschmelzung von Dienstgruppen durch Wandlung

Gruppen von Diensten können von Wandlung betroffen sein, indem sie in Teilgruppen *partitionieren* oder zu neuen Gruppen *verschmelzen*. Im Weiteren wird unter Verschmelzung auch das Hinzufügen eines einzelnen neuen Diensts zu einer Gruppe verstanden.

Ursache sind in letzter Konsequenz Veränderungen im Kommunikationsnetz, über das die Anlagenstruktur und -hierarchie ermittelt wird. Indem Entitäten an neuralgischen Punkten ausfallen oder entfernt und dadurch Kommunikationsverbindungen unterbrochen werden, entstehen getrennte Kommunikationsnetze und damit eine *Partitionierung*. Umgekehrt resultiert eine *Verschmelzung* aus dem Ver-

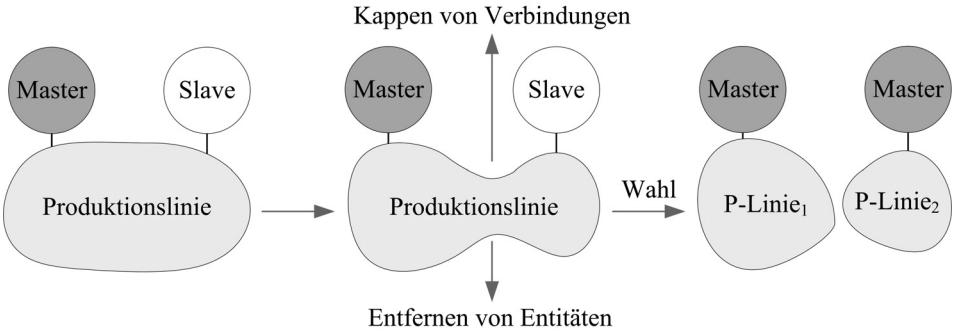


Abbildung 9.6: Beispielhafte einfache Linienpartitionierung.

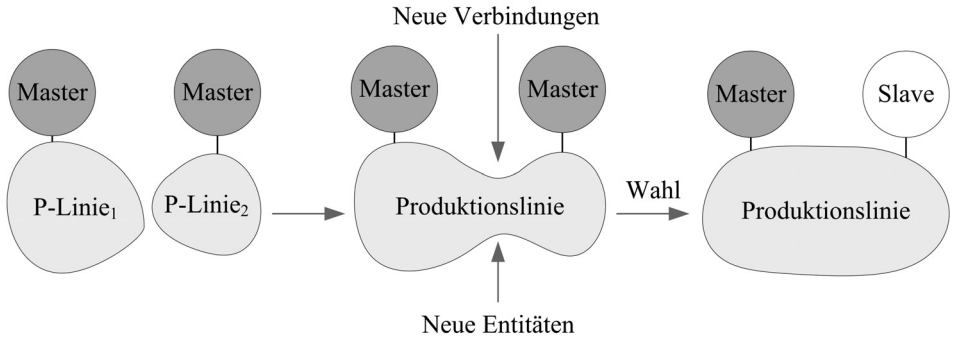


Abbildung 9.7: Beispielhaftes Verschmelzen zweier Linien.

binden von Kommunikationsnetzen, indem Entitäten instand gesetzt werden oder neu hinzukommen. Abbildungen 9.6 und 9.7 zeigen diese Vorgänge am Beispiel einer Produktionslinie.

In diesen Fällen entsteht nicht nur der Bedarf an der Bestimmung eines neuen Masters, sondern bei einer *Verschmelzung* auch die Notwendigkeit eines Abgleichs der in den verschiedenen Diensten gespeicherten Informationen. Dies ist mit dem bereits vorgestellten Synchronisationsmechanismus nicht zu bewerkstelligen. Deshalb wird ein weiterer Mechanismus eingeführt und ein weiterer Publisher *Resync(ids)* mit gleichnamigem Callback vorgesehen.

Sobald eine Zustandsänderung der eigenen Zone über die Strukturerkennung (Kap. 6.4) erkannt und infolgedessen eine Verschmelzung ermittelt wird, erfolgt eine Resynchronisation der Informationen. Hierzu sind alle Dienste gegenseitig über den entsprechenden Callback an den Publishern *Resync(ids)* registriert. Über diese publiziert jeder Dienst eine Liste der Identitäten seiner gespeicherten Informationen. Jeder Empfänger kann darüber seine Informationsdefizite durch einfachen Abgleich ermitteln und diese iterativ über die Standardschnittstelle zur Informationsabfrage *GetInfo(...)* ausgleichen. Abbildung 9.8 zeigt das Verfahren am Beispiel von vier Diensten.

Zwei Probleme können bei der Zustandsänderung auftreten. Tatsächlich erfolgt neben der Strukturerkennung auch eine Umkonfiguration des Kommunikationsnetzes, weshalb Entitäten für einen kurzen Moment gegenseitig nicht erreichbar sind (Kap. 6.7). Ein in dieser Zeit versendetes Ereignis würde unter Umständen nicht zugestellt und dadurch ein Dienst nicht informiert werden, so dass dieser nicht zur Resynchronisierung veranlasst würde. Als Ergebnis würden die Datenbestände der einzelnen Dienste differieren. Dieses Problem wird durch die Kommunikations-Middleware (Kap. 4.4.3)

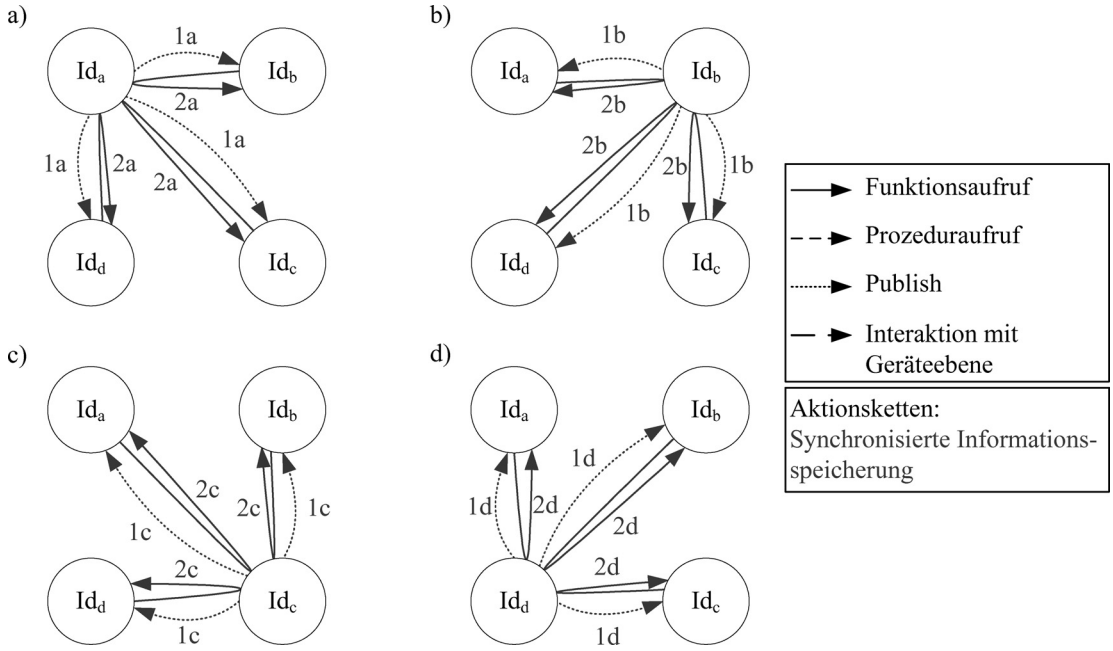


Abbildung 9.8: Resynchronisation bei neuen Diensten oder bei Verschmelzen mehrerer Linien. Zur Wahrung der Übersichtlichkeit sind die gleichzeitigen Resynchronisationsvorgänge getrennt dargestellt.

zuverlässig vermieden, da sie die Übertragung des Ereignisses ggf. mehrfach wiederholt. Das zweite Problem betrifft Informationsänderungen, die *während* der Resynchronisierung stattfinden. Sie werden von dem Mechanismus nicht erkannt. Hier greift allerdings der normale Synchronisationsmechanismus und übernimmt die Benachrichtigung der anderen Dienste.

9.2 Katalogdienst

Sowohl bei Softwaresystemen als auch bei Entitäten können dynamisch, zur Entwicklungszeit nicht vorhersagbare, Interaktionsbedarfe mit anderen Softwaresystemen oder Entitäten – genauer mit Knoten, Facetten und Publishern – entstehen. Ein Beispiel hierfür ist eine Datenerfassungskomponente, die sich an allen spezifischen Publishern bestimmter Facettentypen registrieren soll, um über Ereignisse informiert zu werden. Dazu ist ein System erforderlich, das umfangreiche Informationen über das Steuerungsmodell des Gesamtsystems bereitstellt, so dass gezielt nach Elementen gesucht werden kann, um mit ihnen dann in Kontakt treten zu können. Die Lösung besteht darin, in jeder Zone, die sich in den Hauptebenen zwei und drei des 6-Ebenen$^+$-Modells (Kap. 4.1) befindet, also in allen Produktionslinien und -inseln, Zellen und Segmenten sowie Transportsystemen, einen Katalogdienst – optional in redundanter Ausführung – einzusetzen.

Informationsspeicherung und Aktualisierung

Jeder Katalog ist an die Strukturerkennung gekoppelt, so dass neue und entfernte bzw. ausgefallene Entitäten erkannt werden. Der Dienst erfragt von jeder neu erkannten Entität den vollständigen hier-

```
FACET_CATALOG = [
   Publishers = [
        EntityAdded(STRING id,  INT seqNo, NODE_INFO info)
        EntityDeleted(STRING id, INT seqNo)
        Resync(STRINGS ids)
   ]
   SubscriptionCallbacks = [
        EntityAdded(STRING id, INT seqNo, STRING info)
        EntityDeleted(STRING id, INT seqNo)
        Resync(STRINGS ids)
   ]
   AddEntity(STRING id, NODE_INFO info): STRING THROWS SameAlreadyExistent
   DelEntity(STRING id, STRING entityId) THROWS Unknown

   GetTreesBy(STRINGS locators, STRINGS names,
                BOOL withFacets, BOOL withPublishers): NODE_INFOS
   GetNodesBy(STRINGS locators, STRING names,
                 STRINGS nodeIds, BOOL withPublishers): NODE_INFOS
   GetFacetsBy(STRINGS locators, STRINGS names,
                STRINGS facetIds, STRINGS types): FACET_INFOS
   GetPublishersBy(STRINGS locators, STRINGS names,
                    STRING types): PUBLISHER_INFOS
]
```

Abbildung 9.9: Facette mit Schnittstellen zur Katalogverwaltung.

archischen Aufbau über die Funktion *GetSubTreeInfo()* des Wurzelknotens (Abb. 5.8, S. 59). Jede, der als Baum strukturierten Information wird, unter Erhalt der strukturellen Zusammenhänge, in einzelne Informationsblöcke für Knoten, Facetten und Publisher zerlegt und in der internen Datenbank gespeichert. Sind Entitäten nicht mehr existent, löscht der Katalogdienst die entsprechenden Datenbankeinträge. Ändern sich Entitäten während des Betriebs strukturell in ihrem Aufbau, versagt dieser Mechanismus. In diesem Fall muss die Entität die bestehenden Informationen über die Schnittstellenfacette des Katalogdiensts (Abb. 9.9) durch Aufruf der Funktion *AddEntity(id, info)* überschreiben.

Schnittstellen zur Informationsabfrage

Interessenten stehen zwei Wege zur Informationsbeschaffung offen. Der Erste führt über Schnittstellen, mit denen Informationen zu Bäumen, Knoten, Facetten oder Publishern abgefragt werden können (Abb. 9.9). Diese sind *GetTreesBy(locators, names, withFacets, withPublishers), getNodesBy(locators, names, nodeIds, withPublishers), getFacetsBy(locators, names, facetIds, types)* und *GetPublishersBy(locators, names, types)*. *Locators* benennt hierbei entweder eine Liste von Elementen, über die Informationen bezogen werden sollen, oder Wurzeln von (Sub-)Bäumen, wenn zusätzlich *Namen, Typen* oder *Knotenidentitäten* angegeben werden. In letzterem Fall wird in allen Subbäumen nach passenden Elementen gesucht. Durch Setzen von *withFacets* oder *withPublishers* kann die rückzuliefernde Informationsmenge eingeschränkt werden. Über diese Schnittstellen sind somit Informationen zu einzelnen Elementen oder Elementmengen abfragbar.

Als zweiter Weg steht Interessenten die Möglichkeit offen, sich Veränderungen mitteilen zu lassen,

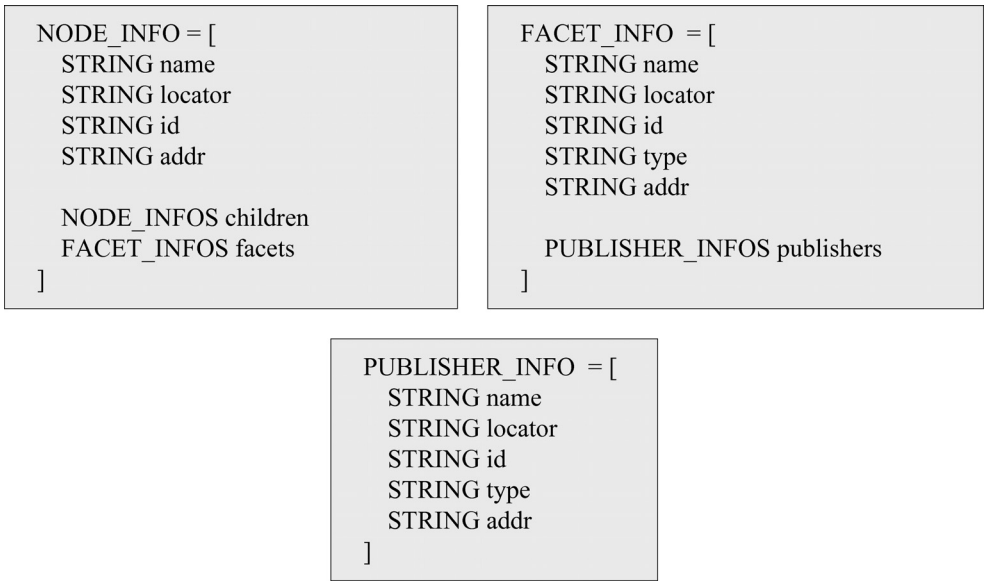


Abbildung 9.10: Datenstrukturen der verschiedenen Informationsobjekte.

indem sie sich an den Publishern *EntityAdded(id, seqNo, info)* und *EntityDeleted(id, seqNo, info)* registrieren. Hierbei handelt es sich um die regulären Publisher zur Synchronisation von Master und Slaves (Kap. 9.1.2). Mit diesem Mechanismus können auch über die Grenzen der Produktionsanlage hinaus strukturelle Veränderungen von Entitäten und Steuerungsmodellen registriert werden.

Die Datenstrukturen der abfragbaren Informationen zu Knoten, Facetten und Publishern zeigt Abbildung 9.10. Alle drei umfassen den Namen (*name*), Locator (*locator*) und die Identität (*id*) sowie bei Facetten und Publishern deren Typ (*type*). In Informationen zu einem Knoten können zudem weitere Informationen zu dessen Facetten (*facets*) und dessen Kindknoten (*children*) geschachtelt werden. Aus Letzterem folgt eine Rekursion, so dass sich eine Baumstruktur isomorph zur tatsächlichen Struktur ergibt. In Facetteninformationen können Informationen zu Publishern (*publishers*) geschachtelt werden.

Referenzierung des Katalogdiensts

Um an einen Katalogdienst überhaupt Anfragen stellen zu können, müssen seine Schnittstellen erreichbar sein, d.h. es muss eine Referenz zu diesem beschafft werden können. Dies setzt jedoch voraus, dass jeder Katalog in jeder Zone am gleichen Ort ausgeführt wird, so dass sich ein festes Schema ergibt. Dazu wird festgelegt, dass ein Master-Katalog immer in einem bestimmten Knoten mit fixem Namen gekapselt wird, der direkt dem Superseparator einer Zone untergeordnet ist (Abb. 9.11). So ist durch Anwendung des Adressierungsschemas (Kap. 4.4.1) ein Katalog erreichbar, da über die Kommunikations-Middleware (Kap. 4.4.3) nun die Referenz beschafft werden kann.

Ein einziges Problem hierbei tritt auf, wenn ein Slave-Katalog neuer Master wird, da dessen Knoten einen anderen Namen trägt. Die Lösung besteht hierbei in der automatischen Umbenennung des Knotens, so dass das Adressierungsschema wieder greift.

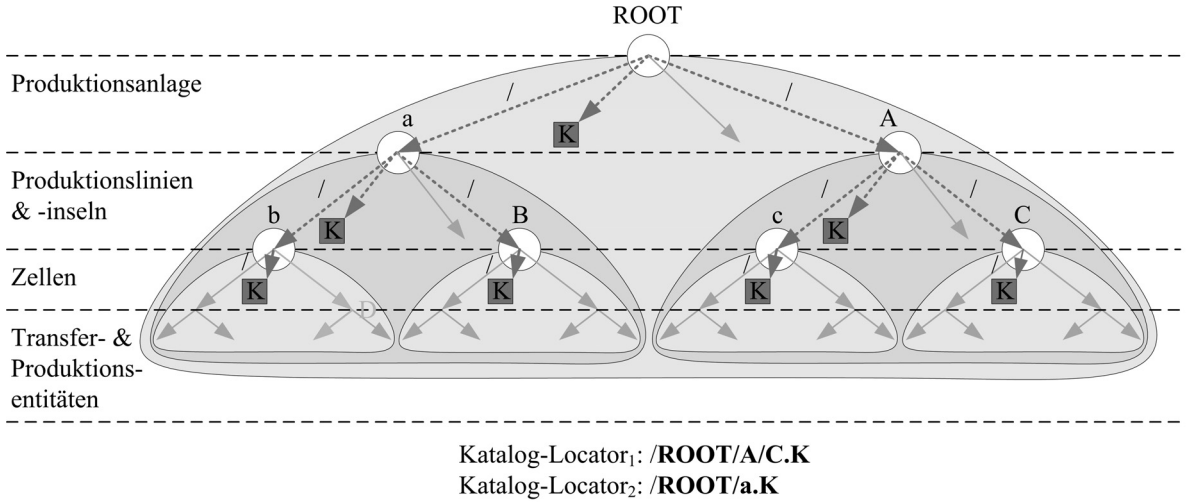


Abbildung 9.11: Anordnung der Kataloge in den hierarchischen Zonen. Durch die feste Position relativ zur Zone ergibt sich ein Schema zur Adressierung der Facettenschnittstellen. So ist stets die Beschaffung von Referenzen und damit das Stellen von Anfragen möglich. *K* = Katalog; *Kreise* = Separatoren.

9.3 Arbeitsplanverwaltung

Aufgabe der Arbeitsplanverwaltung ist die Bereitstellung von Arbeitsplänen. Sie nimmt eine zentrale Position für die Selbstorganisation von Stoffflüssen sowie Produktionslinien und -inseln ein (Kap. 7.7, 8.4).

Arbeitspläne stammen aus Bereichen, die der Produktion vorgelagert sind und werden nur aus diesen hinzugefügt, modifiziert oder gelöscht. Hierzu stehen die Prozeduren *AddWorkingPlan(id, workingPlan)* und *DelWorkingPlan(id, workingPlanId)* über eine Schnittstellenfacette bereit (Abb. 9.14). Über den Publisher *ProdVarsChanged(prodVarIds)* signalisiert die Arbeitsplanverwaltung Interessenten wenn Arbeitspläne von Veränderungen betroffen sind. Zu Veränderungen zählen Modifikationen, das Entfernen nicht mehr benötigter oder das Hinzufügen neuer Arbeitspläne.

Neben der Abfrage eines Arbeitsplans über die Funktion *GetWorkingPlan(prodVarId)*, unter Angabe seiner Identität, existieren zwei Weitere *GetNextTaskId(prodVarId, index)* und *GetProducables(taskIds)*, die primär der Selbstorganisation von Produktionslinien und -inseln (Kap. 8.3) dienen. Die erste Funktion erlaubt die Abfrage des nächsten Arbeitsschritts für ein Werkstück unter Angabe seiner Produkt-Varianten-Kombination (siehe auch Kap. 7.2) und eines Fortschrittszählers. Die zweite Funktion ermittelt unter Angabe einer Liste von Tätigkeitsidentitäten damit bearbeitbare Arbeitspläne und gibt wiederum deren Identitäten in einer Liste zurück.

9.4 Rezepteverwaltung

Die Rezepteverwaltung existiert nur einmal und wird auf der Hauptebene der Produktionsanlagen ausgeführt. Ihre Aufgabe ist es, Rezepte (= Skripte = Beschreibungen von Tätigkeiten) zu speichern.

```
FACET_WORKING_PLAN_MANAGER = [
  Publishers = [
      InfoAdded(STRING id, INT seqNo, STRING info)
      InfoDeleted(STRING id, INT seqNo)
      Resync(STRINGS ids)
      ProdVarsChanged(STRINGS prodVarIds)
  ]
  SubscriptionCallbacks = [
      InfoAdded(STRING id, INT seqNo, STRING info)
      InfoDeleted(STRING id, INT seqNo)
      Resync(STRINGS ids)
  ]
  AddWorkingPlan(STRING id, STRING workingPlan): STRING THROWS SameAlreadyExistent
  DelWorkingPlan(STRING id, STRING workingPlanId) THROWS Unknown

  GetWorkingPlan(STRING prodVarId): STRING THROWS Unknown
  GetNextTaskId(STRING prodVarId, INT index): STRING THROWS Unknown
  GetProducables(STRINGS taskIds): STRINGS THROWS Unknown
]
```

Abbildung 9.12: Facette zur Verwaltung von Arbeitsplänen.

Rezepte stammen aus der Produktion vorgelagerten Bereichen und werden über die Prozedur *AddScript(id, script)* hinzugefügt oder verändert und über die Prozedur *DelScript(id, taskId)* gelöscht. Beide Prozeduren sind Bestandteil einer Schnittstellenfacette (9.13).

Rezepte werden von skriptfähigen Controllern (Kap. 5.6.2) ausgeführt. Diese beziehen die Skripte immer dann, wenn eine Tätigkeit das erste Mal auszuüben ist. Hierzu existiert die Funktion *GetScripts(taskIds)*, die zu einer Liste von Tätigkeitsidentitäten (siehe auch Kap. 7.2) die passenden Skripte zurück liefert. Um aktualisierte oder neue Rezepte zu erhalten, registrieren sich alle Controller am Publisher *ScriptChanged(taskId)* der Rezepteverwaltung. Hierbei handelt es sich um einen extra Publisher. Es werden also nicht die Publisher zur Synchronisierung der Slaves (Kap. 9.1.2) verwendet. Bei Veränderungen eines Skripts wird ein Ereignis von der Rezeptverwaltung erzeugt. Anhand der mitgeschickten *taskId* kann jeder Controller die Relevanz der Information überprüfen und ggf. das neue Rezept nachladen.

9.5 Modellverwaltung

Die Modellverwaltung dient zur Speicherung von Steuerungsmodellen und ist auf der Ebene der Produktionsanlagen angesiedelt. Jede Entität, die ein Verbund (Kap. 3.5, 4.1) darstellt und Zwillings-(Basis-)elemente (Kap. 3.5) besitzt, kann einer Wandlung unterzogen werden (Kap. 4.5).

Hierbei sind Modelle und Modellstrukturen zu unterscheiden. Erstere sind Modellbasis-Elemente, Zweitere die Beschreibung ihrer entitätenspezifischen Verknüpfungen (Kap. 4.5). Beide sind von der Modellverwaltung zu speichern und bei Bedarf zur Verfügung zu stellen.

Die Informationen stammen aus der Produktion vorgelagerten Bereichen, etwa der Produktionsvorbereitung. Strukturen werden über die Prozedur *AddStructure(id, structure)* hinzugefügt oder verändert

```
FACET_SCRIPT_MANAGEMENT = [
  Publishers = [
      ScriptAdded(STRING id, INT seqNo, STRING script)
      ScriptDeleted(STRING id, INT seqNo)
      Resync(STRINGS ids)
      ScriptChanged(STRING taskId)
  ]
  SubscriptionCallbacks = [
      ScriptAdded(STRING id, INT seqNo, STRING info)
      ScriptDeleted(STRING id, INT seqNo)
      Resync(STRINGS infoIds)
  ]
  AddScript(STRING id, STRING script): STRING THROWS SameAlreadyExistent
  DelScript(STRING id, STRING taskId) THROWS Unknown

  GetScripts(STRINGS taskIds): STRINGS
]
```

Abbildung 9.13: Facette zur Skriptverwaltung.

und über die Prozedur *DelStructure(id, structureId)* gelöscht. Beide sind Bestandteil einer Schnittstellenfacette (Abb. 9.14). Analog folgt dies für Modelle über die Schnittstellen *AddModel(force, model)* bzw. *DelModel(modelId)*.

Im Gegensatz zu den anderen vorgestellten Diensten ergeben Benachrichtigungen von Struktur- und Modelländerungen keinen Sinn, da mehrere Entitäten gleichen Typs existieren können, jedoch nicht alle Entitäten gleichzeitig der gleichen Wandlung unterworfen sein müssen. Deshalb ist ein anderer Mechanismus notwendig, bei dem Entitäten nach Umbau initiativ werden und Struktur sowie benötigte Steuerungsmodelle bei der Verwaltung über die Abfrageschnittstellen *GetStructures(structureIds)* und *GetModels(modelIds)* beschaffen. Wie in Kapitel 5.7 erläutert, wird entweder eine automatische Strukturerkennung (Kap. 6.4) postuliert bzw. das explizite Setzen eines neuen Entitätentyps über die Schnittstelle *SetType(type)* des Wurzelknotens (Abb. 5.8) angenommen. Im ersten Fall sind nur noch automatisiert die passenden Modelle zu beschaffen, im zweiten Fall der Neuaufbau der Strukturen explizit von einer Person zu veranlassen.

9.6 Betriebsdatenverwaltung

Eine generische Betriebsdatenverwaltung, die detaillierte Schnittstellen zu Informationsabfrage bietet, ist – insbesondere in Anbetracht branchen-, wenn nicht fallspezifischer Auswertungen von Betriebsdaten zum Produkt-Tracking oder zur Qualitätssicherung – eine hochkomplexe Aufgabe und kann daher an dieser Stelle nicht ausführlich behandelt werden. Stattdessen wird ein generischer Ansatz vorgestellt, der auf einer Datenbank mit SQL-Schnittstelle (*Structured Query Language*, [4]) aufsetzt. Übergaben, Abfragen und Löschen von Informationen erfolgen als SQL-Anweisungen, mit denen eine direkte Manipulation der Datenbank bewirkt wird.

Der Vorteil dieser Herangehensweise besteht in der Generizität, da die Implementierung fallspezifisch nicht angepasst werden muss (allerdings sind Tabellen fallabhängig zu gestalten). Jedoch bestehen

```
FACET_MODEL_MANAGEMENT = [
  Publishers = [
      StructureAdded(STRING id, INT seqNo, STRING structure)
      StructureDeleted(STRING id, INT seqNo)
      ModelAdded(STRING id, INT seqNo, STRING model)
      ModelDeleted(STRING id, INT seqNo)
      Resync(STRINGS ids)
  ]
  SubscriptionCallbacks = [
      StructureAdded(STRING id, INT seqNo, STRING structure)
      StructureDeleted(STRING id, INT seqNo)
      ModelAdded(STRING id, INT seqNo, STRING model)
      ModelDeleted(STRING id, INT seqNo)
      Resync(STRINGS ids)
  ]
  AddStructure(STRING id, STRING structure): STRING THROWS SameAlreadyExistent
  DelStructure(STRING id, STRING structureId) THROWS Unknown
  AddModel(STRING id, STRING model): STRING THROWS SameAlreadyExistent
  DelModel(STRING id, STRING modelId) THROWS Unknown

  GetStructures(STRINGS structureIds): STRINGS
  GetModels(STRINGS modelIds): STRINGS
]
```

Abbildung 9.14: Facette zur Verwaltung von Steuerungsstrukturen und -modellen.

auch deutliche Nachteile. Da Datenbestände einer direkten Beeinflussung ausgesetzt sind, besteht eine Gefahr für die Datensicherheit. Falsche SQL-Anweisungen können Daten invalidieren oder löschen. Komplexe SQL-Anweisungen, die auf mehreren Datensätze operieren und diese verändern, sind schwer (re)synchronisierbar und bergen ebenfalls die Gefahr von Datenverlusten. Zudem müssen neben den eigentlichen Daten auch die original SQL-Anweisungen gespeichert werden, da sie für die (Re)Synchronisation von wesentlicher Bedeutung sind. Ohne sie ginge die Information verloren, was in einem Schritt genau mit der Datenbank „zu tun“ ist.

Nachfolgende Vereinbarungen und Regeln sind erforderlich, um die genannten Nachteile abzuschwächen. Zunächst muss gelten, dass kein Datensatz gelöscht werden darf, was in Anbetracht der Speicherung von Historien im Sinne der Betriebsdatenerfassung liegt. Lediglich eine übergeordnete Instanz ist dazu berechtigt, wobei der Löschvorgang immer am Master erfolgt. Durch (Re)Synchronisation werden Slaves aktualisiert. Dies unterstützt die Erhaltung der Konsistenz redundanter Systeme. Als weitere Regel gilt, dass nur einzelne Informationen eingefügt werden dürfen, die jeweils in einem eigenen Datensatz abzuspeichern sind. Bestehende Datensätze sind nachträglich – außer von einer übergeordneten Instanz – nicht mehr zu verändern. Auch dies trägt zur Konsistenzerhaltung bei, da andernfalls durch Resynchronisation Daten, die durch Partitionierung (nur) an einem Master zwischenzeitlich verändert wurden, überschrieben werden könnten.

Jede Facette im System kann mit der Verwaltung Kontakt aufnehmen und in SQL-Statements verpackte Informationen übermitteln, die von anderen Facetten abgefragt werden können. Hierzu stehen die Schnittstellen *AddPD(sqlStatement)* und *GetPDs(sqlStatement)* bereit. Das Löschen von Informationen geschieht über die Schnittstelle *DelPD(sqlStatement)*. Im Gegensatz zum allgemeinen Grundschema der Schnittstellenfacette ist beim Hinzufügen und Löschen von Informationen keine

```
FACET_PDA_MANAGEMENT = [
  Publishers = [
      PDadded(STRING id, INT seqNo, STRING sqlStatement)
      PDdeleted(STRING id, INT seqNo, STRING sqlStatement)
      Resync(STRINGS ids)
  ]
  SubscriptionCallbacks = [
      PDadded(STRING id, INT seqNo, STRING sqlStatement)
      PDdeleted(STRING id, INT seqNo, STRING sqlStatement)
      Resync(STRINGS ids)
  ]
  AddPD(~~STRING id,~~ STRING sqlStatement): STRING THROWS ~~SameAlreadyExistent~~, SqlError
  DelPD(~~STRING id,~~ STRING sqlStatement) THROWS SqlError

  GetPD(STRING sqlStatement): STRING THROWS SqlError
]
```

Abbildung 9.15: Facette einer generischen, SQL-basierten Betriebsdatenverwaltung. Aufgrund von Besonderheiten weichen die Schnittstellen vom Grundschema ab. Die Änderungen sind entsprechend markiert.

Übergabe einer Identität erforderlich, da die eigentlichen Handlungsanweisungen immer explizit in der SQL-Anweisung enthalten sind. Abbildung 9.15 zeigt die Schnittstellenfacette.

Hierarchische Anordnung von Betriebsdatenverwaltungen

Die Betriebsdatenverwaltung ist ein neuralgischer Punkt der Produktionsanlage, da eine hohe Kommunikationslast zu erwarten ist. Deshalb ist eine Aufteilung in mehrere Einheiten sinnvoll, die hierarchisch angeordnet sind. Die Vorgehensweise ist dabei identisch zur Realisierung hierarchischer Katalogdienste. In jeder Zone wird eine Betriebsdatenerfassung installiert, die aufgrund des Datenverkehrsaufkommen als eigenständige Entität mit fixen Namen ausgeführt ist, so dass auch hier ein Schema zur Adressierung und Beschaffung von Referenzen greift (Kap. 9.2 und Abb. 9.11).

9.7 Allgemeine Softwaresysteme

Allgemeine Softwaresysteme übernehmen bspw. Aufgaben der Informationsverarbeitung, der Steuerung höherer Ordnung, der Produktionsüberwachung, des Produkt-Trackings, der Qualitätssicherung, der Diagnose oder sind Bestandteil von Mensch-Maschine-Schnittstellen. Aufgrund der Vielfältigkeit werden an dieser Stelle keine konkreten Anwendungen vorgestellt. Stattdessen soll verdeutlicht werden, wie diese durch die bisher vorgestellten Konzepte für den Einsatz in einem wandelbaren Umfeld gerüstet werden können.

Softwareapplikationen sind als Steuerungsmodelle zu realisieren und können mit anderen Teilen des Gesamtsystems interagieren. Die Wichtigsten sind dabei der Katalogdiensts, die Betriebsdaten- und die Arbeitsplanverwaltung, da sie zentrale Informationen der Produktionsanlage bereitstellen. Werden darüber hinaus die Facetten der Auftrags- und Lagerverwaltung (Abb. 8.10 S. 126, 8.4 S. 120)

um Informationsschnittstellen erweitert, sind insgesamt zu jedem Zeitpunkt umfangreiche Zustandsinformationen der Produktionsanlage ermittelbar.

Wandlungen der Produktionsanlage wirken sich in jedem Fall auf den Katalogdienst aus. Indem sich eine Softwareapplikation an den Publishern dieses Dienstes registriert, wird sie stets über Wandlungsereignisse informiert und kann sich an neue Situationen adaptieren. Zudem besteht für jede Softwareapplikation innerhalb der Produktionsanlage Zugang zur Nachbarschaftserkennung, so dass selbst ohne Katalogdienst strukturelle Wandlungen wahrgenommen werden können.

Applikationen sind im Gegensatz zu allen anderen Systemen nicht auf die Ebenen der Produktionsanlage beschränkt. Stattdessen ist es möglich, diese auch auf höheren Ebenen der Fabrik anzusiedeln. Ihr Zugang zu Entitäten der Produktionsanlage und die Detektion erfolgter Wandlungen läuft dann über ein Gateway, das einen erweiterten Katalogdienst ausführt, der zu jedem Element eine externe Kommunikationsadresse speichert (Kap. 6.8).

Kapitel 10

Referenzimplementierung der Softwarearchitektur und Evaluation der Konzepte

Bei der Umsetzung der Softwarearchitektur und der Steuerungskonzepte lag das Ziel in einer Referenzimplementierung, die Steuerungstechnik, Software für Entitäten sowie Werkzeuge zur Unterstützung der Softwareentwicklung umfasst.

Da die Steuerungskonzepte auf dezentralen Softwaresystemen basieren, mit einem Schwerpunkt auf Wandelbarkeit, wäre ihre Evaluation an einer realen Produktionsanlage nicht zielführend gewesen. Mechanische Umbauten zur Nachstellung von Wandlungsszenarien wären zu zeitintensiv, um systematisch eventuelle Fehler in Algorithmen und Implementierungen zu ermitteln. Deshalb wurde ein Labor für verteilte Steuerungssysteme eingerichtet, in dem die Entwicklungen erprobt wurden.

10.1 Wahl der Steuerungstechnik

Die Wahl der Steuerungstechnik erfolgte unter dem Gesichtspunkt der Homogenität, um auf Hardwareseite eine hohe Systemkonsistenz zu erzielen und auf Softwareseite Mehrfachentwicklungen zu vermeiden. Letztere führen grundsätzlich zu höheren Entwicklungszeiten und vor allem zu Problemen der Konsistenzerhaltung.

Rechentechnik

Bei der Rechentechnik war zwischen PC, SPS und Mikrocontroller (Kap. 2.3.1) eine Entscheidung zu treffen. Aufgaben für Rechentechnik in der Produktion umfassen neben der Steuerung auch die Informationserzeugung, -verarbeitung und -verwaltung (Kap. 2.2). Mikrocontroller finden Einsatz als kostengünstige Lösungen für spezielle Problemstellungen und sind wegen ihrer geringen Leistungsfähigkeit nicht als Plattform geeignet. SPSen werden in Steuerungen eingesetzt, die Bearbeitung anderer Aufgabenstellungen ist mit ihnen standardmäßig nicht vorgesehen. Zudem sind sie mit vergleichsweise hohen Anschaffungskosten behaftet, so dass der Einsatz in anderen Aufgabenfeldern als unwirtschaftlich erscheint. Dagegen besticht der PC einerseits durch Kostenvorteile und andererseits, wegen seiner offenen Architektur, durch die Erweiterbarkeit mit zusätzlicher Hardware. Darüber hinaus sind PCs in unterschiedlichen Leistungsklassen, Qualitätsstufen und Preissegmenten erhältlich, so dass je nach Aufgabenstellung ein passendes System auswählbar ist. Ein weiteres Argument ist ihre „monokulturelle“ Verbreitung auf höheren Fabrikebenen. Grundsätzlich sind die vorgestellten Konzepte auch dort anwendbar, so dass die Integration aller Fabrikrechner denkbar ist. Die Wahl der

Rechentechnik fiel deshalb auf PCs und die Referenzimplementierung erfolgte mit dieser Rechentechnologie.

SPSen und Mikrocontroller werden dennoch nicht komplett ausgeschlossen. Als Geräte interpretiert können sie über Softwaretreiber integriert werden. Kapitel 10.4 zeigt dies exemplarisch anhand einer Mehrfachschnittstellen-Baugruppe auf Mikrocontrollerbasis.

Betriebssysteme

Bei den Betriebssystemen für PCs dominieren die zwei Familien *Unix* und *Microsoft Windows*. Unixsysteme existieren in verschiedenen kommerziellen und freien Varianten und Distributionen. Ein nicht kommerzieller Vertreter ist *Linux*, das in unterschiedlichen Varianten und Versionen verfügbar ist. Es handelt sich hierbei um ein modularisiertes und dadurch skalierbares Betriebssystem, dass – wie alle Unixsysteme – über eine saubere, konsistente Architektur, insbesondere zur Gerätetreibereinbindung, verfügt (alle Gerätezugriffe erfolgen über Dateien). Es ist quelloffen und kostenlos verfügbar. Dagegen ist Microsoft mit seinen verschiedenen Windowsversionen Marktführer, weshalb viele Hersteller Gerätetreiber ausschließlich für dieses Betriebssystem anbieten. Aus Sicht der Steuerungstechnik besticht Windows durch unterschiedlichste Ansätze zur Gerätetreibereinbindung, wobei sich diese zusätzlich in den verschiedenen Versionen unterscheiden. Deshalb ist häufig zu beobachten, dass Steuerungssysteme auf Windowsbasis nur für spezifische Versionen freigegeben sind und Anpassungen an neue Versionen unterlassen werden.

Zwar erfährt Linux in den letzten Jahren zunehmend Unterstützung bei der Integration von Gerätetreibern, jedoch kann Windows aus heutiger Sicht nicht vernachlässigt werden. Dieser Zwiespalt führte zu einer zweigleisigen Berücksichtigung beider Betriebssysteme.

Kommunikationstechnik

Der Einsatz von Kommunikationstechnik in Fabriken ist klassischer Weise durch die Verwendung von Feldbussen abwärts der Produktionsebene und Ethernet aufwärts der Leitstandsebene gekennzeichnet. Allerdings drängen mit unterschiedlichen Industrial-Ethernet-Derivaten echtzeittaugliche Lösungen in den Markt (Kap. 2.3.2), die sich einerseits teilweise durch einen deutlichen Leistungsvorsprung vor klassischen Feldbussen auszeichnen und andererseits als Ebene-3-Protokoll des ISO-OSI-Schichtenmodells das weit verbreitete und daher wichtige TCP/IP in die Feldebene bringen. Dadurch ist erstmalig grundsätzlich eine *homogene* Vernetzung aller Fabrikebenen möglich.

Da sich Ethernet mit TCP/IP für die Konzepte zur Strukturerkennung und Selbstkonfiguration besser eignet als Feldbusse – Letztere erlauben nur den Aufbau von Linien- oder Baumstrukturen, aber keine vermaschten Netze – fiel die Wahl auf Ethernet, wobei für *strukturierte Bereiche* kabelgebundenes Ethernet und für *unstrukturierte Bereiche* WLAN vorgesehen wurde (Kap. 4.2).

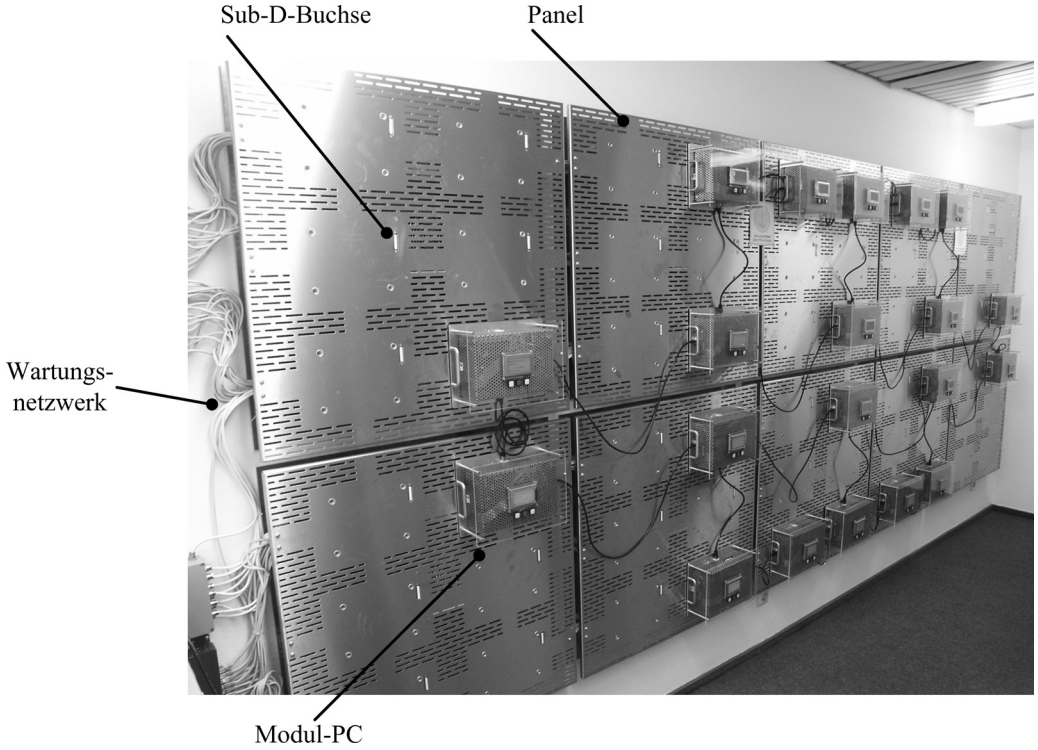


Abbildung 10.1: Laboraufbau. Nicht dargestellt ist der Leitstand, der einem herkömmlichen PC entspricht.

10.2 Aufbau des Labors für verteilte Steuerungssysteme

Im Labor finden sich die vier Komponenten *Modul-PCs, Steckwand, Wartungsnetzwerk* und *Leitstand*, die in Abbildung 10.1 dargestellt sind.

Modul-PCs emulieren Entitäten (Steuerungen oder Softwareapplikationen) und sind an der Wand flexibel steckbar. Jeder Modul-PC verfügt über mehrere Netzwerkschnittstellen, so dass zwischen benachbarten Steuermodulen Verbindungen erstellt werden können.

Indem Modul-PCs bspw. Steuerungen von Arbeitsstationen oder Weichen emulieren und Netzwerkkabel Transfersegmente repräsentieren, wird dem in den Kapiteln 3.4 und 6.1 geforderten Isomorphismus Rechnung getragen. Dadurch ist es möglich Produktionssysteme (ausschnittsweise) nachzubilden. Dies ist prinzipiell auf verschiedenen hierarchischen Ebenen einer Produktionsanlage und für verschiedene Szenarien denkbar. Untersuchbare Szenarien sind Wandlung durch Hinzufügen, Entfernen oder Ausfall von Entitäten und struktureller Umbau sowie Partitionierung und Verschmelzung von Produktionssystemen während des Betriebs.

Modul-PCs

Das Labor verfügt über 23 Modul-PCs auf Grundlage eines 5.25-Zoll Industrie-PCs. Dieser ist mit einem, mit 733MHz getakteten und passiv-gekühlten, Via-Eden-Prozessor geringer Prozessorleistung,

256MB Hauptspeicher und einer Compactflash-Karte mit 256MB Kapazität ausgestattet. Zur Kommunikation stehen fünf 100MBit-Ethernet-Netzwerkschnittstellen zur Verfügung, von denen vier zum Aufbau von Anlagenstrukturen und zur Kommunikation im Anlagennetz dienen. Über die fünfte Schnittstelle erfolgt die Ankopplung an das Wartungsnetz.

Jeder Modul-PC verfügt über periphere Geräte in Form eines graphischen LCDs mit einer Auflösung von 128x64 Pixeln, einer Audioausgabe und 16 digitale Schnittstellen. Letztere sind über den USB-Bus angebunden und als Ein- bzw. Ausgänge frei konfigurierbar. Sie werden im Modul als Eingänge für Taster und Schalter genutzt. Abbildung 10.2 zeigt einen Modul-PC.

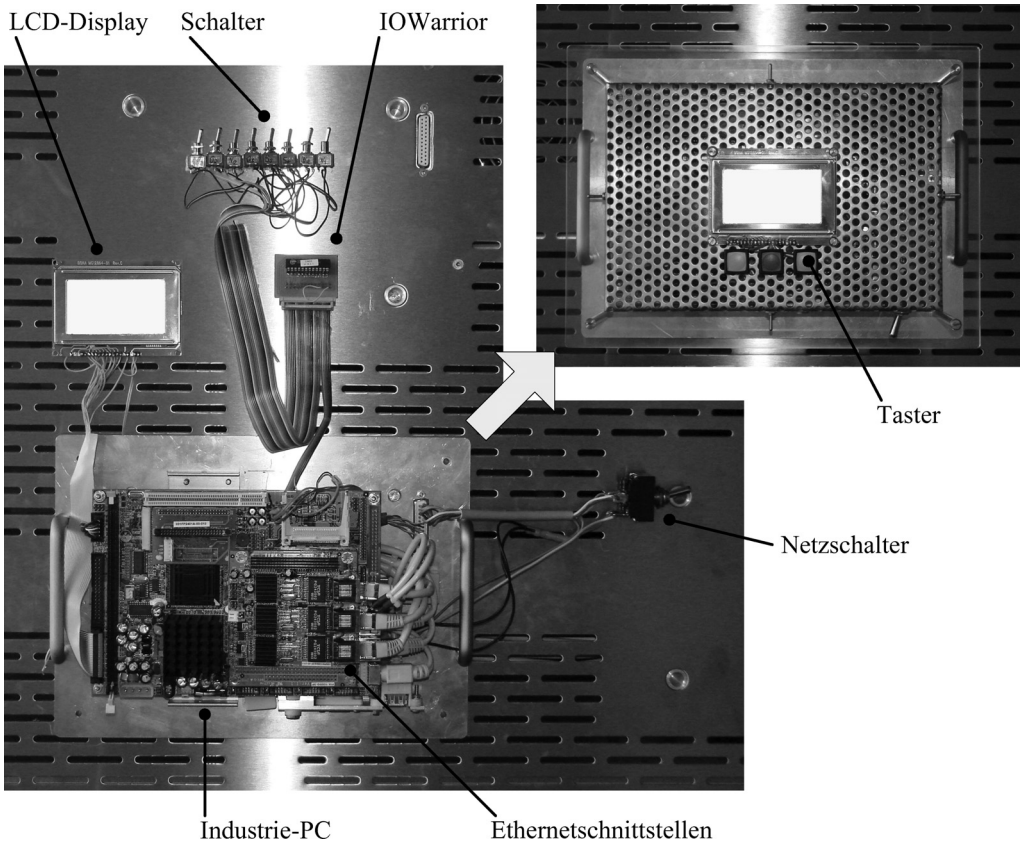


Abbildung 10.2: Modul-PC auf Linuxbasis.

Als Betriebssystem kommt die Eigendistribution *FINUX* (FAPS-Linux) zum Einsatz. Sie basiert im Wesentlichen auf dem Linux-Kernel 2.6.16 [60], Busybox 1.0.1 [12] und verfügt über eine abgespeckte Pythoninstallation 2.4 [81]. Dabei orientiert es sich am Einsatz in Embedded-Umgebungen und zeichnet sich durch einen geringen Festspeicherbedarf von weniger als 5MB aus. Das Betriebssystem ist zweifach vorhanden, wovon immer nur eines aktiv ist. Das nicht aktive System kann über das Netzwerk aktualisiert werden, so dass die Module hotupdatefähig sind. Bei Neustart wird das jeweils aktuellste Betriebssystem gestartet. So beschränken sich Ausfallzeiten durch Aktualisierungen auf die reine Zeit des Neustarts und somit auf wenige Sekunden.

Steckwand, Wartungsnetzwerk und Leitstand

Die Steckwand besteht aus zehn Panelen mit jeweils zentraler Stromversorgung und sechs Steckplätzen für Steuermodule. Die Stromversorgung und Ankopplung an das im Hintergrund verlegte Wartungsnetzwerk erfolgt in Form einer 25-poligen Sub-D-Buchse über eine gemeinsame Schnittstelle. Über zwei Switches mit jeweils 48 Kanälen sind die Kommunikationsschnittstellen aller Steckplätze zu einem Subnetz zusammengeschaltet. Modul-PCs können durch einfaches Aufstecken an der Wand integriert werden, es müssen keine Verkabelungen für die Betriebsbereitschaft vorgenommen werden.

An das Wartungsnetzwerk sind Entwicklungsrechner gekoppelt. Diese erlauben zum einen die aufwandsarme Verteilung neuer Software und ermöglichen zum anderen die Kontrolle über die Modul-PCs, wie Starten, Stoppen von Programmen oder Rechnerneustart, etc.

Der Leitstand ist in Form eines Standard PCs realisiert, der mit Linux als Betriebssystem betrieben wird. Ein alternativer Einsatz von Windows ist technisch möglich. Als Teil der Produktionsanlage ist der Leitstand nicht über das Wartungsnetzwerk mit den Steuermodulen verbunden. Stattdessen erfolgt die Ankopplung über das jeweils realisierte Anlagennetzwerk.

10.3 Softwaresysteme für Entitäten

Die Software für Entitäten muss fünf grundlegende Aufgabenbereiche abdecken, um Wandlungsfähigkeit auf allen Ebenen zu erreichen. Diese sind:

- Implementierung der Steuerungsarchitektur (Kap. 5) zur Integration entitätenübergreifender, geräteunabhängiger Steuerungsmodelle
- Übergreifende Kommunikations-Middleware zur Entitäteninteraktion zwischen Steuerungsmodellen
- Optionale Integration spezifischer Geräte, falls Entitäten Gerätesteuerungen sind
- Automatische Erkennung der Struktur der Produktionsanlage
- Selbstkonfiguration der Netzwerkschnittstellen

Softwarepakete und Systemarchitektur der Entitäten

Im Vorfeld der Realisierung war zu entscheiden, ob die genannten Aufgaben in ein monolithisches Softwarepaket zusammengefasst werden oder eine Trennung in einzelne, thematisch abgeschlossene, miteinander interagierende Einzelpakete erfolgen sollte. Zwar führt letztere Lösung zu komplexerer Software, da pro Entität mehrere Prozesse ablaufen und aufeinander abzustimmen sind, allerdings können Einzellösungen einfacher entwickelt und getestet werden. Zudem verfügen sie über den Vorteil, auch in konzeptfremden Anwendungen verwendet werden zu können. Deshalb wurde dieser Ansatz verfolgt und folgende Softwarepakete entwickelt:

- Plant-O-Matic – Steuerungsarchitektur

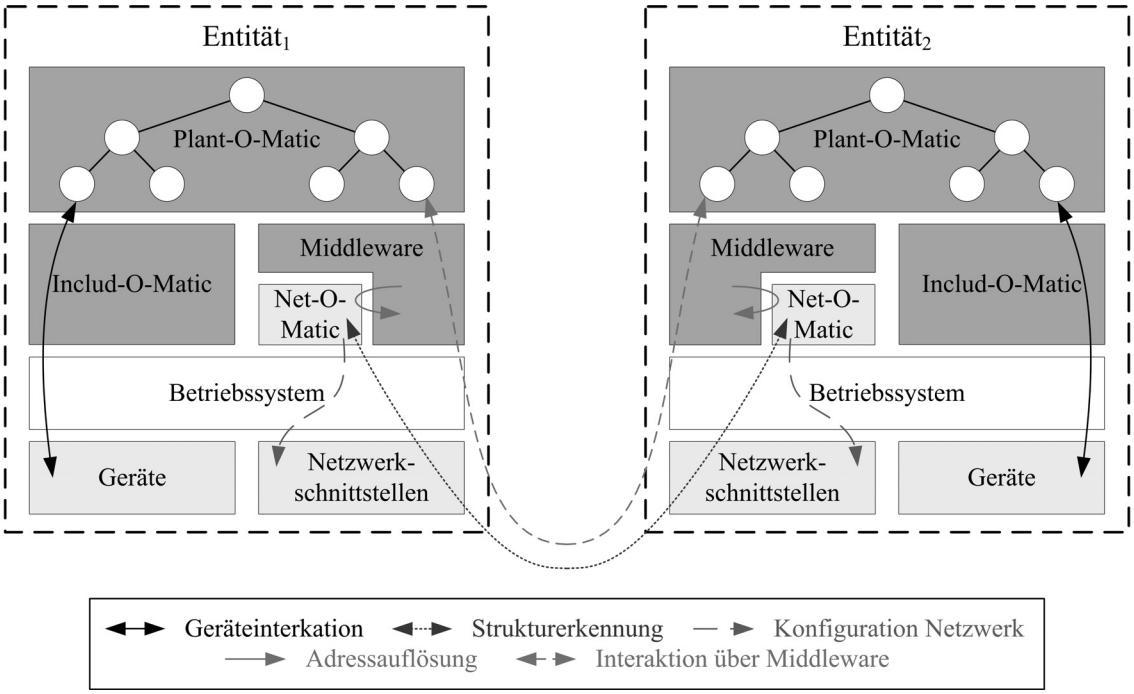


Abbildung 10.3: Anordnung der Softwarepakete in Entitäten.

- ComManager – Kommunikations-Middleware
- Includ-O-Matic – Abstraktionsschicht zum einheitlichen Gerätezugriff
- Net-O-Matic – Strukturerkennung und Selbstkonfiguration der Netzwerkschnittstellen

Mit den Softwarepaketen erstellte Systeme implementieren eine Systemarchitektur (Abb. 10.3), die aus mehreren kooperierenden Einzelprogrammen zusammengesetzt ist. Plant-O-Matic, ComManager und Includ-O-Matic sind dabei in einem einzelnen Steuerungsprogramm zusammengefasst, das die Ablaufsteuerung und die Ausführungssteuerung (Kap. 3.4) mit Gerätezugriff realisiert. Ohne die Verwendung von Includ-O-Matic entstehen reine Softwareanwendungen, wie sie für Informationsdienste und allgemeine Softwaresysteme notwendig sind (Kap. 9).

Auswahl der Programmiersprachen

Die Auswahl der Programmiersprache war von entscheidender Bedeutung, da mehrere Randkriterien zu beachten sind. Quellcode wird entweder kompiliert oder interpretiert. Wird er kompiliert muss ein (Cross-)Compiler für das Zielsystem verfügbar sein. Wird er interpretiert, muss der Interpreter auf dem Zielbetriebssystem vorhanden sein.

Kompilierte Programme sind zwar in Bezug auf die Ausführungsgeschwindigkeit den interpretierten Programmen deutlich überlegen, allerdings wirken sich zeitintensive Compiliervorgänge negativ auf Testzyklen aus. Interpretierte Programme müssen nicht „vorbehandelt“ werden, weshalb in der Regel

eine schnellere Entwicklung möglich ist. Deshalb ist es sinnvoll von beiden Seiten zu profitieren. Prozessorlastige Algorithmen werden in einer kompilierten Sprache implementiert, die nicht zeitintensive Verschaltung verschiedener Komponenten zu komplexen Softwaresystemen und ihre Koordination kann mit einer Skriptsprache erfolgen.

Eine Programmiersprache muss betriebssystemspezifische Besonderheiten kapseln und dem Entwickler einen möglichst einheitlichen Zugriff auf Betriebssystemressourcen, wie z.B Dateien oder Netzwerkschnittstellen, bieten. Ansonsten sind Mehrfachimplementierungen die Folge, mit entsprechend hohen Wartungsaufwendungen und Inkonsistenzen. In diesem Zusammenhang ist auch die Verfügbarkeit umfangreicher, plattformneutraler Bibliotheken von entscheidender Bedeutung, um Standardaufgaben nicht reimplementieren zu müssen.

Ein weiterer Aspekt ist die Wahl des Programmierparadigmas [120]. In Kapitel 5.1 wurde die besondere Eignung des Componentware-Ansatzes mittels der objektorientierten Programmierung erörtert. Aus der Menge der verbreiteten objektorientierten Sprachen *C++, Delphi, Eiffel, Java, Python, Ruby, Smalltalk* wurde Python als Hauptprogrammiersprache ausgewählt. Python ist eine objektorientierte, interpretierte Skriptsprache, die für eine Vielzahl von Betriebssystemen verfügbar ist und sich durch eine einfache Erlernbarkeit, ein hohes Maß an Flexibilität und umfangreiche Bibliotheken auszeichnet. Sie erlaubt das Erstellen von sehr kompakten, gut lesbaren Quellcodes und weist dabei ein hohes „Zeilen/Nutzen-Verhältnis“, mit positiven Auswirkungen auf Entwicklungszeiten, auf [81]. Zudem ermöglicht Python das einfache Einbinden externer Bibliotheken über *ctypes* [105], die in gängigen Compilersprachen verfasst sein können. Damit ist es möglich, Programmteile, die eine hohe Ausführungsgeschwindigkeit benötigen oder nur in Form von kompilierten Bibliotheken vorliegen, problemfrei einzubinden.

Zur Einbindung externer Bibliotheken, primär zur Integration herstellerspezifischer Gerätetreiber, unterstützt Includ-O-Matic exemplarisch *GnuC++, Visual C++* und *Delphi*.

Net-O-Matic – Nachbarschaftserkennung und Selbstkonfiguration des Kommunikationsnetzes

Die in Kapitel 6.4 und 6.7 vorgestellten Verfahren zur Strukturerkennung und Selbstkonfiguration des Netzwerks wurden in dem Programmpaket *Net-O-Matic* umgesetzt, das in Python realisiert ist und für Linux und Windows zur Verfügung steht. Das Paket besteht aus den beiden Programmen *NetStructure* und *NetConfig* sowie der Klasse *NetRepository*.

NetStructure implementiert den Algorithmus zur Nachbarschaftserkennung. Hierzu wird auf jeder Entität ein Sende-Thread ausgeführt, der Informationen zu lokalen Nachbarn über alle Netzwerkschnittstellen verteilt. Für jede Schnittstelle läuft ein eigener Empfangs-Thread, der Nachrichten schnittstellenspezifisch entgegennimmt, so dass eine Richtungszuordnung erfolgen kann. Die Nachrichten werden – sofern sie neue Strukturinformationen enthalten – über die restlichen Netzwerkschnittstellen weitergeleitet. In diesem Fall wechselt *NetStructure* in den Zustand *instabile Phase*. Werden über einen freikonfigurierbaren Zeitraum keine Änderungen mehr detektiert, wechselt das Programm zurück in den Zustand *stabile Phase* (Kap. 6.7).

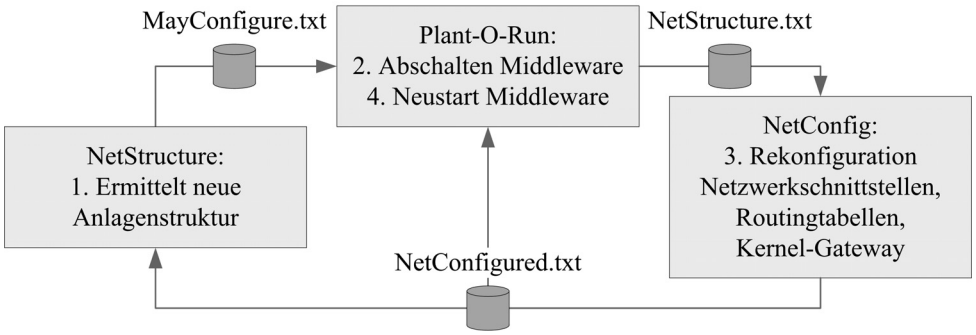


Abbildung 10.4: Ablauf der kommunikationstechnischen Adaption an Wandlungsvorgänge.

Bei dem Wechsel zurück berechnet *NetStructure* deterministisch sowohl die eigene neue IP-Adresse als auch die der anderen Entitäten im gesamten System (Kap. 6.7). Zusammen mit der Anlagenstruktur werden diese Informationen in einer Datei „NetStructure.txt“ auf dem Festspeicher abgelegt.

Die zweite Applikation *NetConfig* überwacht diese Datei, erkennt dabei Veränderungen, liest sie aus und veranlasst die Neukonfiguration der Kommunikationsschnittstellen sowie der Routentabellen. Den Abschluss dieser Umkonfigurationen signalisiert das Programm über eine weitere Datei „NetConfigured.txt“, die sie leer überschreibt (der Schreibevorgang, nicht der Dateiinhalt ist an dieser Stelle von Bedeutung).

Ein mit Plant-O-Matic realisiertes Softwareprogramm einer Entität kommuniziert über den *ComManager* mit anderen Entitäten (Kap. 10.3). Es muss Sorge getragen werden, dass Umkonfigurationen nicht durchgeführt werden, während Aufrufe über das Kommunikationsnetz stattfinden. Der *ComManager* muss kontrolliert abgeschaltet werden, da andernfalls Verbindungsfehler resultieren und zugrunde liegende TCP-Streams zusammenbrechen könnten. Insbesondere in Bezug auf *exactly once Aufrufsemantiken* wäre dies unvorteilhaft.

Aus diesem Grund existiert eine dritte Datei „MayConfigure.txt“, die aktualisiert werden muss – auch hier ist der Schreibvorgang von Bedeutung – bevor eine Umkonfiguration der Netzwerkschnittstellen erfolgen darf. Hierzu integriert das Softwareprogramm eine Instanz der Klasse *NetRepository*, über die sich der *ComManager* über Strukturänderungen des Gesamtsystems informieren lassen kann. Tritt ein solcher Fall ein, schaltet der *ComManager* kontrolliert ab. Erst danach signalisiert *NetRepository* an *NetConfigure* die Erlaubnis zur Umkonfiguration. Indem die Instanz auf „NetConfigured.txt“ lauscht, können danach die Kommunikationsverbindungen neu aufgesetzt werden und die Softwareprogramme wieder interagieren. Abbildung 10.4 stellt diese Vorgänge schematisch dar.

Net-O-Matic realisiert damit das kommunikationstechnische Reaktionsvermögen von Entitäten im wandelbaren Umfeld.

Plant-O-Matic – Referenzimplementierung der Softwarearchitektur

Bei Plant-O-Matic handelt es sich um eine erweiterbare, in Python erstellte, Softwarebibliothek, in die die Bausteine der auf Wandelbarkeit ausgerichteten Softwarearchitektur aus Kapitel 5 implementiert sind. Alle in den vorangegangenen Konzeptionskapiteln vorgestellten Knoten und Facetten sind

enthalten. Weitere Bestandteile sind zusätzlich entwickelte Basisbausteine, die von Facetten häufig genutzte Grundfunktionalitäten beinhalten. Zu ihnen zählen Publisher, Subscriber, SubscriberAgency, Threadpool, Timer, Mehrfach-Leser-Schreiber-Lock, Datenkonverter zur generischen Serealisierung von Objekten zur Netzwerkübertragung oder Speicherung, UUID-Generator, Dateiüberwachung, BroadcastSockets, Zugriff auf Netzwerkschnittstellen, Konfigurationsdatei-Parser, u.a.

Mit Hilfe der Bibliothek ist die Erstellung von Steuerungen (als geräteunspezifischer Teil das Steuerungsmodell) bzw. Softwareapplikationen für Entitäten möglich, die unter Linux und Windows betrieben werden können. Hierzu erstellt der Entwickler ein Pythonprogramm, indem benötigte Facetten ausgewählt, konfiguriert und in Knoten organisiert werden. Durch eine hierarchische Verknüpfung mehrerer Knoten entstehen Baumstrukturen (Kap. 5.3.1), deren Wiederverwendbarkeit zur Reduktion von Entwicklungsaufwendungen führen. Anhand der Organisation als Bäume erfolgt die Adressierung von Knoten, Facetten und Publishern (Kap. 5). Adressen werden für den Entwickler transparent in Referenzen von lokalen oder entfernten Komponenten aufgelöst, so dass zwischen der lokalen und rechnergrenzenüberschreitenden Interaktion weitgehend kein Unterschied besteht. Dies trägt zu einer Reduktion der Entwicklungszeiten bei. Lediglich bei Ausfällen von Partnerentitäten besteht ein Unterschied zwischen lokalen und entfernten Komponenten, da lokale Ausfälle im Gegensatz zu entfernten nicht detektiert werden können (das Softwareprogramm wird entweder ausgeführt oder nicht).

Die entitätenübergreifende Interaktion wird durch den *ComManager* realisiert, über den jede Entität verfügt. Er unterhält zu allen Partnerentitäten – für den Entwickler transparent – Kommunikationsverbindungen und passt sie ggf. nach Wandlungen an neue Situationen an. Der Zugriff auf entfernte Komponenten erfolgt über das Adressierungsschema (Kap. 4.4.1), wobei bei der Adressauflösung nicht lokale Adressen erkannt und über den *ComManager* dann Stellvertreter (*Stubs*) erzeugt werden. Diese leiten lokal erzeugte Aufrufe, für den Entwickler transparent, über die Middleware zum Empfänger weiter.

Includ-O-Matic – Integration peripherer Geräte

Mit den geräteunspezifischen Steuerungen von Plant-O-Matic allein ist kein Anlagenbetrieb möglich, da zum Zwecke der Wiederverwendbarkeit von Facetten bewusst keine direkte Integration realer Geräte vorgesehen ist. Deshalb ist eine Möglichkeit zur aufwandsarmen Ankopplung von Peripherie (Sensorik, Aktorik, Geräte) erforderlich. Die Beschäftigung mit der Fragestellung der Integration zeigt, dass kein etablierter, einheitlicher Standard existiert. Je nach Hersteller und teilweise je nach Typ erfolgt die Anbindung über unterschiedliche Hardwareschnittstellen, wie z.B. USB, Seriellport, Parallelport, Ethernet oder Steckkarten sowie proprietäre Softwareschnittstellen und Protokolle. Jeder Geräteanbieter entwickelt deshalb seine eigenen, proprietären Treiber, woraus insgesamt enorme systemintegratorische Entwicklungsaufwendungen resultieren.

Mit Includ-O-Matic steht eine Abstraktionsschicht zur Verfügung, die Plant-O-Matic einen standardisierten Zugriff auf Peripherie ermöglicht, unabhängig von Herstellern, Hard- und Softwareschnittstellen sowie Protokollen. Dies wird dadurch erreicht, dass für jeden Peripherietyp eine Standardisierung der Funktionalität in ein ideales Peripheriemodell erfolgt und dabei Standardschnittstellen festgelegt werden. Herstellerspezifische Treiber werden in - ebenso spezifische - *Includ-O-Matic-Treiber* gekapselt, die allerdings nach außen zu jeder Peripherie die standardisierten Standardschnittstellen anbieten. Alle spezifischen Details der Hardwareankopplung bleiben dem Entwickler verborgen. Jede

Peripherie wird in ein eigenes, von allen anderen unabhängiges, Softwareobjekt gekapselt. Dadurch erfolgt bei Mehrfach-Schnittstellenhardware, wie zum Beispiel Multi-Digital/Analog-IO-Karten, eine Abstraktion. Für den Entwickler wird es bedeutungslos, ob etwa ein digitaler Eingang über das Setzen von Registern im Kontext mit anderen digitalen Eingängen anzusprechen ist (oft entspricht ein digitaler IO einem Bit in einem n-Bit Register).

Alle Schnittstellenobjekte aller Includ-O-Matic-Treiber werden von einer zentralen Peripherieverwaltung verwaltet. Jedes Schnittstellenobjekt kann eindeutig identifiziert werden, um so die notwendige logische Verbindung der Software zum physischen Objekt herzustellen. Die Identität ist dabei ein Tripel aus *Treiber-Id, SuperDevice-Id* und *Device-Id*. Durch dieses Schema ergibt sich eine dreistufige Hierarchie, die aus dem Umstand resultiert, dass über einen Herstellertreiber häufig mehrere Mehrfach-Schnittstellenhardware angesprochen werden können. Jede Hardware gilt deshalb als SuperDevice und die eigentliche Schnittstelle als Device. Die Verwaltung hat zudem die Aufgabe, zu jedem Zeitpunkt für jedes Schnittstellenobjekt exklusiv nur einen Nutzer zu erlauben, um so konkurrierende und dadurch kritische Mehrfachzugriffe zu unterbinden. Ein Nutzer kann ein Schnittstellenobjekt jedoch wieder freigeben, so dass ein anderer Nutzer darauf zugreifen kann.

Die Ankopplung an Plant-O-Matic erfolgt dadurch, dass zu jeder Hardwareschnittstelle im System eine Schnittstellenfacette das zugehörige Schnittstellenobjekt kapselt. Dadurch wird es möglich, ein Steuerungsmodell von Plant-O-Matic flexibel und aufwandsarm an unterschiedliche Hardware zu koppeln. Schnittstellenfacetten finden sich in Kapitel 5.6. Beispiele, die von diesem Ansatz Gebrauch machen, sind das generische Verzweigungsmodell für Kreuzungen und Weichen für Transfersysteme (Kap. 7.4) und die Steuerung für Lager (Kap. 8.2).

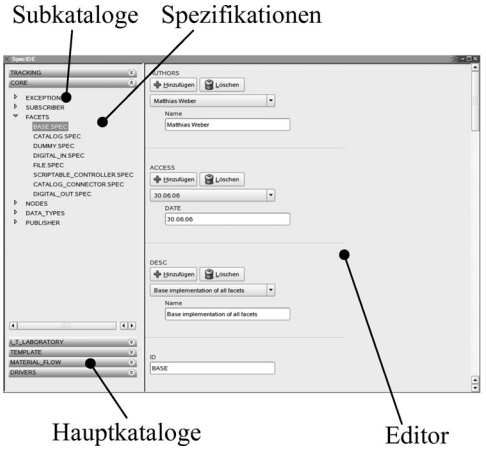
Includ-O-Matic kann losgelöst von Plant-O-Matic genutzt werden. Im Gegensatz zu diesem ist eine Plattformunabhängigkeit der Includ-O-Matic-Treiber aufgrund der herstellerspezifischen Gerätetreiber nicht automatisch gegeben.

10.4 Entwicklungswerkzeuge

Plant-O-Matic und Includ-O-Matic basieren streng auf der Standardisierung der Schnittstellen ihrer Komponenten. Deshalb existieren für die in den vorangegangenen Kapiteln vorgestellten Facetten formale Spezifikationen. Auch Includ-O-Matic beruht auf Spezifikationen der Gerätetypen. Da Spezifikationen auf irgendeine Weise in Quellcode übergeführt werden müssen, wurden mit Plant-O-Dev und Includ-O-Dev Werkzeuge erstellt, die Entwickler bei ihrer Tätigkeit unterstützen und Entwicklungsaufwendungen reduzieren helfen. Sie basieren auf einer automatisierten Quellcodeerzeugung. Die Implementierung von Funktionalität erfolgt danach von Hand.

Plant-O-Dev – Werkzeugkette für Plant-O-Matic

Plant-O-Dev besteht einerseits aus einem hierarchischen Katalog sowie Werkzeugen zur Verwaltung von Spezifikationen und andererseits aus einer Menge von Quellcodegeneratoren für Python, die Spezifikationen in Quellcode übersetzen.




```
BASE = OBJECT(develsys.spec_defs.SpecFacet.Object) [
        AUTHORS = LIST [
                STRING: Matthias Weber
        ]
        ACCESS = DICT [
                30.07.2006 = STRING: NONE
        ]
        DESC = LIST [
                STRING: Base implementation of all
                        facets
        ]
        ...
]
```

Abbildung 10.5: Links – Graphisches Eingabewerkzeug für Spezifikationen. Rechts – Erzeugte formale Spezifikation zur Weiterverarbeitung.

Spezifikationen werden für Knoten, Facetten, Publisher, Subscriber und einfache wie komplexe Datentypen (Listen, Dictionaries und Strukturen) erstellt. Sie verweisen auf andere Spezifikationen, etwa bei der Nutzung von Datentypen für Attribute, Funktions- oder Funktionsrückgabeparameter. Hierdurch entstehen komplexe, gerichtete Spezifikationsgraphen, die allerdings anfällig für manuelle Fehler sind. Deshalb wurde ein Kommandozeilenwerkzeug entwickelt, das das Erzeugen, Löschen und Verschieben von Spezifikationen im Katalog ermöglicht. Das Löschen einer Spezifikation wird nur durchgeführt, wenn keine andere Spezifikation auf diese verweist. Wird eine Facette verschoben, werden die Verweise in allen anderen Spezifikationen angepasst. Dadurch kann eine hohe Konsistenz erreicht werden. Zudem existiert ein Werkzeug, dass Spezifikationen auf ihre Konsistenz überprüft. So werden bspw. Verweise auf nicht existente Spezifikationen (Verschreibefehler) systematisch erkannt und können vom Entwickler ausgebessert werden. Als Alternative zur rein textuellen Erstellung der Spezifikationen wurde prototypisch ein graphisches Eingabewerkzeug entwickelt, das in Abbildung 10.5 zu sehen ist.

Zur Beschreibung der Spezifikationen wurde eine eigene einfache Spezifikationssprache entwickelt. Listing 10.1 zeigt exemplarisch die Spezifikation der Basisfacette, auf der alle weiteren Facetten basieren. Alle in den Kapiteln 5, 7, 8 und 9 dargestellten Facettenbeschreibungen sind Vereinfachungen der zugrunde liegenden Spezifikationen.

```
1  BASE = OBJECT(util.spec_defs.SpecFacet.Object) [
2      INHERITS_FROM = STRING: NONE
3      MODULES = LIST [
4      ]
5      INIT_ATTRIBUTES = DICT [
6          INFO = STRING: SPEC CORE/DATA_TYPES/FACET/INFO
7          NODE = STRING: SPEC CORE/NODE
8      ]
9      ATTRIBUTES = DICT [
10     ]
11     NODES = DICT [
12     ]
13     FACETS = DICT [
14     ]
```

```
15      PUBLISHER = DICT [
16      ]
17      SUBSCRIPTIONS = DICT [
18      ]
19      METHODS = DICT [
20          GET_STATE = DICT [
21              EXPORT = STRING: NO
22              PARAMETERS = DICT [
23              ]
24              RETURN_TYPE = STRING: SPEC CORE/DATA_TYPES/GENERAL/PRIMITIVES/STRING
25              THROWS = LIST [
26              ]
27          ]
28          SET_STATE = DICT [
29              EXPORT = STRING: NO
30              PARAMETERS = DICT [
31                  STATE = STRING: SPEC CORE/DATA_TYPES/GENERAL/PRIMITIVES/STRING
32              ]
33              RETURN_TYPE = STRING: NONE
34              THROWS = LIST [
35              ]
36          ]
37          GET_INFO = DICT [
38              EXPORT = STRING: YES
39              PARAMETERS = DICT [
40              ]
41              RETURN_TYPE = STRING: SPEC CORE/DATA_TYPES/FACET/INFO
42              THROWS = LIST [
43              ]
44          ]
45          GET_NODE = DICT [
46              EXPORT = STRING: NO
47              PARAMETERS = DICT [
48                  LOCATOR = STRING: SPEC CORE/DATA_TYPES/GENERAL/PRIMITIVES/
                        LOCATOR
49              ]
50              RETURN_TYPE = STRING: SPEC CORE/NODE
51              THROWS = LIST [
52              ]
53          ]
54          GET_PUBLISHER = DICT [
55              EXPORT = STRING: YES
56              PARAMETERS = DICT [
57                  LOCATOR = STRING: SPEC CORE/DATA_TYPES/GENERAL/PRIMITIVES/
                        LOCATOR
58              ]
59              RETURN_TYPE = STRING: SPEC CORE/PUBLISHER/BASE
60              THROWS = LIST [
61              ]
62          ]
63      ]
64  ]
```

Listing 10.1: Exemplarische Spezifikation der Basisfacette. Alle Weiteren sind von dieser abgeleitet.

Mit Plant-O-Matic ergibt sich eine Vorgehensweise, wie sie in Abbildung 10.6 abgebildet ist. Zunächst wird manuell eine Spezifikation erstellt und im Katalog gespeichert. Anschließend erfolgt die automatisierte Übersetzung in Python-Quellcode und die manuelle Implementierung der Funktionalität. Die so erstellten Pythonmodule stehen dann für die Entwicklung von Steuerungssoftware oder Softwareapplikationen zur Verfügung.

Die Werkzeuge von Plant-O-Dev sind unter Linux wie Microsoft Windows lauffähig, so dass unter beiden Betriebssystemen entwickelt werden kann.

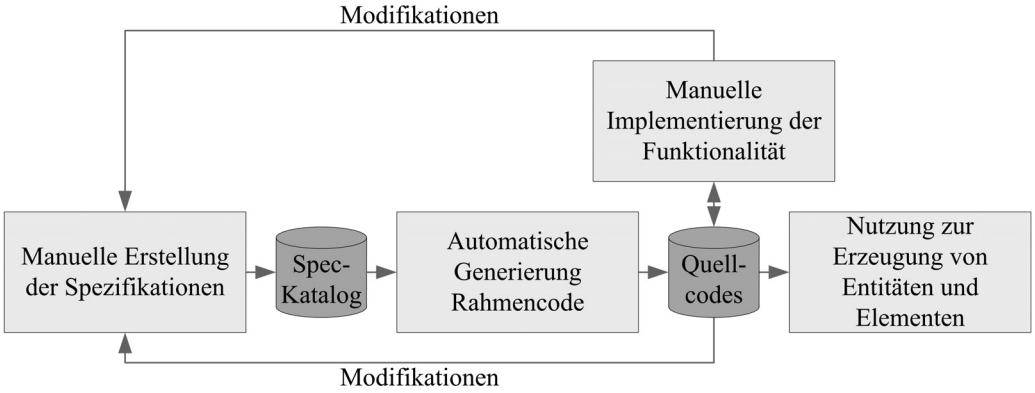


Abbildung 10.6: Vorgehensweise bei der Entwicklung von Komponenten für Plant-O-Matic.

Includ-O-Dev – Werkzeugkette für Includ-O-Matic

Im Allgemeinen ist die Einbindung von Gerätetreibern ein aufwändiges Unterfangen, da verschiedenste Wege zur Integration von Peripherie existieren und von Geräteherstellern auch genutzt werden. Sie lassen sich grob in zwei Hauptgruppen mit zwei bzw. mehreren Untergruppen einteilen. Die Hauptgruppen sind

- Zugriff über Treiberbibliotheken
- Zugriff über Standardhardwareschnittstellen

Treiberbibliotheken stehen entweder vorkompiliert (statisch / dynamisch) oder im Quellcode zur Verfügung. Bei Zugriff über Standardhardwareschnittstellen werden in der Regel zwar keine Treiber mitgeliefert (obwohl eine Kapselung in Bibliotheken technisch möglich ist), jedoch Beschreibungen definierter Kommunikationsprotokolle. Standard-Hardware umfasst beispielhaft Ethernet, TCP/IP, Seriellport, Parallelport und USB-Bus.

Aufgrund dieser Vielfalt und verstärkt durch die Zielsetzung, dass Includ-O-Matic-Treiber beliebige Peripherie je nach Typ in standardisierte Peripheriemodelle kapselt, ist die Erstellung der Treiber mit Aufwand verbunden. Deshalb wurden zur Reduzierung der Integrationsaufwendungen Werkzeuge entwickelt. Sie basieren analog zu Plant-O-Dev auf der formalisierten Spezifikation von Peripheriemodellen einerseits und auf Quellcodegeneratoren andererseits. Es existieren eine Reihe von Generatoren, um die verschiedenen Integrationsmöglichkeiten abzudecken. Während Standardschnittstellen rein in Python gekapselt und die Kommunikationsprotokolle darin implementiert werden, sind für

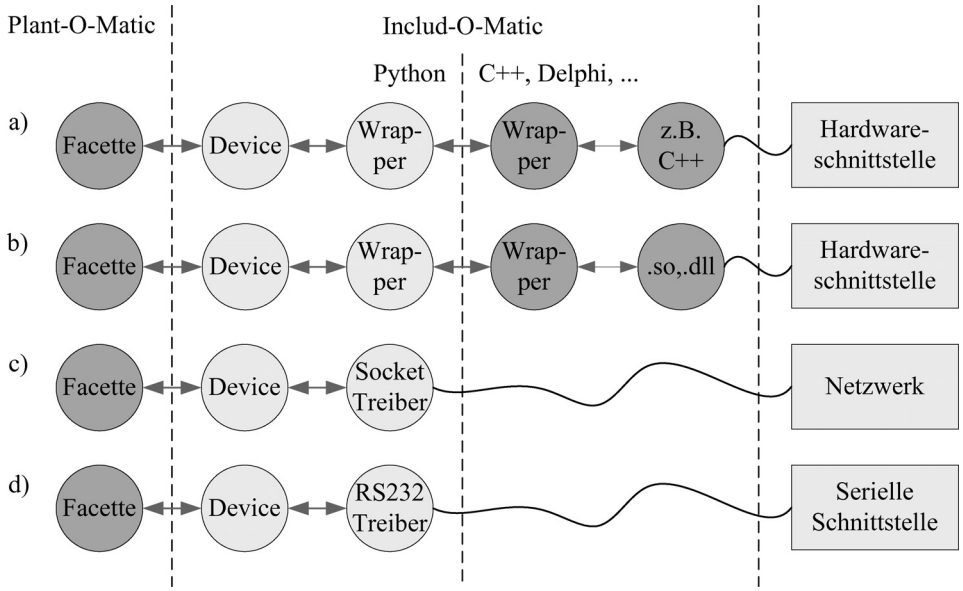


Abbildung 10.7: Mögliche Arten des Zugriffs auf Peripherie am Beispiel einer Digital-Eingangs-Facette von Plant-O-Matic. a) Herstellerspezifischer Gerätetreiber in C++. b) Herstellerspezifischer Gerätetreiber als vorkompilierte Bibliothek unter Linux oder Windows. c) Gerätekopplung über Ethernet. d) Gerätekopplung über RS232.

vorkompilierte oder im Quellcode vorliegende Bibliotheken ein Wrapper in Python und ein Wrapper in der Zielsprache der Bibliothek zur Anbindung notwendig. Includ-O-Dev unterstützt dazu exemplarisch die Sprachen GnuC++ (Linux) , VisualC++ (Windows) und Delphi. Abbildung 10.7 verdeutlicht die unterschiedlichen Kopplungsmöglichkeiten, während Abbildung 10.8 die Vorgehensweise bei der Erstellung der Includ-O-Matic-Treiber zeigt.

Includ-O-Matic-Treiber sind im Allgemeinen nicht plattformneutral, d.h. sie müssen unter Umständen für Linux und Windows zweifach implementiert werden. Dies liegt darin begründet, dass Programmiersprachen nicht immer auf jeder Plattform verfügbar sind, Quellcode plattformspezifische Teile enthalten kann oder Kompilate betriebssystemspezifisch sind. Durch die Verwendung betriebssystemspezifischer Bibliotheken wird so beispielsweise die Plattformunabhängigkeit eingeschränkt. Bei sorgfältiger Entwicklung in C++ ist jedoch Plattformneutralität zu erreichen. In Python implementierte Treiber sind immer auf beiden Betriebssystemen einsetzbar.

Includ-O-Dev wurde exemplarisch an zwei verschiedenen Multi-Schnittstellen-Geräten validiert. Im ersten Fall handelt es sich um eine, am Lehrstuhl FAPS entwickelte, Baugruppe auf Basis des IO-Warriors [15] (Abb. 10.9, links), im zweiten Fall um eine, am Lehrstuhl FAPS mitentwickelte, Baugruppenfamilie auf Basis eines Rabbit-Core-Mikrocontrollers [82] (Abb. 10.9, rechts).

IO-Warrior – Digitale USB-Schnittstellencontroller

Der IO-Warrior ist ein über den USB-Bus anschließbarer Controller, für den unter Linux und Windows Treiber im Quellcode vorliegen. Er ist in drei Varianten mit 16, 32 oder 50 digitalen Schnittstellen, die wahlweise als Ein- bzw. Ausgänge genutzt werden können, verfügbar. Die Erstellung des

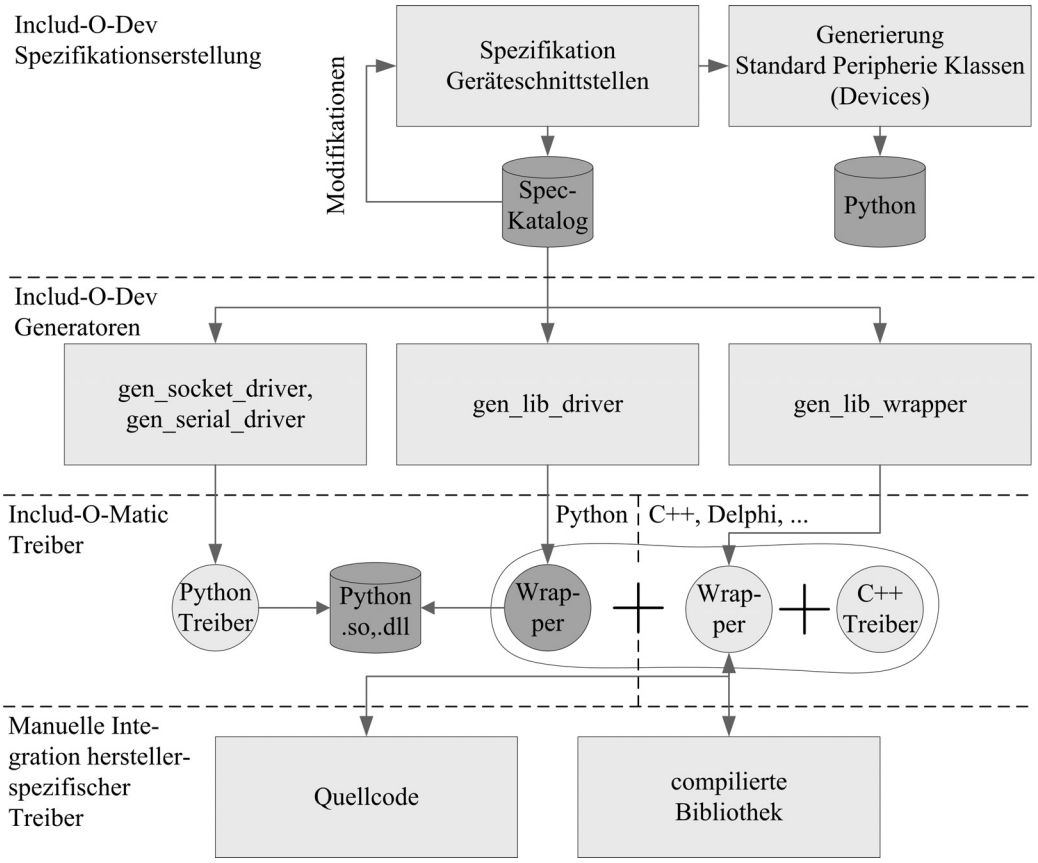


Abbildung 10.8: Werkzeugkette für die Erstellung von Includ-O-Matic-Treibern zur Geräteintegration.

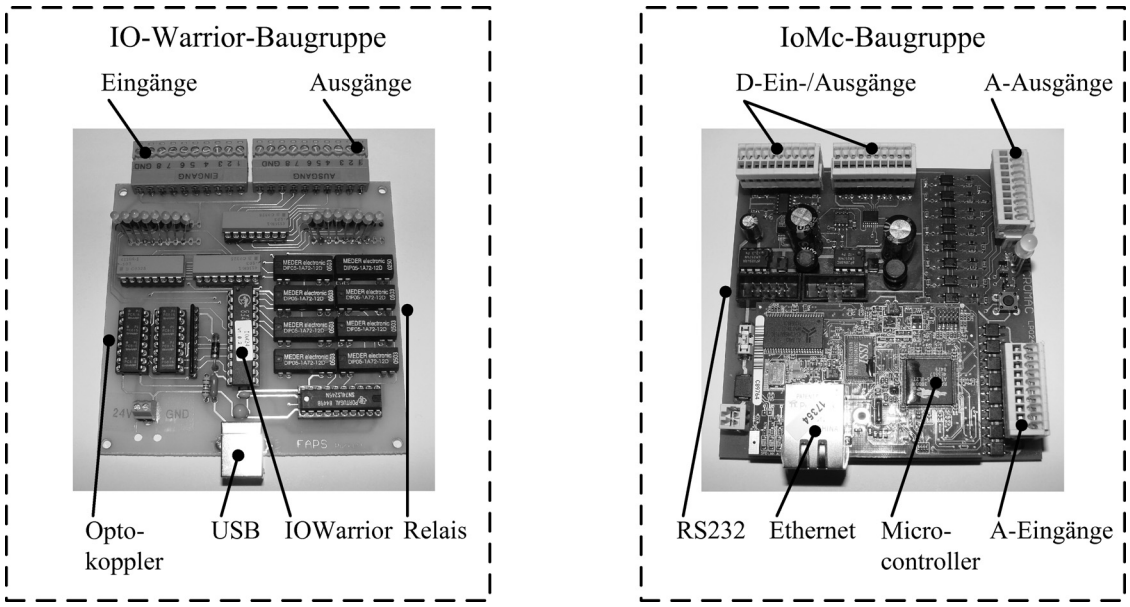


Abbildung 10.9: Links – IO-Warrior Baugruppe zur Ankopplung von acht digitalen Ein- und Ausgängen über USB. Rechts – IoMc in der Ausführung acht digitale, vier analoge und vier RS232 Ein- und Ausgänge.

Includ-O-Matic-Treibers erfolgte mit der Programmiersprache C++. IO-Warrior mit 16 Schnittstellen sind in den PC-Modulen des Testlabors verbaut und werden zur Integration von Tastern und Schaltern genutzt. Der Gerätetreiber kann beliebig viele IO-Warriors bedienen, wobei die Herstellerbibliotheken den Zugriff auf 16 Controller beschränken. Damit sind jedoch bis zu 800 digitale Schnittstellen in eine Entität integrierbar. Der Treiber ist auf beiden Betiebssystemen verfügbar.

IoMc – Microcontroller mit heterogenen Schnittstellen

IoMc ist eine Baugruppen-Familie zur Ansteuerung digitaler, analoger und serieller Schnittstellen über Ethernet und TCP/IP. Sie wurde am Lehrstuhl FAPS mitentwickelt und ist in verschiedenen Varianten mit einer unterschiedlichen Anzahl Schnittstellen und -typen verfügbar. Die Erstellung des Includ-O-Matic-Treibers erfolgte als Socket-Treiber mit der Programmiersprache Python. Der Gerätetreiber zeichnet sich dadurch aus, dass die im System verfügbaren Microcontroller automatisch detektiert und dynamisch die zur Verfügung stehenden Schnittstellen ermittelt werden. Somit ist der Treiber in der Lage, auch zukünftige Microcontroller mit unterschiedlicher Schnittstellenausstattung bekannter Typen ansteuern zu können, ohne angepasst werden zu müssen. Erst wenn andere Schnittstellentypen hinzukommen wird eine Anpassung des Treibers erforderlich.

10.5 Erweiterung von Python zur Skriptsprache für Rezepte

Die zeitliche und ereignisabhängige Koordination (prinzipiell) aller Komponenten von Plant-O-Matic, und damit von in Facetten und Subbäumen gekapselten Prozessen sowie Peripherie, ist eine elementare Aufgabe zur Realisierung von Steuerungsabläufen. Hierzu wurde eine Lösung erarbeitet, die auf drei Bausteinen beruht.

Mit *ReceiPy* wurde Python um Schlüsselwörter und Hilfsfunktionen erweitert, mit denen ein einfacher rechnerübergreifender Zugriff auf Knoten und Facetten, ein einfaches Registrieren von Callbackprozeduren an Publishern und das Erzeugen von Publishern sowie die sequenzielle und parallele Abarbeitung von Anweisungsblöcken möglich ist. Ziel war dabei eine vereinfachte Rezeptentwicklung. Tabelle 10.1 listet die Erweiterungen auf. Die Wahl der Programmiersprache Python als Grundlage der Skriptsprache ist vorteilhaft, da einerseits ein vollständiger Interpreter und andererseits umfangreiche Funktionsbibliotheken verfügbar sind.

Eine Schwierigkeit besteht darin, dass die Schlüsselwörter nicht zum Sprachumfang von Python gehören und deswegen Rezepte nicht unmittelbar ausführbar sind. Deshalb wurde mit dem *SkriptGenerator* ein Übersetzerbaustein entwickelt, der Rezepte in reine Pythonskripte übersetzt, die dann vom Pythoninterpreter verarbeitet werden können. Mit Hilfe des Generators ist es zudem möglich, zusätzlichen Quellcode einzufügen, der Tests vereinfacht. Rezepte durchlaufen in der Regel verschiedene Entwicklungsphasen. In der ersten Phase (*Debugging*) ist eine Unterstützung bei der Fehlersuche notwendig, indem das Rezept jederzeit gestoppt werden kann und das Setzen und Löschen von Breakpoints sowie ein schrittweises Abarbeiten des Rezepts möglich ist. In der zweiten Phase (*Evaluation*) kann es genügen, Rezepte jederzeit anhalten zu können. In der Anwendungsphase (*Release*) kann es ausreichen, an bestimmten Stellen des Rezepts von außen einen Abbruch vornehmen zu können, wozu der Entwickler mögliche Abbruchstellen durch die Schlüsselwörter *break_if_signaled* und

Schlüsselwörter und Funktionen	Funktion
loc:	Durch Angabe eines Locators erfolgt automatisch die Beschaffung einer Referenz auf lokale oder entfernte Knoten oder Facetten.
createPub(...)	Erzeugen eines Publishers.
destroyPub(...)	Löschen eines Publishers.
regCb(...)	Registrierung einer Callbackprozedur an einem lokalen oder entfernten Publisher.
unregCb(...)	Deregistrierung einer Callbackprozedur.
par:	Parallele Ausführung eines Anweisungsblocks.
atomic:	Anweisungsblock, der nicht unterbrochen werden darf.
stop_if_signaled	Setzen eines Unterbrechungspunkts, an dem die Rezeptausführung angehalten werden kann.
break_if_signaled	Setzen eines Abbruchpunkts, an dem die Rezeptausführung beendet werden kann.

Tabelle 10.1: Schlüsselwörter, um die die Programmiersprache Python erweitert wurde.

stop_if_signaled markieren kann. Ersteres zeigt an, dass an dieser Stelle ein kompletter Abbruch der Ausführung möglich ist, während Zweiteres anzeigt, dass die Ausführung vorläufig unterbrochen und zu einem späteren Zeitpunkt fortgesetzt werden kann. Insbesondere bei zyklischen Rezepten, die sich nicht selbst beenden, ist ein Setzen von Unterbrechungspunkten sinnvoll. Ein weiterer Grund für die Wahl des Übersetzungsmodus liegt in Geschwindigkeitseinbußen von bis zu 50 Prozent, die durch das Einfügen von Debug-Code entstehen und im Labor mit einfachen Zeitmessungen ermittelt wurden. Um den unterschiedlichen Phasen zu entsprechen, verfügt der Generator über die drei Modi *DebugCode, EvaluationCode* und *ReleaseCode*. Je nach Wahl fügt er in den ersten beiden Modi die zusätzliche Funktionalität dem Skript hinzu. Im dritten Modus erfolgt keine Erweiterung, der Code wird mit voller Leistung ausgeführt.

Die Rezeptausführung erfolgt im *ReceiptController*, dem dritten Baustein. Dieser ist in der Lage verschiedene Rezepte lokal zu verwalten und stellt entsprechende Schnittstellen zur Verfügung. Zusätzlich existieren Schnittstellen zur Ausführungskontrolle. Damit ist es auch möglich, einen ReceiptController aus einem Rezept heraus zu steuern, so dass eine hierarchische Verschachtelung von Rezepten möglich ist.

10.6 Materialfluss für verzweigte Transportsysteme

Das generische Verzweigungsmodell des Konzepts zu selbstorganisierenden Stoffflüssen (Kap. 7.4) wurde mit *Matfl-O-Matic* umgesetzt. Hierbei lag der Fokus auf dem hardwareunabhängigen Teil, da bei Stoffflüssen die Kernanforderung der Wandelbarkeit auf logistischer Ebene liegt. Hierzu wurde eine Routerfacette zur dezentralen Flusssteuerung entwickelt, die anhand einer Ankopplung an Net-O-Matic in der Lage ist, selbstständig Routentabellen zu generieren und anhand dieser geeignete Ausgänge für Werkstücke zu ermitteln. Ausfälle im Transportsystem schlagen sich automatisch in den

Routentabellen nieder, so dass zu jedem Zeitpunkt nur vorhandene Routen gewählt werden. Existiert keine Route zum Ziel, kann ein Alternativziel angesteuert werden, etwa eine Notfallausschleuse.

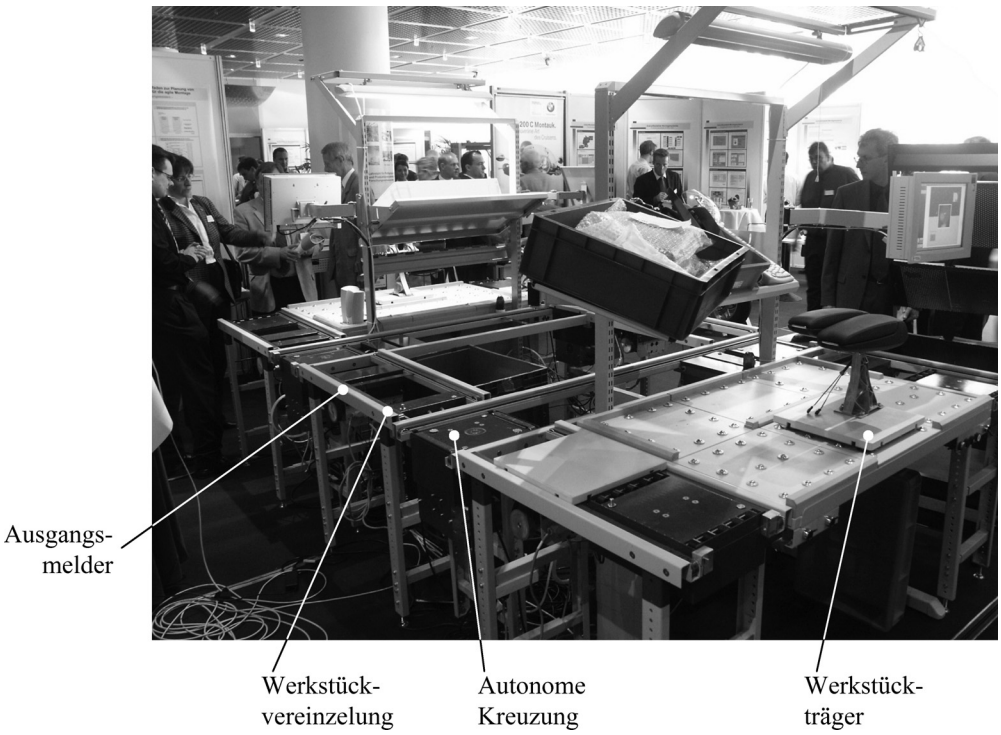


Abbildung 10.10: MAMOS-Montagesystem mit selbstkonfigurierendem Transfer. Die Routentabellen wurden zentral von einem Materialflusssimulator spezifisch für jede Kreuzung berechnet.

Die Validierung der dezentralen Steuerung erfolgte im Labor, wobei Wandlungen in Form simulierter Ausfälle und Umstrukturierungen von Weichen, Transportsegmenten und Arbeitsstationen erfolgten.

Die Umsetzbarkeit der Ansätze auf reale Transportsysteme wurde bereits zu einem früheren Zeitpunkt im Rahmen des BMBF-geförderten Projekts *Marktorientierte Montagestrategien (MAMOS)* am Beispiel eines Reibschlusstransfersystems demonstriert [30, 96]. Abbildung 10.10 zeigt das MAMOS-Montagesystem mit autonom agierendem Transfersystem. Im Gegensatz zur selbstständigen Generierung der Routentabellen auf den Kreuzungen erfolgte dieser Schritt zentral über einen angekoppelten ereignisdiskreten Materialfluss-Simulator (EM-Plant [104]). Dieser war an eine zentrale Anlagenstrukturerkennung gekoppelt und informationstechnisch mit den Kreuzungen verbunden. Ein weiteres Merkmal, die Kreuzung betreffend, bestand in einer bereits hierarchisch, modularisierten Steuerungssoftware und einem ereignisdiskreten Steuerungsparadigma mit lokal begrenztem Publisher-Subscriber-Mechanismus [33].

10.7 Prototypischer Leitstand

Zur Unterstützung der Softwareentwicklungen und -tests einerseits, zur Zustandsvisualisierung, Interaktion und Konfiguration andererseits, wurde ein prototypischer Leitstand entwickelt. Er vereint

eine drei-dimensionale Visualisierung der Anlagenstruktur nahe der Echtzeit mit graphischen Benutzerschnittstellen zur Mensch-Entitäten-Interaktion.

Der Leitstand ist konzeptkonform als Entität ausgelegt, die nachfolgend dargestellten Bestandteile als Facetten realisiert.

3d-Visualisierung der Anlagenstruktur

Die Darstellung der Anlagenstruktur ist durch einen dynamischen Szenenaufbau realisiert, bei dem 3d-Einzelmodelle von Entitäten entsprechend der strukturellen Verknüpfung, die durch Net-O-Matic ermittelt wird, automatisch miteinander zu einem Gesamtmodell verbunden werden. Einzelmodelle unbekannten Typs lädt der Visualisierer direkt von der Entität, so dass bei neuen 3d-Modellen keine nachträglichen Installationen vorgenommen werden müssen.

3d-Einzelmodelle werden räumlich passend zusammengefügt, wozu allerdings eine Bestimmung notwendiger Rotationen und Translationen erforderlich ist. Dies ist allein aus 3d-Daten nicht möglich, weshalb zusätzliche Informationen benötigt werden. Deshalb erfolgt eine Erweiterung der 3d-Modelle um sog. Konnektoren. Dabei handelt es sich um die Beschreibung von Verbindungsflächen an denen andere 3d-Modelle angekoppelt werden können. Die Flächen sind typisiert, so dass nur zwei passende Konnektoren koppelbar sind. Ein Konnektor ist durch ein Tripel aus Koppelpunkt $\vec{c}$ (center), Normalenvektor $\vec{n}$ (normal) und Aufvektor $\vec{u}$ (up) beschrieben. Zwei 3d-Einzelmodelle werden miteinander verbunden, indem das zweite Modell zunächst, um den Winkel α zwischen $\vec{n_1}$ und $\vec{n_2}$ rotiert, stirnseitig zum Ersten ausgerichtet wird. Im Anschluss erfolgt eine Rotation um den Winkel β zwischen $\vec{u_1}$ und $\vec{u_2}$ entlang der Normalenachse, so dass das zweite Modell vertikal korrekt zum Ersten ausgerichtet ist. Durch Translation t um den Differenzvektor zwischen $\vec{c_1}$ und $\vec{c_2}$ wird das zweite Modell zum ersten verschoben. Dieser Mechanismus erlaubt eine beliebige räumliche Ausrichtung der Konnektorflächen. Jede Konnektorfläche entspricht einer realen Verbindungsfläche des physischen Objekts.

Die 3d-Visualisierung wurde konzeptkonform als Facette realisiert, wobei zur Darstellung die 3d-Bibliothek *Ogre* [72] und zur Integration in Plant-O-Matic *PyOgre* [80] verwendet wurde. Abbildung 10.13 zeigt die Darstellung einer exemplarischen Anlagenstruktur. Der erreichte zeitliche Verzug in der Darstellung der aktuellen Anlagenstruktur beträgt im Labor zwei bis drei Sekunden. Dieser Versatz hängt ausschließlich von der Nachbarschaftserkennung und dem Wechsel in die stabile Phase zur Konfiguration des Netzwerks ab (Kap. 6.7).

Zur Nutzung von 3d-Modellen in Ogre müssen diese in dem speziellen Format *OgreXML* vorliegen. Allerdings ist kein CAD-Werkzeug bekannt, das dieses Format unterstützt. Dafür verfügen die meisten dieser Werkzeuge über VRML-2.0 Exportfilter. So wurde ein Kommandozeilenwerkzeug entwickelt, das 3d-Modelle von VRML-2.0 nach OgreXML konvertiert. Der Konverter wurde mit 3d-Modellen aus Pro Engineer [76] und SolidWorks 2005 [98] getestet.

Mensch-Entitäten-Interaktion

Zur Interaktion mit Entitäten wurden exemplarisch graphische Benutzerschnittstellen als Facetten entwickelt, die mit Entitäten über die Kommunikations-Middleware von Plant-O-Matic interagieren.

Zur Simulation einer Auftragseinlastung und Produktvarianten wurde eine manuelle Lösung erarbeitet. Hier wird zunächst von einem Bediener in dem 3d-Modell eine Stationsfolge für ein Werkstück markiert und dieses über Tastendruck in die Anlage eingeschleust. Von da an transportieren die Modul-PCs das Werkstück über die Stationen zum Ziel, wobei bei Strukturänderungen eine automatische Anpassung der Route erfolgt.

Jede Entität ist mit einem RezeptController ausgestattet. Um für sie Rezepte zu erstellen und auf eine oder mehrere Entitäten zu verteilen, wurde eine einfache Entwicklungsumgebung *Script-IDE* implementiert, die in Abbildung 10.11 dargestellt ist. Sie erlaubt zudem das kontrollierte Starten, Stoppen und Zurücksetzen von Skripten sowie das Setzen von Breakpoints oder die schrittweise Ausführung zu Testzwecken. Die Kontrolle kann auf einzelne oder eine Auswahl mehrerer Entitäten ausgeübt werden.

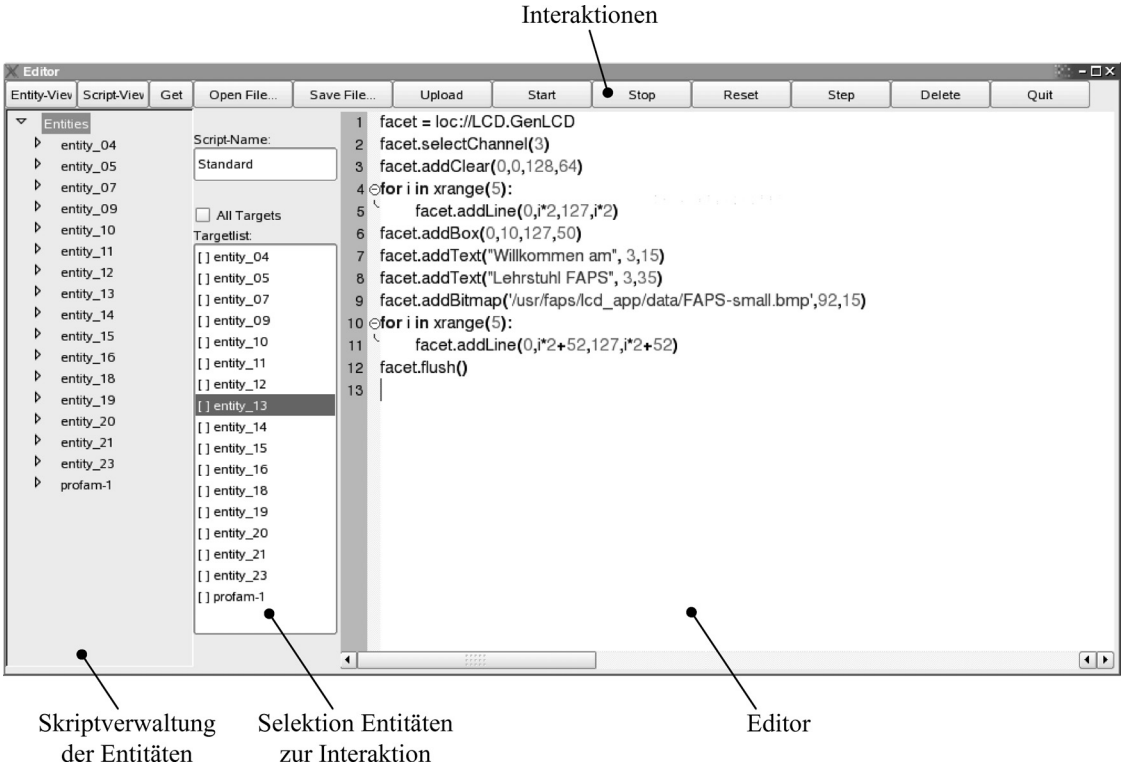


Abbildung 10.11: SkriptEditor zur Erstellung von Skripten und zur Interaktion mit den skriptfähigen Controllern auf Entitäten.

Die graphischen Benutzerschnittstellen nutzen die plattformunabhängige Widgetbibliothek *wxWidgets* [126], die über *wxPython* [125] in Plant-O-Matic integriert ist. So sind die Oberflächen unter Linux und Windows lauffähig. Zudem sind alle Benutzerschnittstellen an den Entitäten hinterlegt und werden durch Auswahl einer Entität im 3d-Modell von dieser geladen. Dies demonstriert die Möglichkeit, beliebige graphische Benutzerschnittstellen auf Entitäten zu hinterlegen. Der Leitstand oder etwa portable Anzeigegeräte müssen vorab nicht über diese Oberflächen verfügen. So ist eine aufwandsarme Integration neuer Entitäten jederzeit möglich.

Eine weitere Form der Mensch-Entitäten-Interaktion besteht über physische Schnittstellen. Hierzu sind Modul-PCs exemplarisch mit Tastern ausgestattet, die als digitale Eingangsfacetten in Plant-O-

Matic integriert sind. Da jede Facette über Rezepte ansprechbar ist, wurden verschiedene Testszenarien als Rezepte realisiert, die auf allen Modul-Steuerungen ausgeführt wurden. Das Ziel lag im Nachweis der Verknüpfbarkeit von Sensorik und Aktorik auch über Rechnergrenzen - und damit Gerätegrenzen - hinweg. Hierzu wurden Taster mit entfernten LCD-Displays und Audioausgaben gekoppelt, so dass der Tastendruck über das Netzwerk an den entsprechenden Facetten Aktionen auslöst. So bestand ein Experiment darin, jede Entität bei drei Tastern für jeweils zwei Ereignisse (Taste gedrückt, Taste nicht mehr gedrückt) zu registrieren, so dass zwischen 23 Modul-PCs insgesamt ca. 3000 virtuelle Verbindungen erfolgreich geschalten wurden. Diese Verbindungen hielten auch manuellen Wandlungen der Systemstruktur stand, wobei Rechner umverkabelt wurden.

Abbildung 10.12: Testlabor im Betrieb. Die Wandprojektion zeigt die emulierte Anlagenstruktur.

10.8 Das Labor im Betrieb

Mit dem bisher beschriebenen Testlabor für verteilte Steuerungssysteme kann das kooperative Zusammenwirken dezentraler Steuerungssysteme im wandelbaren Umfeld entwickelt und validiert werden. Dies geschieht dadurch, dass Ausschnitte von Produktionsanlagen oder Produktionslinien nachgestellt werden. Ebenso ist der hierarchische Nachbau einer Zelle oder eines Gerätes denkbar.

Abbildung 10.12 zeigt das Labor im Einsatz bei der Nachstellung eines Produktionssystems mit drei Arbeitsstationen, die im Nebenschluss über ein Reibschlusstransfersystem an einen Hauptring gekoppelt sind. Die Steuerrechner der Arbeitsstationen und Weichen sind an der Wand aufgesteckt und über Netzwerkkabel entlang der Transfersegmente miteinander verbunden. Die Wandprojektion visualisiert dabei das nachgestellte System. Sie wird von dem Leitstandsrechner in Echtzeit erzeugt.

Abbildungen 10.14 zeigt eine Wandlungssequenz des gerade beschriebenen Systems, in dem ausgehend von einem Hauptringtransfer mit zirkulärem Materialfluss zunächst eine Arbeitsstation im

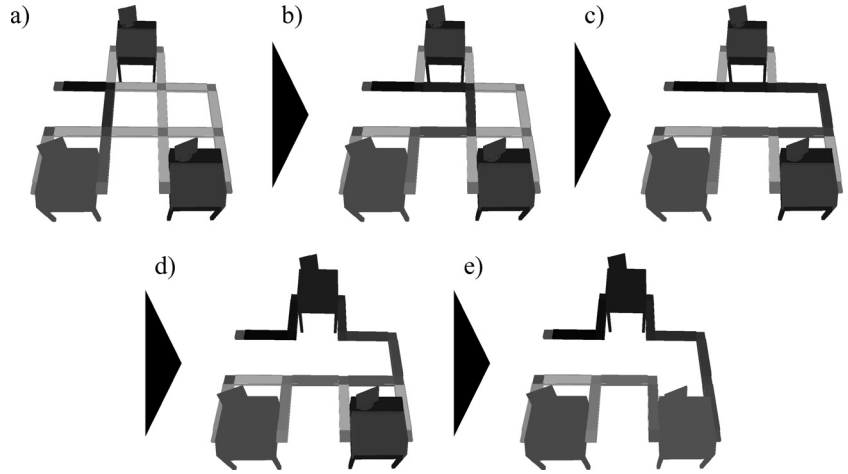


Abbildung 10.13: Selbstständige Anpassung des Materialflusses bei schrittweisem Ausfall von Transportsegmenten. Nachstellung im Labor.

Nebenschluss angekoppelt wird. Nach Ankopplung zweier weiterer Arbeitsstationen ist das Produktionssystem vollständig ausgebaut. In einem letzten Schritt wird schließlich der Komplettausfall eines Nebenschlusses simuliert, indem zwei Weichen und eine Arbeitsstation abgeschaltet werden.

Abbildung 10.13 zeigt schließlich die Visualisierung der automatischen Wegfindung von Matfl-O-Matic. Rot markiert sind die Entitäten, über die Werkstücke zwingend geroutet werden müssen. Grün dargestellt ist der kürzeste Weg, dies zu erreichen. Durch Änderung der Netzwerkverkabelung wird eine strukturelle Änderung des Transfersystems emuliert. Sobald die Nachbarschaftserkennung wieder in der stabilen Phase angelangt ist, passt sich der Materialfluss automatisch an.

Die gezeigten Darstellungen verdeutlichen die Bedeutung des Labors für Entwicklung und Validierung verteilter Steuerungssysteme aber auch für die Demonstration und Nachvollziehbarkeit der „inneren“ Vorgänge von Produktionssystemen.

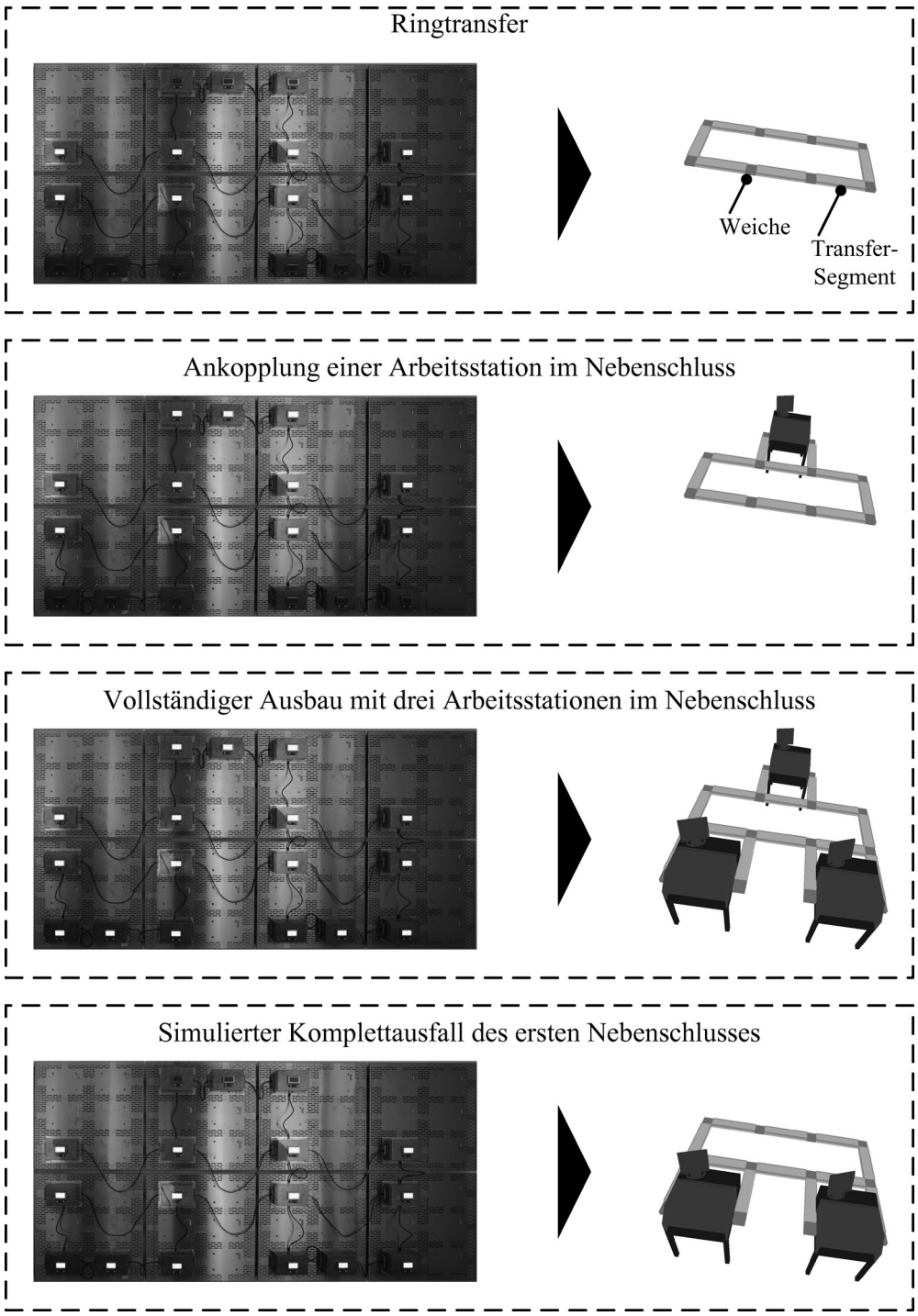


Abbildung 10.14: Wandlungssequenz einer Produktionslinie. Die Modul-PCs emulieren Weichen und Arbeitsstationen, Netzwerkkabel dagegen Transfersegmente. Ausfälle werden simuliert, indem Modul-PCs abgeschaltet werden.

Kapitel 11

Zusammenfassung

Die fortwährende Anpassbarkeit der Produktion an Veränderungen des Umfelds gilt als ein Schlüssel zur Steigerung der Wettbewerbsfähigkeit von Unternehmen im Spannungsfeld turbulenter, globaler Märkte. Produktionsanlagen sind hierfür derart mit Wandlungspotenzial auszustatten, dass eine effiziente Reaktion auf externe wie interne Wandlungstreiber erfolgen kann. Möglichkeiten finden sich in technischen wie organisatorischen Bereichen der Produktion.

Die vorliegende Arbeit ist auf die technische Seite der Wandlungsfähigkeit ausgerichtet und fasst in der Ausgangssituation eine Produktionsanlage als geschachtelten Komplex verschiedenartiger Haupt- und Nebenflusssysteme auf, durch die Stoffe, Energien, Signale und Informationen fließen und transformiert werden. Die Modularisierung einer Produktionsanlage, der wichtigste technische Ansatzpunkt im Themenfeld der Wandlungsfähigkeit, kann dadurch als erfolgreich definiert werden, wenn alle Flussfunktionen von Haupt- und Nebenflüssen zur Erfüllung einer spezifischen Aufgabe in einem einzelnen (hierarchischen) Systemmodul zusammengefasst sind und jede Teilaufgabe der Gesamtaufgabe „Produktion“ durch ein eigenes Systemmodul gelöst wird.

Die modulare Gestaltung von Produktionsanlagen von mechanischer und energetischer Seite ist in Teilen bereits Stand der Technik, auch wenn ein durchgängiger Ansatz über alle Ebenen nicht auszumachen war. Unbehandelt sind dagegen die steuerungs- und softwaretechnischen Aspekte und die damit verbundenen Potenziale durch Selbstorganisation und Dezentralisierung. Diesen widmet sich die Arbeit.

In einer Analyse basierend auf der allgemeinen Systemtheorie wurden für modulare Produktionsanlagen wesentliche Anforderungen erarbeitet. Zu ihnen zählen die Forderung nach einer durchgängigen Modularisierung mit nach innen gerichtetem Wirkbereich der Module sowie eine ganzheitliche Betrachtung aller Schnittstellen. Dabei muss das Ziel in einer abgeschlossenen, minimalen Menge verschiedener, auch geschachtelter, Schnittstellentypen liegen. Softwaresysteme sind ausschließlich aus Softwaremodulen hierarchisch zusammenzusetzen. Der Übergang zwischen virtueller und physischer Seite muss durch Zwillingsbausteine realisiert werden. Dabei handelt es sich um untrennbare Paare aus physischem Modul und Softwaremodul. Ein weiteres Ziel stellt ein verteilter Einsatz von Steuerungstechnik dar. Idealerweise wird jedes Systemmodul damit ausgestattet. Softwaresysteme sind – soweit möglich – nicht für Systemmodule und dem spezifischen Einsatz für deren Steuerungstechnik zu gestalten. Stattdessen muss bewusst eine Trennung herbeigeführt werden, die es erlaubt, Software auch in weniger dezentralen Konstellationen betreiben zu können. Dieser Ansatz trägt zur Erhöhung der Wiederverwendbarkeit von Softwaremodulen bei. In diesem Zusammenhang ist ein weiterer wichtiger Aspekt die Trennung der Steuerungen – Logistische eingeschlossen – in Ablauf- und Ausführungssteuerungen.

Diesen Anforderungen folgend wurde ein formales, theoretisches Systemkonzept für wandelbare Produktionsanlagen entwickelt, welches ein Fundament für alle weiteren Arbeitsschritte bildet. Auf diesem wurde eine wandelbare Systemlandschaft auf Basis eines Ebenenmodells konzipiert. Das 6-Ebenen$^+$-Modell begreift Produktionsanlagen und ihre Bestandteile als hierarchisch beliebig rekursiv geschachtelte Systeme, deren kleinste Elemente Sensoren und Aktoren bilden. Für alle Ebenen wurden Wandlungsmöglichkeiten aufgezeigt.

Die Systemlandschaft integriert dezentrale, homogene Steuerungstechnik und Softwaresysteme, die auf Basis einer Kommunikations-Middleware und eines übergreifenden hierarchischen Adressierungsschemas miteinander interagieren und kooperativ Produktionsaufgaben koordinieren. Alle Softwaresysteme sind hierarchisch aus Softwaremodulen aufgebaut und über standardisierte Softwareschnittstellen miteinander – auch rechnerübergreifend – verbunden. Sie umfassen Steuerungen, selbstadaptierende generische Informationsdienste und andere Softwareapplikationen, die für eine wandelbare Produktionsanlage erforderlich sind.

Zur Erstellung dieser Softwaresysteme wurde auf Grundlage des Softwareparadigmas „Componentware“ und der objektorientierten Programmierung eine auf Wandelbarkeit ausgerichtete Softwarearchitektur konzipiert. In dieser werden Elemente, Interaktionsformen sowie Verhalten definiert und Möglichkeiten zur Erzeugung komplexer Softwaresysteme beschrieben. Die Architektur wurde mit der Programmiersprache *Python* für *Linux*- und *Windowsbetriebssysteme* in eine Referenzimplementierung umgesetzt, die unter anderem auch eine eigenentwickelte *TCP/IP-basierte* Kommunikations-Middleware und das hierarchische Adressierungsschema beinhaltet.

Aus Systemmodulen aufgebaute Produktionsanlagen sind einfach umbaubar, weshalb Wandlungen schnell durchgeführt werden können. Indem Module mit eigener Steuerungstechnik und Steuerungssoftware ausgestattet werden, wie es die wandelbare Systemlandschaft vorsieht, ist es möglich, Produktionsabläufe selbstständig und ohne manuelle Eingriffe an Veränderungen anzupassen. Wichtige Aufgaben der Produktionsabläufe sind unter anderem die Organisation von Stoffflüssen, die Verwaltung von Produktionsaufträgen und ihre kontrollierte Freigabe in die Produktionssysteme, die Bereitstellung von Stoffen an den Arbeitsstationen und die Kompensation von Systemstörungen.

Grundlagen hierfür sind in allen Fällen Kenntnisse über die Anwesenheit von Systemmodulen, ihre strukturellen Beziehungen untereinander, ihre Positionen in der Anlagenhierarchie, ihre prozesstechnischen Fertigkeiten zur Bearbeitung von Erzeugnissen und ein konfiguriertes Kommunikationsnetz. Auf Basis einer vorausgesetzten Isomorphie zwischen Systemstruktur und Struktur des Kommunikationsnetzes wurden mehrere dezentrale Verfahren entwickelt, mit denen Systemmodule diese Informationen ermitteln. Jedes Systemmodul erhält dadurch totale Kenntnis über den Aufbau der Produktionsanlage. Durch eine Erweiterung der Verfahren ist es jedem einzelnen Systemmodul möglich, für *alle* Elemente Netzwerkadressen deterministisch zu berechnen. Damit können eigene Netzwerkschnittstellen konfiguriert und andere Systemmodule erfolgreich adressiert werden. Alle Verfahren wurden mit der Programmiersprache *Python* für *Linux*- und *Windowsbetriebssysteme* auf Basis von *TCP/IP* implementiert.

Mit Kenntnis der Anlagenstruktur, der Arbeitsstationen und ihren Fertigkeiten sind die Flüsse für Werkstücke und andere Stoffe automatisiert konfigurierbar. Hierzu wurden verschiedene Flussstrategien entwickelt, die neben einem zielgerichteten Transfer von Gegenständen im Falle von Werkstücken auch zur Kompensation von Störungen fähig sind. Für gestörte, pausierende, in Wartung

oder Instandsetzung befindliche Arbeitsstationen werden automatisch Alternativen ausgewählt, sofern Redundanzen verfügbar sind. Zudem werden mit Hilfe der selbstständigen Strukturerkennung neue Arbeitsstationen und Veränderungen in den Transfernetzen automatisch erkannt und in die Flussorganisation integriert. Für den Aufbau realer autonomer Steuerungen für Kreuzungen, Weichen und Transportfahrzeuge wurde eine Ablaufsteuerung auf Grundlage der Softwarearchitektur entworfen, die ohne manuelle Eingriffe auf diese drei verschiedenen Verzweigungsarten angepasst werden kann, allein durch eine geeignete Auswahl anzukoppelnder Sensorik. Diese Ablaufsteuerung dient als ein Beispiel für den Aufbau einer flexiblen Steuerung auf Basis von Softwaremodulen.

Selbstorganisation kann auf Produktionssysteme ausgedehnt werden, indem Produktionsaufträge an diese dynamisch nach ihren Fertigkeiten verteilt werden. Änderungen in Produktionssystemen werden erkannt, so dass nur fertigbare Aufträge eingelastet werden. Eine weitere Aufgabe, die automatisiert gelöst werden kann, ist die Zuordnung von Stoffen zu Produktionssystemen und Arbeitsstationen, indem Erstere mit Systemeingangs- und Systemausgangslagern und Zweitere mit Eingangslagern ausgestattet werden. Es wurde eine dezentrale generische Ablaufsteuerung auf Basis der Softwarearchitektur konzipiert, die aus einer Verwaltung für Produktionsaufträge und mehreren Lagerverwaltungen, in Abhängigkeit des Systemaufbaus, besteht. Sie kann ohne manuelle Eingriffe in verschiedenen Produktionssystemen eingesetzt werden. Als Prinzip der Fertigungssteuerung kommt CONWIP (Constant Work in Process) zusammen mit dem PULL-Prinzip zum Einsatz. Mehrere Produktionssysteme dieser Art ergeben eine Produktionsanlage. Mit Eingangs- und Ausgangslagern ausgestattet, und mit Hilfe einer geeigneten Schnittstelle zur Materialwirtschaft und Auftragsfreigabe der Produktionsplanung und -steuerung, ist schließlich die gesamte Anlage selbstorganisierbar und über die Strukturerkennung zur Betriebszeit wandelbar.

Einen letzten Schwerpunkt der Arbeit bilden selbstadaptierende generische Informationsdienste und allgemeine Softwaresysteme. Diese übernehmen verschiedene Aufgaben der Informationsverarbeitung, wie Betriebsdatenerfassung, Informationsverwaltung, Rezeptverwaltung, Produktionsüberwachung, Produkt-Tracking, Qualitätssicherung, Diagnose, Mensch-Maschine-Schnittstellen, usw. Für datenbankgestützte Informationsdienste wurde ein generischer Ansatz für den redundanten Betrieb entwickelt. Dienste bilden nach Art und Position in der Anlagenhierarchie Gruppen und bestimmen darin selbstständig Master und Slaves zur Absicherung gegen Ausfälle. Die Gruppen kompensieren Wandlungen, Partitionierungen und Verschmelzungen von Teilsystemen, wie sie durch Anlagenumbau, -erweiterung, -abbau oder Störungen entstehen können. Über Synchronisationsmechanismen werden Daten konsistent gehalten. Für alle anderen Softwaresysteme wurde aufgezeigt, wie für sie eine wandelbare Gestaltung erzielt werden kann.

Mit der vorgelegten Arbeit konnte gezeigt werden, dass eine Unterstützung der Wandlungsfähigkeit durch die Berücksichtigung und geeignete Gestaltung der Steuerungstechnik und Softwaresysteme sowie mit Hilfe von Verfahren zur Selbsterkennung, -konfiguration und -organisation, effektiv möglich ist.

Summary

The continuous adaptability of production to changes, which are caused by internal as well as external impacts, is a primary key for increasing the competitiveness of producers within turbulent, global markets. Therefore production systems have to be set up with potentials of changeability, covering aspects of mechanics, energetics, signals, controls and software, so that efficent reactions are feasible.

The objective of this thesis was the support of production systems' changeability by holistic consideration of the modularization paradigm, leading to greater flexibility and thus increased efficiency in doing modifications. Since the modularization of physical aspects is already well known, the focus was on control systems and their corresponding software.

Based on the system theory and considering production as a nested complex of different flow systems, which are passed by materials, energy, signals and information, a theoretical analysis concentrated on controls and their software was performed at first. Derived from this study a theoretical concept of holistic changeable production systems was constructed and design rules were concluded. From this theoretical approach a changeable system environment was derived, which comprehends production as a multilevel hierarchized system. Changeable controls and software systems were applied on all levels of the hierarchy and interconnected horizontally and vertically in order to perform tasks cooperatively.

For developing those software systems a special software architecture was conceived, adjusted to the aspects of changeability. This software architecture is strictly based on the results of the former theoretical analysis. It relies on the paradigms of componentware and object-oriented programming. The architecture primarily aims the assembly of software by connecting prepared modules, which are part of certain construction kits. Software executed on controls is thought to interact cooperatively via an adaptable communication middleware, which fits seamless into the architecture.

Decentralized mechanisms for self-detection, self-configuration and self-organisation have an important influence for the reduction of planning, development and setup times by simultaneous assurance of effective workflows. Therefore solutions for the decentralized detection of system modules and their horizontal as well as their vertical interconnections were developed. Based on structural information the autonomous configuration of the communication network within a production system is possible, thus modified systems can be configured without manual interventions.

By means of these fundamentals, material flows can be self-organized effectively. Therefore solutions were developed, including a generic control for turnouts, crossings and transport vehicles as well as efficient decentralized strategies for material forwarding. Those strategies implicitly imply mechanisms for the compensation of failures within the transfer systems or working stations. An expansion of self-organisation to single production systems and further more to groups of them were treated afterwards. Here, the primary objective was the generation of continuous material flows while breakdowns and failures at operation time get compensated.

For operating production systems usually software systems are needed for information management on the one hand and performing services on the other hand. Those systems may interact and thus might be affected by modifications. Therefore concepts for self-adapting information services are introduced and solutions for designing general software systems are suggested and discussed finally.

Literaturverzeichnis

[1] AMTEC ROBOTICS. Power cube. http://www.powercube.de, Stand: 18.05.2007.

[2] D. ARNOLD, H. ISERMANN, A. KUHN UND H. TEMPELMEIER. *Handbuch Logistik*. Springer Verlag, Berlin, 2002.

[3] M. BAUMEISTER. *Fabrikplanung im turbulenten Umfeld - Methodik zur Zielplanung einer Fabrik unter Berücksichtigung eines turbulenten Unternehmensumfeldes und der übergeordneten Unternehmensziele*. Dissertation, TU Karlsruhe, 2002.

[4] A. BEAULIEU. *Einführung in SQL*. O'Reilly, Beijing, 2006.

[5] P. A. BERNSTEIN. Middleware - An Architecture for Distributed System Services. Technischer Bericht CRL 93/6, Digital Equipment Corporation, 1993.

[6] A. D. BIRELLI UND B. J. NELSON. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1), 1984.

[7] G. BOOCH. *Objektorientierte Analyse und Design*. Addison-Wesley, Bonn, 1997.

[8] M. BÄR UND S. WIRTH. *Flexible Fertigungssysteme – Gestaltung und Anwendung in der Teilefertigung*. Hüthig Buch Verlag, Heidelberg, 1989.

[9] K. BRANTNER. *Adaptierbares Leitsteuerungssystem für flexible Produktionssysteme*. Dissertation, Universität Stuttgart, 1993.

[10] H. BRENNER, R. ZARNEKOW UND H. WITTIG. *Intelligent Software Agents: Foundations and Applications*. Springer Verlag, 1998.

[11] B. BRINZER, J. PRIESE, P. KLEMM UND G. PRITSCHOW. Neue Ansätze zur dezentralen Produktionsregelung. In *wt Werkstatttechnik online*, Jahrgang 95 2005.

[12] BUSYBOX. http://www.busybox.net, Stand: 18.05.2007.

[13] V. CICIRELLO UND S. SMITH. Ant Colony Control for Autonomous Decentralized Shop Floor Routing. In *Fifth International Symposium on Autonomous Decentralized Systems*.

[14] V. CLAUS UND A. SCHWILL. *Duden Informatik*. Dudenverlag, Mannheim, 2001.

[15] CODE MERCENARIES. http://www.codemercs.com/D_index.html, Stand: 18.05.2007.

[16] A. W. COLOMBO, R. SCHOOP UND R. NEUBERT. An Agent-Based Intelligent Control Platform for Industrial Holonic Manufacturing Systems. *IEEE Transactions on Industrial Electronics*, 53(1), 2006.

[17] W. T. COUNCILL UND G. T. HEINEMAN. *Component-Based Software Engineering*. Addison-Wesley, New York, 2001.

[18] W. DANGLMAIER. *Fertigungsplanung. Planung von Aufbau und Ablauf der Fertigung - Grundlagen, Algorithmen und Beispiele*. Springer Verlag, 1999.

[19] C. DANIEL. *Dynamisches Konfigurieren von Steuerungssoftware für offene Systeme*. Dissertation, Universität Stuttgart, 1995.

[20] R. DIESTEL. *Graphentheorie*. Springer Verlag, Heidelberg, 2006.

[21] S. DÜRRSCHMIDT. *Planung und Betrieb wandlungsfähiger Logistiksysteme in der variantenreichen Serienproduktion*. Dissertation, TU München, 2001.

[22] W. EMMRICH. Software Engineering and Middleware: A Roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, 2000.

[23] W. EVERSHEIM. *Organisation in der Produktionstechnik*, Band 4. VDI, 1989.

[24] W. EVERSHEIM UND G. SCHUH. *Betriebshütte – Produktion und Management*. Springer Verlag, Berlin, 1996.

[25] W. EVERSHEIM UND G. SCHUH. *Gestaltung von Produktionssystemen*. Springer Verlag, 1999.

[26] K. FELDMANN, S. JUNKER UND W. WOLF. Innovative Mechatronic Devices for Flexible Automated Assembly Systems, 2005.

[27] K. FELDMANN UND S. SLAMA. Highly flexible Assembly - Scope and Justification. *Annals of the CIRP*, 50(2), 2001.

[28] K. FELDMANN, S. SLAMA UND S. JUNKER. Umfrage zu modularen Montagesystemen 2000 bei ausgewählten, repräsentativen Herstellern von Montagesystemen. Unveröffentlichte Marktstudie, Erlangen.

[29] K. FELDMANN, M. WEBER UND W. WOLF. Design of a Theoretical Holistic System Model as Base of Construction Kits for Building Plug&Produce-able Modular Production Systems. *Production Engineering XIV/? (akzeptiert)*, 2007.

[30] K. FELDMANN, M. WEBER UND W. WOLF. Performance Analysis of Distributed Strategies for Material Flow Control of Modular Production Systems Based on Discrete Event Simulation. *IEEE International Conference on Industrial Informatics (INDIN)*, 2004.

[31] K. FELDMANN, M. WEBER UND W. WOLF. Decentralized Structure Recognition and Automated Network Configuration for Plug&Produce-able Modular Assembly Systems. *Production Engineering*, XIII(2), 2006.

[32] K. FELDMANN, W. WOLF UND M. WEBER. Design of a Formal Model for the Specification of Agent Platforms Based on Plug&Produce-able Production Systems. *Production Engineering XIV/? (akzeptiert)*, 2007.

[33] K. FELDMANN, W. WOLF UND M. WEBER. Development of an Open, Event-Based and Platform Independent Architecture for Distributed and Intelligent Control Systems. *IEEE International Conference on Industrial Informatics (INDIN)*, 2004.

[34] K. FELDMANN, W. WOLF UND M. WEBER. Self Organizing Material Flow for Plug&Produce-able Modular Assembly Systems. *Production Engineering*, XIII(2), 2006.

[35] O. FRAGER. *Durchgängige Programmierung von Fertigungszellen*. Dissertation, Universität Stuttgart, 1995.

[36] E. GOTTSCHALK (Hrsg.). *Rechnergestützte Produktionsplanung und -steuerung*. Technik Verlag, 1989.

[37] F. GRIFFEL. *Componentware – Konzepte und Techniken eines Softwareparadigmas*. dpunkt Verlag, Heidelberg, 1998.

[38] R. HACKSTEIN. *Produktionsplanung und -steuerung (PPS) - Ein Handbuch für die Betriebspraxis*. Springer Verlag, 1989.

[39] H. O. HÄBERLE, K. RIEGER UND H. RÖDER. *Fachkunde Büro- und Informationselektronik mit Radio-, Fernseh- und Medientechnik*. Europa-Lehrmittel, Hann-Gruiten, 2000.

[40] R. G. HERRTWICH UND G. HOMMEL. *Nebenläufige Programme*. Springer Verlag, 1994.

[41] T. HILDEBRAND. *Theoretische Grundlagen der bausteinbasierten, technischen Gestaltung wandlungsfähiger Fabrikstrukturen nach dem PLUG+PRODUCE Prinzip*. Dissertation, TU Chemnitz, 2005.

[42] T. HILDEBRAND, K. MÄDING UND U. GÜNTHER. *PLUG+PRODUCE Gestaltungsstrategien für die wandlungsfähige Fabrik*. IBF, 2005.

[43] HMS INDUSTRIAL NETWORKS. Feldbusse - Schlüsseltechnologie für die Automatisierung. http://www.feldbusse.de, Stand: 18.05.2007.

[44] O. HOLTHAUS UND C. RAJENDRAN. New Dispatching Rules for Scheduling in a Job Shop - An Experimental Study. *International Journal of Advanced Manufacturing Technology*, 13(2), 1997.

[45] W. J. HOPP UND M. L. SPEARMAN. *Factory physics*. Erwin McGraw-Hill, Boston, 1996.

[46] R. HÄRING. *Die layoutflexible Fabrik - Grundlagen und technische Gestaltung mittels ortsveränderlicher Fertigungseinheiten*. Dissertation, GMU Duisburg, 1997.

[47] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS INC.. IEEE Standards Association. http://standards.ieee.org/regauth/oui/index.shtml, Stand: 18.05.2007.

[48] R. JÜNEMANN UND T. SCHMIDT. *Materialflusssysteme - Systemtechnische Grundlagen*. 2000, Springer Verlag.

[49] S. KIRCHNER, R. WINKLER UND E. WESTKÄMPER. Unternehmensstudie zur Wandlungsfähigkeit von Unternehmen. In *wt Werkstatttechnik online*, Jahrgang 93 2003.

[50] Y. KOREN, U. HEISEL, F. JOVANE, T. MORIWAKI, G. PRITSCHOW, G. ULSOY UND H. VAN BRUSSEL. Reconfigurable Manufacturing Systems. *Annals of the CIRP*, 48, 1999.

[51] E. KREIS, W. EVERSHEIM UND T. LANGE-STALINSKI. Die mobile Fabrik - Ein Mittel zur flexiblen Reaktion auf veränderte Marktbedingungen. In *Tagungsunterlagen Karlsruher Arbeitsgespräche*, 2002.

[52] J. F. KUROSE UND K. W. ROSS. *Computernetze. Ein Top-Down-Ansatz mit Schwerpunkt Internet*. Pearson Studium, Berlin, 2004.

[53] L. LAMPORT. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7), 1978.

[54] T. LANGE-STALINSKI. *Methodik zur Gestaltung und Bewertung mobiler Produktionssysteme*. Dissertation, RWTH Aachen, 2003.

[55] J. L. M. LASTRA. UND A. W. COLOMBO. Engineering framework for agent-based manufacturing control. *Engineering Applications of Artificial Intelligence*, 19, 2006.

[56] H. LÖDDING. *Verfahren der Fertigungssteuerung*. Springer Verlag, Berlin, 2005.

[57] P. LEITAO, A. W. COLOMBO UND F. K. RESTIVO. ADACOR: A Collaborative Production Automation and Control Architecture. *IEEE Intelligent Systems*, 20(1), 2005.

[58] J. LEWEK. *Adaptierbares Informationssystem zur Erstellung baukastenbasierter Fertigungseinrichtungen*. Dissertation, Universität Stuttgart, 2005.

[59] S. LIBERT UND U. SONDHOF. Dezentrale Steuerung für Materialflusssysteme am Beispiel von Stückgutförder- und sortieranlagen. In *Tagungsband der SPS/IPS/DRIVES, 22.-24. November 2005*.

[60] LINUX KERNEL ORGANIZATION. http://www.kernel.org, Stand: 18.05.2007.

[61] H. LUCZAK, W. EVERSHEIM UND M. SCHOTTEN. *Produktionsplanung- und -steuerung*. Springer Verlag, Berlin, 1998.

[62] R. LUTZ. *Softwaretechnik für maschinennahe Steuerungsfunktionen bei Fertigungseinrichtungen*. Dissertation, Universität Stuttgart, 1999.

[63] F. MATTERN. *Verteilte Basisalgorithmen*. Springer Verlag, 1989.

[64] F. P. MATURANA UND D. H. NORRIE. Multiagent mediator architecture for distributed manufacturing. *Journal of Intelligent Manufacturing*, 7, 1996.

[65] M. D. MCILROY. Mass Produced Software Components. In *Software Engineering, Report on a conference sponsored by the NATO Science Committee*, 1968.

[66] B. MEYER. *Object-Oriented Software Construction*. Prentice Hall PTR, 1997.

[67] MICROSOFT. .NET Framework. http://www.microsoft.com/germany/msdn/netframework/default.mspx, Stand: 18.05.2007.

[68] R. H. MORALES. *Systematik der Wandlungsfähigkeit in der Fabrikplanung*. Dissertation, Universität Hannover, 2002.

[69] O. NIERSTRASZ UND D. TSICHRITZIS. *Object-oriented software composition*. Prentice Hall International, Hertfordshire, 1995.

[70] D. NOFEN, J. H. KLUSSMANN, F. LÖLLMANN UND H.-P. WIENDAHL. Regelkreisbasierte Wandlungsprozesse – Wandlungsfähigkeit auf Basis modularer Fabrikstrukturen. *wt Werkstatttechnik online*, 93, 2003.

[71] G. NUSSER. *Automatische Generierung von Softwarekomponenten auf der Basis von Metamodellen und XML*. Dissertation, Universität Tübingen, 2005.

[72] OGRE. http://www.ogre3d.org, Stand: 18.05.2007.

[73] OPC FOUNDATION. http://www.opceurope.org, Stand: 18.05.2007.

[74] R. ORFALI, D. HARKEY UND J. EDWARDS. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, New York, 1995.

[75] R. OSE. *Elektrotechnik für Ingenieure, Bd.1, Grundlagen*. Fachbuchverlag Leipzig, 2001.

[76] PARAMETRIC TECHNOLOGY. http://www.ptc.com/appserver/mkt/products/home.jsp?k=403, Stand: 18.05.2007.

[77] R. PESENTI UND L. CASTELLI. Scheduling in a Realistic Environment Using Autonomous Agents: A Simulation Study. In *Proceedings of Agent Based Simulation II Workshop*, 2001.

[78] F. POSSEL-DÖLKEN. *Projektierbares Multiagentensystem für die Ablaufsteuerung in der flexibel automatisierten Fertigung*. Dissertation, RWTH, Aachen, 2006.

[79] G. PRITSCHOW, G. SPUR, M. WECK UND A. PAULS. *Leit- und Steuerungstechnik in flexiblen Produktionsanlagen*. Carl Hanser Verlag, 1991.

[80] PYOGRE. http://www.ogre3d.org/wiki/index.php/PyOgre, Stand: 18.05.2007.

[81] PYTHON SOFTWARE FOUNDATION. http://www.python.org, Stand: 18.05.2007.

[82] RABBIT SEMICONDUCTOR. Rabbit 3000 microprocessors.

[83] G. REINHART. Wandlungsfähige Fabrikgestaltung. *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb*, 97, 2002.

[84] A. RITTER. *Ein Multi-Agenten-System für mobile Einrichtungen in Produktionssystemen*. Dissertation, Universität Stuttgart, 2003.

[85] G. ROPOHL. *Flexible Fertigungssysteme*. Dissertation, TU Stuttgart, 1971.

[86] G. ROPOHL. *Allgemeine Technologie - Eine Systemtheorie der Technik*. Carl Hanser Verlag, 1999.

[87] S. J. ROSENSCHEIN, R. A. WILSO UND F. C. KEIL. *Encyclopedia of Cognitive Sciences*, chapter Intelligent Agent Architecture. MIT Press, 1999.

[88] A. SAAD, K. KAWAMURA UND G. BISWAS. Performance Evaluation of Contract Net-based Heterarchical Scheduling for Flexible Manufacturing Systems. *Intelligent Automation and Soft Computing*, 3(3), 1997.

[89] A.-W. SCHEER. *CIM Computer Integrated Manufacturing. Der computergesteuerte Industriebetrieb*. Springer Verlag, Berlin, 1990.

[90] M. SCHENK UND R. SEELMANN-EGGEBERT. *Moving into Mass Customization*, chapter Mass Customization Facing Logistics Challenges. Springer, Berlin, 2002.

[91] M. SCHENK UND S. WIRTH. *Fabrikplanung und Fabrikbetrieb*. Springer Verlag, 2004.

[92] H. W. SCHÜSSLER. *Digitale Signalverarbeitung 1 - Analyse diskreter Signale und Systeme*. Springer Verlag, Berlin, 1994.

[93] G. SCHUH UND H.-P. WIENDAHL. *Komplexität und Agilität*. Springer, Berlin, 1997.

[94] M. SESTERHENN. *Bewertungssystematik zur Gestaltung struktur- und betriebsvariabler Produktionssysteme*. Dissertation, RWTH Aachen, 2003.

[95] S. SILBERNAGL UND A. DESPOPOULOS. *Taschenatlas der Physiologie*. Thieme Verlag, 1991.

[96] S. SLAMA. *Effizienzsteigerung in der Montage durch marktorientierte Montagestrukturen und erweiterte Mitarbeiterkompetenz*. Dissertation, FAU Erlangen, 2004.

[97] R. SMITH. The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. *IEEE Transaction on Computers*, 12, 1980.

[98] SOLIDWORKS DEUTSCHLAND. http://www.solidworks.de, Stand: 18.05.2007.

[99] M. L. SPEARMAN UND M. A. ZAZANIS. Push and pull production systems: Issues and comparisons. *Operations Research*, 40(3), 1992.

[100] W. SPERLING. *Modulare Systemplattformen für offene Steuerungssysteme*. Dissertation, Universität Stuttgart, 1999.

[101] C. SZYPERSKI. *Component Software. Beyond Object-Oriented Programming*. ACM Press, New York, 2002.

[102] A. S. TANENBAUM. *Computer networks*. Prentice Hall, 2003.

[103] A. S. TANENBAUM UND M. VAN STEEN. *Verteilte Systeme*. Pearson Studium, 2003.

[104] UGS TECNOMATIX. http://www.ugsplm.de/produkte/tecnomatix, Stand: 18.05.2007.

[105] THE CTYPES PACKAGE. http://python.net/crew/theller/ctypes, Stand: 18.05.2007.

[106] THE OBJECT MANAGEMENT GROUP (OMG). CORBA. http://www.omg.org, Stand: 18.05.2007.

[107] INTERNATIONAL TELECOMMUNICATION UNION. ITU-T Recommendation X.667. http://www.itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf, Stand: 18.05.2007.

[108] J. M. USHER UND Y.-C. WANG. Multiagent mediator architecture for distributed manufacturing. *Journal of Intelligent Manufacturing*, 7, 1996.

[109] VDI 2815. *Begriffe für die Produktionsplanung und -steuerung: Betriebsmittel*. Beuth Verlag, 1978.

[110] WDMF. Wandlungsfähigkeit durch modulare Fabrikstrukturen. http://www.wdmf.de/forschungsprojekt.html, Stand: 18.05.2007.

[111] E. WESTKÄMPER. Die Wandlungsfähigkeit von Unternehmen. *Werkstatttechnik*, 19(9/4), 1999.

[112] E. WESTKÄMPER, E. ZAHN, P. BALVE UND M. TILEBEIN. Ansätze zur Wandlungsfähigkeit von Produktionsunternehmen. Ein Bezugsrahmen für die Unternehmensentwicklung im turbulenten Umfeld. *Operations Research*, 90(1), 2000.

[113] H.-P. WIENDAHL. Betriebsorganisation für Ingenieure. Technical report, Carl Hanser Verlag, 1989.

[114] H.-P. WIENDAHL UND R. HERNÁNDEZ. Fabrikplanung im Blickpunkt. Herausforderung Wandlungsfähigkeit. *wt Werkstatttechnik online*, 92(4), 2002.

[115] H.-P. WIENDAHL, D. NOFEN, J. H. KLUSSMANN UND F. BREITENBACH. *Planung modularer Fabriken*. Carl Hanser Verlag, 2005.

[116] WIKIPEDIA. Distributed Component Object Model. http://de.wikipedia.org/wiki/Distributed_Component_Object_Model.

[117] WIKIPEDIA. Betriebsdatenerfassung. http://de.wikipedia.org/wiki/Betriebsdatenerfassung, Stand: 18.05.2007.

[118] WIKIPEDIA. Mutex. http://de.wikipedia.org/wiki/Mutex, Stand: 18.05.2007.

[119] WIKIPEDIA. Nebenläufigkeit. http://de.wikipedia.org/wiki/Nebenl%C3%A4ufige_Programmierung, Stand: 18.05.2007.

[120] WIKIPEDIA. Programmierparadigma. http://de.wikipedia.org/wiki/Programmierparadigma, Stand: 18.05.2007.

[121] WIKIPEDIA. Verteiltes System. http://de.wikipedia.org/wiki/Verteilte_Systeme, Stand: 18.05.2007.

[122] S. WIRTH. Die flexible, temporäre Fabrik - Arbeitsschritte auf dem Weg zu wandlungsfähigen Fabrikstrukturen. Technical report, FZ Karlsruhe, Technik und Umwelt, Wissenschaftliche Berichte FZKA-PFT 203, 2001.

[123] WORLD WIDE WEB CONSORTIUM. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). http://www.w3.org/TR/2006/PER-soap12-part1-20061219, Stand: 18.05.2007.

[124] WORLD WIDE WEB CONSORTIUM. Web Services Activity. http://www.w3.org/2002/ws/, Stand: 18.05.2007.

[125] WXPYTHON. http://www.wxpython.org, Stand: 18.05.2007.

[126] WXWIDGETS. http://www.wxwidgets.org, Stand: 18.05.2007.

[127] G. ZEICHEN, B. FAVRE-BULLE UND W. STÖCHER. Europäische Produktion am Wendepunkt. *wt Werkstatttechnik online*, 96, 2006.

[128] ZEROC INC.. http://www.zeroc.org, Stand: 18.05.2007.

[129] P. ZIESCHE. *Nebenläufige & verteilte Programmierung*. W3L GmbH, 2004.

[130] H. ZIMMERMANN. OSI Reference Model - The IS0 Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, Com-28(4), 1980.