Entwicklung einer Methode zur objektorientierten Spezifikation von Steuerungen

Von der Fakultät für Maschinenbau
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von
Dipl.-Ing. Arnulf Braatz
aus Itzehoe

Hauptberichter:	Univ.-Prof. Dr.-Ing. Prof. e.h. Dr.-Ing. e.h. Dr. h.c. E. mult. E. Westkämper
Mitberichter:	Univ.-Prof. Dr. H. Ehrig
Tag der mündlichen Prüfung:	26.04.2005

Institut für Industrielle Fertigung und Fabrikbetrieb der Universität Stuttgart
2005

IPA-IAO Forschung und Praxis

Berichte aus dem
Fraunhofer-Institut für Produktionstechnik und
Automatisierung (IPA), Stuttgart,
Fraunhofer-Institut für Arbeitswirtschaft und
Organisation (IAO), Stuttgart,
Institut für Industrielle Fertigung und
Fabrikbetrieb (IFF), Universität Stuttgart
und Institut für Arbeitswissenschaft und
Technologiemanagement (IAT), Universität Stuttgart

Herausgeber:
Univ.-Prof. Dr.-Ing. Prof. e.h. Dr.-Ing. e.h. Dr. h.c. mult. Engelbert Westkämper
und
Univ.-Prof. Dr.-Ing. habil. Prof. e.h. Dr. h.c. Hans-Jörg Bullinger
und
Univ.-Prof. Dr.-Ing. Dieter Spath

Universität Stuttgart
Institut für Industrielle Fertigung und Fabrikbetrieb

Fraunhofer Institut Produktionstechnik und Automatisierung

Arnulf Braatz

Entwicklung einer Methode zur objektorientierten Spezifikation von Steuerungen

Nr. 423

JOST-JETTER VERLAG
Fachverlag · 71296 Heimsheim

Dr.-Ing. Arnulf Braatz
Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA), Stuttgart

Univ.-Prof. Dr.-Ing. Prof. e.h. Dr.-Ing. e.h. Dr. h.c. mult. Engelbert Westkämper
ord. Professor an der Universität Stuttgart
Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA), Stuttgart

Univ.-Prof. Dr.-Ing. habil. Prof. e.h. Dr. h.c. Hans-Jörg Bullinger
ord. Professor an der Universität Stuttgart
Präsident der Fraunhofer-Gesellschaft, München

Univ.-Prof. Dr.-Ing. Dieter Spath
ord. Professor an der Universität Stuttgart
Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO), Stuttgart

D 93

ISBN 3-936947-66-X Jost Jetter Verlag, Heimsheim

Druck: printsystem GmbH, Heimsheim

Geleitwort der Herausgeber

Über den Erfolg und das Bestehen von Unternehmen in einer marktwirtschaftlichen Ordnung entscheidet letztendlich der Absatzmarkt. Das bedeutet, möglichst frühzeitig absatzmarktorientierte Anforderungen sowie deren Veränderungen zu erkennen und darauf zu reagieren.

Neue Technologien und Werkstoffe ermöglichen neue Produkte und eröffnen neue Märkte. Die neuen Produktions- und Informationstechnologien verwandeln signifikant und nachhaltig unsere industrielle Arbeitswelt. Politische und gesellschaftliche Veränderungen signalisieren und begleiten dabei einen Wertewandel, der auch in unseren Industriebetrieben deutlichen Niederschlag findet.

Die Aufgaben des Produktionsmanagements sind vielfältiger und anspruchsvoller geworden. Die Integration des europäischen Marktes, die Globalisierung vieler Industrien, die zunehmende Innovationsgeschwindigkeit, die Entwicklung zur Freizeitgesellschaft und die übergreifenden ökologischen und sozialen Probleme, zu deren Lösung die Wirtschaft ihren Beitrag leisten muss, erfordern von den Führungskräften erweiterte Perspektiven und Antworten, die über den Fokus traditionellen Produktionsmanagements deutlich hinausgehen.

Neue Formen der Arbeitsorganisation im indirekten und direkten Bereich sind heute schon feste Bestandteile innovativer Unternehmen. Die Entkopplung der Arbeitszeit von der Betriebszeit, integrierte Planungsansätze sowie der Aufbau dezentraler Strukturen sind nur einige der Konzepte, welche die aktuellen Entwicklungsrichtungen kennzeichnen. Erfreulich ist der Trend, immer mehr den Menschen in den Mittelpunkt der Arbeitsgestaltung zu stellen - die traditionell eher technokratisch akzentuierten Ansätze weichen einer stärkeren Human- und Organisationsorientierung. Qualifizierungsprogramme, Training und andere Formen der Mitarbeiterentwicklung gewinnen als Differenzierungsmerkmal und als Zukunftsinvestition in *Human Resources* an strategischer Bedeutung.

Von wissenschaftlicher Seite muss dieses Bemühen durch die Entwicklung von Methoden und Vorgehensweisen zur systematischen Analyse und Verbesserung des Systems Produktionsbetrieb einschließlich der erforderlichen Dienstleistungsfunktionen unterstützt werden. Die Ingenieure sind hier gefordert, in enger Zusammenarbeit mit anderen Disziplinen, z. B. der Informatik, der Wirtschaftswissenschaften und der Arbeitswissenschaft, Lösungen zu erarbeiten, die den veränderten Randbedingungen Rechnung tragen.

Die von den Herausgebern langjährig geleiteten Institute, das

- Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA),

- Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO),

- Institut für Industrielle Fertigung und Fabrikbetrieb (IFF), Universität Stuttgart,

- Institut für Arbeitswissenschaft und Technologiemanagement (IAT), Universität Stuttgart

arbeiten in grundlegender und angewandter Forschung intensiv an den oben aufgezeigten Entwicklungen mit. Die Ausstattung der Labors und die Qualifikation der Mitarbeiter haben bereits in der Vergangenheit zu Forschungsergebnissen geführt, die für die Praxis von großem Wert waren. Zur Umsetzung gewonnener Erkenntnisse wird die Schriftenreihe „IPA-IAO - Forschung und Praxis“ herausgegeben. Der vorliegende Band setzt diese Reihe fort. Eine Übersicht über bisher erschienene Titel wird am Schluss dieses Buches gegeben.

Dem Verfasser sei für die geleistete Arbeit gedankt, dem Jost Jetter Verlag für die Aufnahme dieser Schriftenreihe in seine Angebotspalette und der Druckerei für saubere und zügige Ausführung. Möge das Buch von der Fachwelt gut aufgenommen werden.

Engelbert Westkämper Hans-Jörg Bullinger Dieter Spath

„Ich merkte, dass alles, was Gott tut, das besteht für ewig;
man kann nichts dazutun noch wegtun."
Prediger 3, 14

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit am Fraunhofer Institut für Produktionstechnik und Automatisierung (IPA) sowie meiner Mitarbeit im DFG-Schwerpunkt-Programm *Softwarespezifikationen* am Institut für industrielle Fertigung und Fabrikbetrieb (IFF) der Universität Stuttgart. Die Thematik und die Lösungsidee der Dissertation entstammen meiner intensiven Beschäftigung mit der Anwendung von Techniken der Softwarespezifikation und deren Einsatz in der Produktionstechnik.

Meinem Doktorvater Herrn Professor Westkämper danke ich für die akademische Freiheit und wohlwollende Unterstützung meiner Ideen. Mein Dank gilt ebenfalls Herrn Professor Ehrig und Herrn Dr. Große-Rhode für die vielen anregenden Diskussionen, die mir geholfen haben eine fachliche Brücke zwischen der theoretischen Informatik und ihrer Anwendung in der Produktionstechnik zu schlagen. Insbesondere danke ich Herrn Professor Ehrig für die sorgfältige Durchsicht dieser Arbeit, für die hilfreichen Verbesserungsvorschläge und die Übernahme des Mitberichts.
Meinen wissenschaftlichen Weggefährten und Kollegen am Institut danke ich für die viele wertvollen Anregungen und Diskussionen sowie für die manchmal notwendigen Aufmunterungen bei nachlassender Motivation. Besonders hervorheben möchte ich dabei meinen Dissertationspaten Dr. Arno Ritter und ebenso Luzia Schumacher für das unermüdliche und doch bisweilen ermüdende Korrekturlesen dieser Arbeit.
Nicht unerwähnt bleiben sollen auch meine studentischen Mitarbeiter Stefan Aschenbach, Stefan Engelke und Dimitri Minich, die durch ihre Unterstützung bei der praktischen Umsetzung der theoretischen Konzepte einen maßgeblichen Anteil am Erfolg dieser Arbeit haben.

Mein ganz besonderer Dank gilt meiner Familie, insbesondere meiner Mutter, für die geduldige Unterstützung und Begleitung meines langen Ausbildungsweges.

Stuttgart, August 2005

Arnulf Braatz

Inhalt

0 Grundlagen

0.1 Abkürzungen

ABK	Anzeige- und Bedienkomponente
ALI	Application-Layer-Interface
API	Application Programming Interface
ACSI	Abstract Common Service Interface
BDE	Betriebsdaten-Erfassung
C	Classifier
CIM	Classifier Interaction Model
CL	Classifier Lifcycle,
CRM	Classifier Relationship Model
COM/DCOM	Component Object Model/ Distributed Component Object Model®
DLL	Dynamic linked libraries (zur Laufzeit einer Applikation gebundene Programm-Bibliothek)
ERP	Enterprise Resource Planning
IT	Informationstechnologie
IPC	Industrie-PC
MAS	Multi-Agenten-System
OCL	Object Constraint Language
ODEMA	Object-oriented Method for Developing Technical Multi-Agent systems
OMG	Object Management Group
OPC	Object Linking and Embedding (OLE) for Process Control
PA	Produktionsagenten
PA-MAS	Produktionsagenten Multi-Agenten-System
PF	Prozessführung
PNK	Prozessnahe Komponenten (z.B. SPS, IPC)
PS	Prozesssteuerung
SLL	Static Linked Libraries (zum Zeitpunkt der Programmierung gebundene Programm-Bibliothek)
UML	Unified Modelling Language
MES	Management Execution System
MDE	Maschinendaten-Erfassung
CASE-Tool	Computer Aided Software Engineering Tool
QoS	Quality of Service
XMI	XML Metadata Interchange
XML	Extensible Markup Language

0.2 Abkürzungen aus der Referenzfallstudie

AUS	Ausgangslager
BL	Batterieladestation
EIN	Eingangslager
HTF	Holonisches Transportfahrzeug
HTS	Holonisches Transportsystem (mindestens ein HTF)
PB	Parkbereich
WZM	Werkzeugmaschine

0.3 Formelzeichen

Formelzeichen	Bedeutung	Formelzeichen	Bedeutung
D_S	Der spätestens gültige Zeitpunkt zur Beendigung eines Dienstes	$t_{syncDelay}$	Verzögerung eines synchronen Kommunikationssystems
m_{K+SP}	Anzahl der in Reihe geschalteten Steuerungsprozesse mit jeweils einem Kommunikationskanal	t_{REC}	Zeitpunkt des Empfangens einer Nachricht oder eines Stimuli
n_{AG}	Anzahl der Geräte in einem Kommunikationssystem	t_{SEND}	Zeitpunkt des Sendens einer Nachricht oder eines Stimuli
n_{AGmax}	Maximal erlaubte Anzahl der Geräte in einem Kommunikationssystem	$t_{Prozess}$	Zulässige Reaktionszeit auf Änderungen im technischen Prozess
R_S	Der frühestens erlaubte Zeitpunkt zur Beendigung eines Dienstes	t_V	Allgemeine zeitliche Verzögerung
Δt	Zeitschranke im Sequenzdiagramm	$t_{V,OP}$	Maximale Ausführungszeit eines Dienstes
$t_{asyncDelay}$	Asynchrone Verzögerungszeit eines Kommunikationskanals als Komponenten-Implementierung	$t_{V,SP}$	Verzögerungszeit eines Steuerungsprozesses
Δt_{EXE}	Ausführungszeit einer nicht-unterbrechbaren Aktivität	$t_{V,Kanal}$	Verzögerungszeit eines Kommunikationskanals
$\Delta t_{EXE,MAX}$	Maximale Ausführungszeit einer nicht-unterbrechbaren Aktivität	$t_{Zykl,SP}$	Zykluszeit eines Steuerungsprozesses
Δt_{LOOP}	Zeitspanne einer zeitgesteuerten Schleife	$t_{Zykl,Kom}$	Buszykluszeit
$t_{maxResponseTime}$	Maximale Antwortzeit ein Steuerungsprozesses		

Tabelle 0.1: Übersicht der in dieser Arbeit verwendeten Formelzeichen

0.4 Unified Modeling Language

Die *Unified Modeling Language* (UML) ist ein grafische Beschreibungsmittel der logischen und physischen Aspekte von Software und besteht aus acht verschiedenen Diagrammtypen (siehe Tabelle 0.2).

UML-Diagrammtyp (Deutsch/Englisch)	Beschreibung	Kategorie nach [HRUBY_00]
Anwendungsfalldiagramm/ Use Case Diagram	Darstellung der Interaktionen zwischen externen Klassen (Akteure) und dem zu entwickelnden System	Anwendungsfallsicht: C (alle Schichten)
Klassendiagramm/ Class Diagram	Darstellung von Klassen, Objekten, Paketen und deren statische Beziehungen. Klassendiagramme dienen sowohl der Darstellung von Klassen aus dem Anwendungsbereich (z.B. Akteure) als auch von Klassen von Objekten der Softwarearchitektur.	Analysesicht, Prozesssicht: C, CRM (alle Schichten)
Sequenzdiagramm/ Sequence Diagram	Darstellung der zeitlichen Abfolge von Nachrichten als Modell der Interaktion bzw. Kommunikation zwischen Objekten (Instanzen von Klassen).	Analysesicht, Prozesssicht: CIM (alle Schichten)
Kollaborationsdiagramm/ Collaboration Diagram	Darstellung des Zusammenwirkens von Objekten beim Ausführen von Operationen.	Prozesssicht, Analysesicht: CIM (alle Schichten)
Zustandsdiagramm/ Statechart Diagram	Darstellung der Zustände und Zustandsübergänge einer Klasse. In Ausnahmefällen kann auch das Verhalten von Anwendungsfällen auf diese Weise beschreiben werden.	Prozesssicht, Analysesicht, Anwendungsfallsicht: CL (alle Schichten)
Aktivitätsdiagramm/ Activity Diagram	Eine spezielle Form des Zustandsdiagramms. Zustandsübergänge erfolgen unmittelbar nach der Beendigung einer Aktion oder Aktivität. Algorithmen und Geschäftsprozesse können so dargestellt werden.	Prozesssicht, Analysesicht, Anwendungsfallsicht: CL (alle Schichten)
Komponentendiagramm/ Component Diagram	Darstellung der physischen Softwarekomponenten eines Systems, wie z. B. Programmcode, Textdateien, ausführbare Programme, etc. und ihre Abhängigkeiten untereinander.	Implementierungssicht: C, CRM
Einsatzdiagramm/ Deployment Diagram	Darstellung der Einsatzumgebung eines Systems, wie Prozessoren, Rechner, Netzwerkgeräte usw. und die darin existierenden Objekten, Softwarekomponenten und Prozessen.	Einsatzsicht: C, CRM

Tabelle 0.2: Beschreibung der UML-Diagrammtypen (C = Classifier, CIM = Classifier Interaction Model, CL = Classifier Lifcycle, CRM = Classifier Relationship Model)

Ein allgemein gültiges Vorgehensmodell ist für die UML nicht spezifiziert. Eine ausführliche Darstellung und Hinweise zur Anwendung der UML-Diagrammtypen sind in den Spezifikationen der *Object Management Group* (OMG) und in entsprechender Literatur zu finden [KAHL_98, BALZERT_99, BOOCH_99, RUMB_99, OESTER_99b, UML_03].

Jeder UML-Diagrammtyp ist mindestens einer Sicht zugeordnet [KRUCHT_98]. Nach HRUBY lässt sich das Sichten-Konzept durch ein verfeinerndes und dekomponierendes Schichten-Konzept ergänzen, so dass eine zwei-dimensionale Darstellung entsteht, die jedem UML-Diagrammtyp eine UML-Diagrammtyp-Rolle zuweist (siehe Tabelle 0.2, [HRUBY_00]). Syntax, Semiotik und Semantik der UML werden durch diese Ergänzung nicht beeinflusst. Es führt aber zur Anwendung der Erweiterungsmechnismen der UML.(*Stereotypen*, *Eigenschafteswerte* und *Einschränkungen*), mit deren Unterstützung UML-Beschreibungselemente einer bestimmten UML-Diagrammtyp-Rolle zugeordnet werden. Die Semantik der UML ist durch das sog. *Metamodell* der UML beschrieben, welches mittels Klassen-Diagrammen und entsprechenden durch die *Object Constraint Language* (OCL) formulierten algebraisch-logischen Einschränkungen beschrieben ist [UML_03].

Sicht / Schicht	Anwendungsfall		Analyse		Prozess		Implementierung		Einsatz	
System	**CRM** System Use Case RM	**CIM** System Use case IM	Domain Object RM	Domain Object IM	System Diagram	System IM	System Component Diagram	System Component IM	System Node Diagram	System Node IM
	C System Use Case	**CL** System Use Case Lifecycle	Domain Object	Domain Object Lifecycle	System	System Lifecycle	System Components	System Component Lifecycle	System Nodes	System Node Lifecycle
Subsystem	Subsystem Use Case RM	Subsystem Use Case IM	Analysis Subsystem RM	Analysis Subsystem Diagram	Subsystem RM	Subsystem IM	Subsystem Component Diagram	Subsystem Component IM	Subsystem Node Diagram	Subsystem Node IM
	Subsystem Use Case	Subsystem Use Case Lifecycle	Analysis Subsystem	Analysis Subsystem Lifecycle	Subsystem	Subsystem Lifecycle	Subsystem Components	Subsystem Component Lifecycle	Subsystem Nodes	Subsystem Node Lifecycle
Klassen	Class Use Case RM	Class Use Case IM	Analysis Class RM	Analysis Object ID	Class RM	Object IM				
	Class Use Case	Class Use Case Lifecycle	Analysis Class	Analysis Class Lifecycle	Class	Class Lifecycle				
Prozeduren	**Wird von der UML nicht beschrieben**		Analysis Procedure	Analysis Procedure algorithm	Procedure	Procedure algorithm				
Programmcode			System (Software und Hardware)							

Tabelle 0.3: Sichten-Schichten-Darstellung der UML nach [HRUBY_00] (C=Classifier, CIM = Classifier Interaction Model, CL = Classifier Lifecycle, CRM = Classifier Relationship Model)

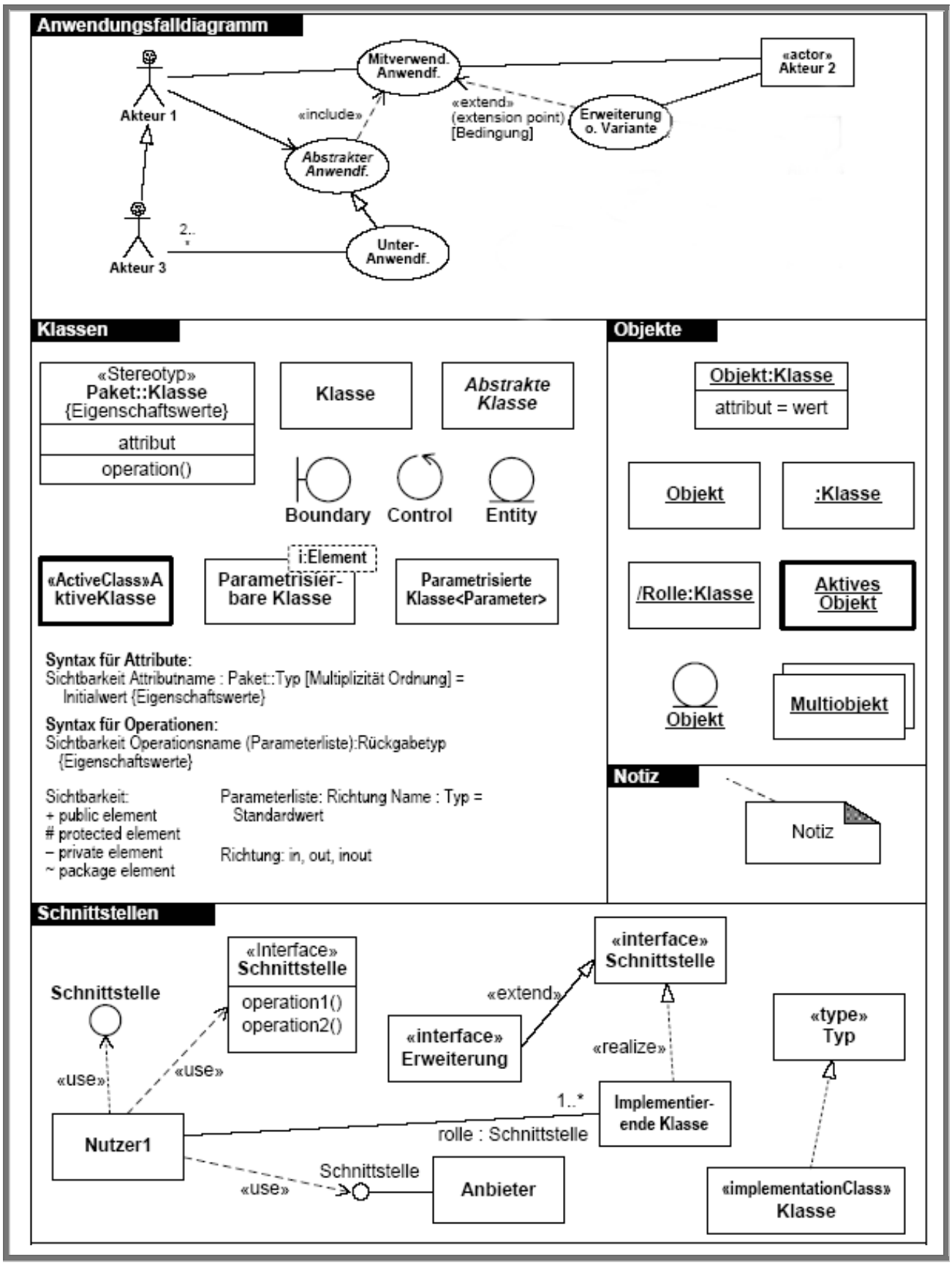

Bild 0.1: Notationsübersicht der UML 1.5 (Teil 1) [OOSE_03]

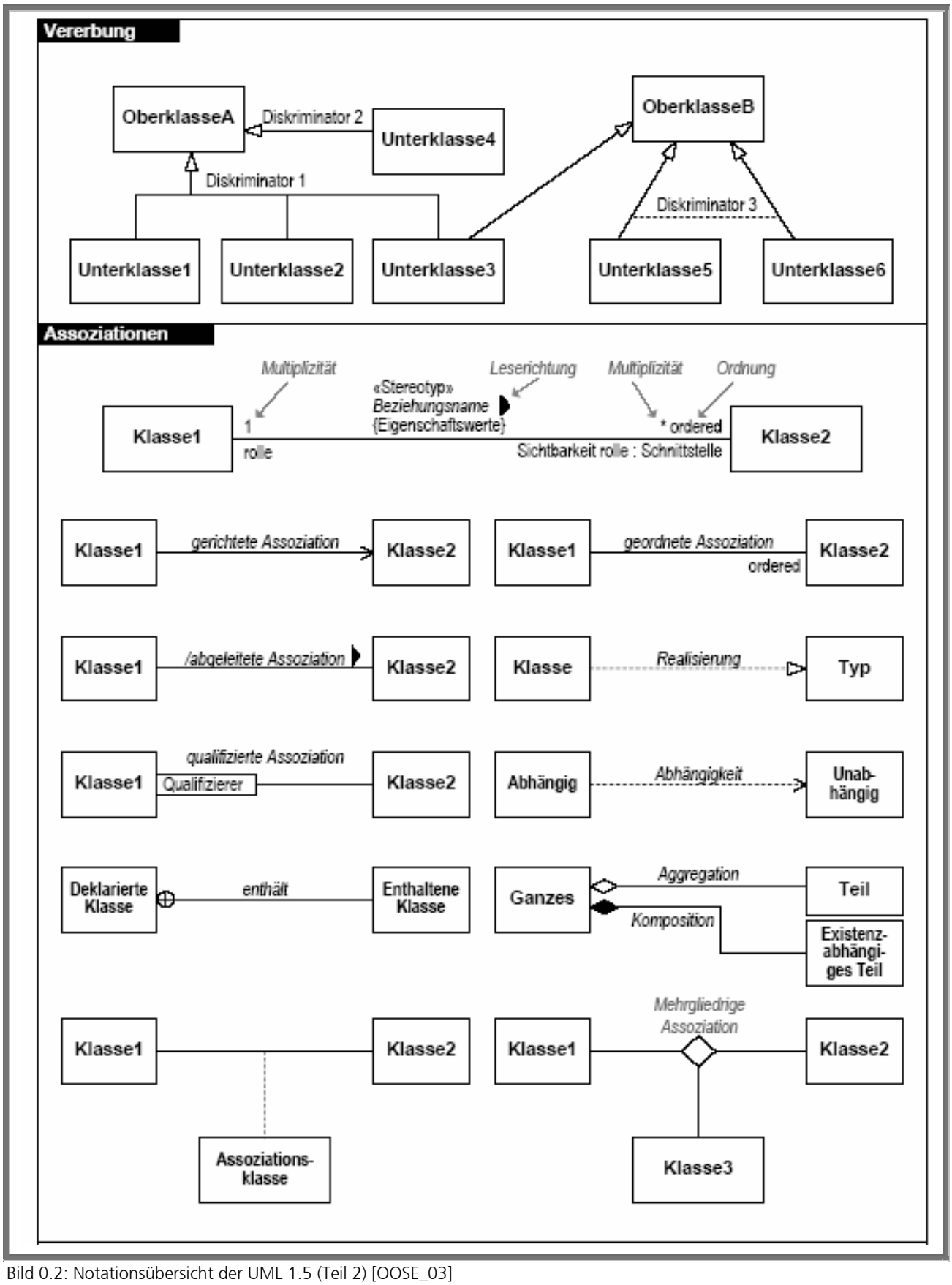

Bild 0.2: Notationsübersicht der UML 1.5 (Teil 2) [OOSE_03]

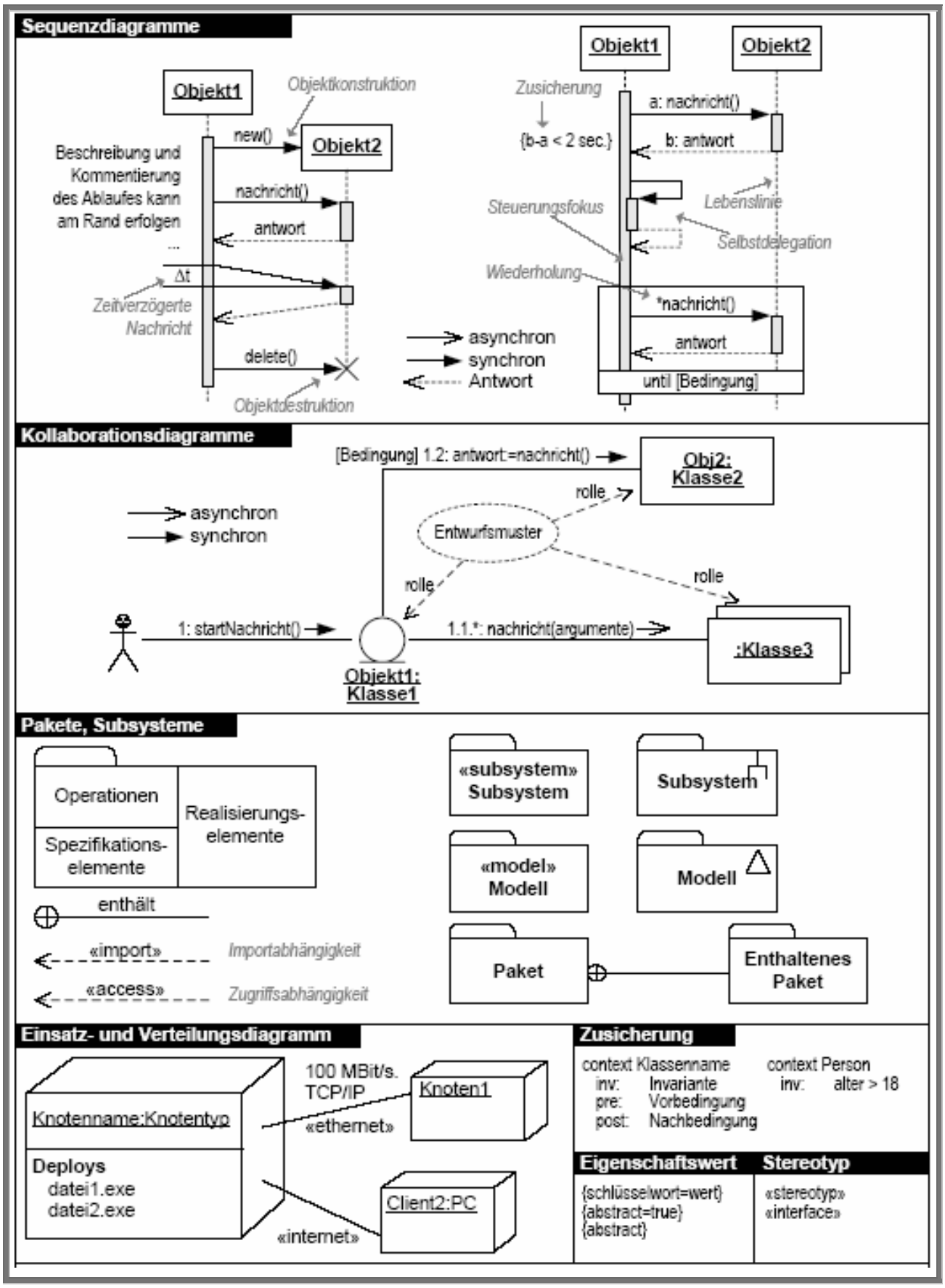

Bild 0.3:: Notationsübersicht der UML 1.5 (Teil 3) [OOSE_03]

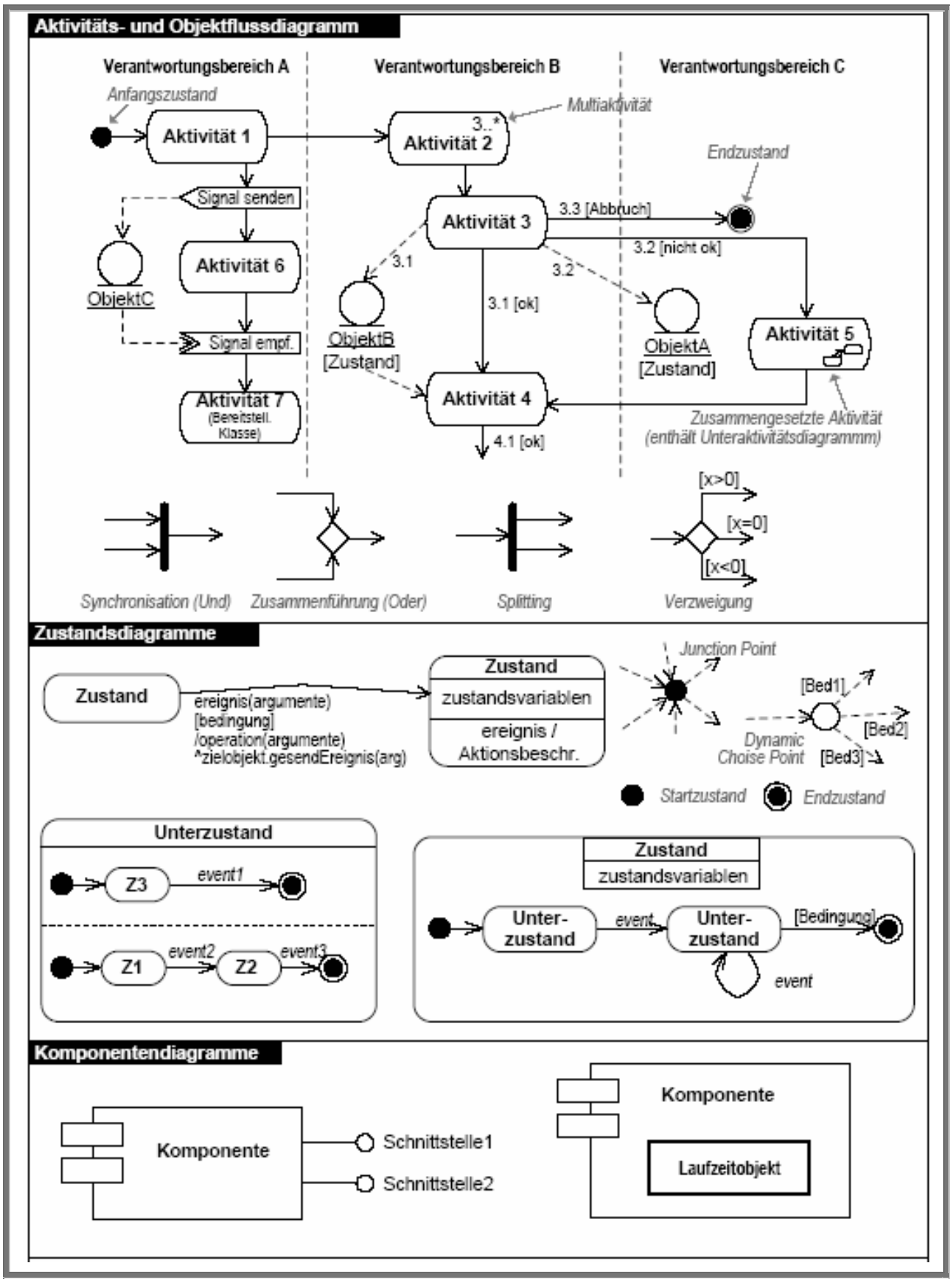

Bild 0.4: Notationsübersicht der UML 1.5 (Teil 4) [OOSE_03]

1 Einleitung

1.1 Ausgangssituation und Problemstellung

Vom Standpunkt der Entwicklung von Steuerungen aus muss festgestellt werden, dass im Hinblick auf die Programmierung von Steuerungen eine heterogene Gerätestruktur vorliegt. Zwar lassen sich Bemühungen um eine Vereinheitlichung der Programmiertechniken beobachten - d.h. für die Programmierung einer SPS haben sich die IEC-61131-Sprachen durchgesetzt - jedoch wird die Anwendersoftware für z.B. numerische Steuerungen und Roboter-Steuerungen noch immer mittels spezifischer Programmiersprachen und Werkzeuge entwickelt. Aktuelle Bestrebungen durch Komponenten-basierte Steuerungen (z.B. ProfiNet®) unterschiedliche Steuerungstechniken auf Basis einer gemeinsamen Middleware zusammen zu führen, haben zu einer weiteren Erhöhung der Heterogenität geführt, da sie auf Hersteller-spezifischen Technologien aufsetzen und somit eine Vielfalt unterschiedlicher Steuerungsplattformen entstanden sind [BEND_00, TERW_01].

Die zunehmende Integration der Daten aus den Steuerungen und der Unternehmens-IT – vertikale Integration genannt – bringt es mit sich, dass Techniken (IPC, Ethernet und Windows®-Betriebssysteme) und Programmiersprachen (z.B. C# und JAVA) aus der Informationstechnik auch in Automatisierungssystemen zu finden sind [BEND_00, DUMSKY_02, GRUND_02, TERW_01]. Neben einer weiteren Erhöhung der Heterogenität der Programmiertechniken sind aus Sicht der methodischen Entwicklung von Software bzw. Steuerungen unterschiedliche Ansätze zu beobachten [BRAATZ_01, KIESS_95, WEISSHA_99, ZERBE_99]. So ist es in der Software-Entwicklung üblich, dass methodisch und durchgängig Planungsdaten und Spezifikationen während des gesamten Entwicklungsprozesses (z.B. Anforderungsdefinition und Software-Entwurf) erstellt werden, bevor die Funktionen programmiert werden. Die grafische und objektorientierte Beschreibungssprache *Unified Modeling Language* (UML) ist hier der Industrie-Standard zur Spezifikation von Software [BOOCH_99, ELTING_01a, b, OESTER_99b, UML_03]. Zur Spezifikation von Steuerungen dagegen werden in der Regel textuelle Spezifikationen in Form von Entwicklungsdokumenten (z.B. Lastenheft) verwendet.

Neben der Heterogenität bezüglich der Programmierung lässt sich aus den dargestellten Entwicklungen in der Steuerungstechnik ein prinzipielles Anwachsen der Software-basierten Funktionen in Automatisierungssystemen ableiten. Zur Zeit wird davon ausgegangen, dass der Anteil von Software bis zu 40 % des Wertes einer Maschine beträgt [SCHLEIF_00, STETTER_00].

Zukünftig ist ein Anhalten dieses Trends zu erwarten. Die Einführung dezentraler Organisationsformen in Fabriken und deren Produktionssystemen ist eine Reaktion auf die Anforderung aus wachsenden Produktvarianten und verkürzten Produktlebenszyklen [KRALL_02, RITTER_00, ULLMA_01, WEST_98]. Hieraus folgt die Steigerung der lokalen Autonomie bzw. Entscheidungskompetenz von Menschen und Maschinen in dezentralen Zellen und damit auch eine weitere Steigerung des Anteils an Software in den Maschinen. Für die Entwicklung von Produktionssystemen bedeutet dies eine grundsätzliche

Abkehr vom bisherigen Paradigma der zentralen Einzelsteuerung hin zu Komponenten- und Agenten-orientierten Ansätzen [HOEPF_97, LAENGLE_97, SUND_01].
Steigende Heterogenität bezüglich der Programmierung und zunehmende Komplexität der Software-basierten Steuerungen von Maschinen und Produktionssystemen bei gleichzeitiger Anwendung von Entwicklungsmethoden, die von der mechanischen und der Elektrokonstruktion geprägt werden, führt zu einer unwirtschaftlichen Entwicklung von Steuerungen. In der industriellen Praxis lassen sich deshalb überlange Entwicklungszeiten und eine mangelnde Qualität der Software beobachten [STETTER_01, Studie_03].
Grundsätzlich sind diese Defizite schon seit ca. 1960 zu beobachten [BOOCH_97]. Mit der UML wurde 1997 ein Beschreibungsmittel definiert, welches Komplexität und Heterogenität von IT-Systemen beherrschbar machen sollte. In den letzten Jahren hat sich nicht nur dieses objektorientierte Beschreibungsmittel in Verbindung mit entsprechenden *Computer Aided Software Engineering-Tools* (CASE-Tools) im industriellen Einsatz durchgesetzt, sondern es sind auch eine Reihe von anwendungsspezifischen, UML-basierten Spezifikationsmethoden entwickelt worden [DOUG_99, OESTER_99a, ELTING_01a,b]. Auch der prinzipielle Einsatz von objektorientierten Beschreibungen zur Spezifikation von Steuerungen wurde bereits analysiert und für aussichtsreich befunden [BRAATZ_01, FEHR_02, FISCHER_02, KIESS_95, METZ_97, RÜFFER_02]. Eine durchgängige und objektorientierte - d.h. UML-basierte - Methode zur Spezifikation von Steuerungen, die sowohl die spezifischen Modelle und das Vorgehen aus der Automatisierungstechnik integriert als auch die Übertragung der Spezifikation auf die heterogenen Automatisierungsgeräte berücksichtigt, ist nicht verfügbar. Diese Methoden-Eigenschaften sind aber unbedingte Voraussetzung, um eine objektorientierte Methode - und damit auch Denkweise - in die industrielle Automatisierungstechnik einzuführen und entsprechende Akzeptanz bei den Anwendern zu erreichen.

1.2 Zielsetzung und Vorgehensweise

Ziel dieser Arbeit ist die Entwicklung einer objektorientierten Methode zur Spezifikation von Steuerungen, die die UML als objektorientiertes Beschreibungsmittel für Steuerungen verwendet und die die mit ihr verbundenen Aspekte des Vorgehens bzw. des Entwicklungsprozesses in das Engineering von Automatisierungssystemen überführt. Besonders berücksichtigt werden hierbei Steuerungen in dezentralen Produktionssystemen, da diese zum einen die steuerungstechnische Grundlage zukünftiger Fabrik-Konzepte bilden und zum anderen zu den komplexesten Systemen in der Steuerungstechnik gehören [RITTER_01, RITBRA_01, SUND_01].
Ausgangspunkt dieses Vorhabens bildet hierbei die Analyse der heute in der industriellen Praxis verwendeten Methoden und Beschreibungsmittel zur Spezifikation von Steuerungen. Anhand von Anforderungskriterien werden diese bewertet und Defizite aufgezeigt. Zur Kompensation dieser Defizite und zur Erhöhung der Anwender-Akzeptanz bezüglich der zu entwickelnden Methode werden in Kapitel 3 Begriffsmodelle und Modellvorstellungen ausgewählt, die in die UML übertragen werden müssen. Basierend auf dieser Analyse werden dann in Kapitel 4 und 5 ein erweitertes Vorgehensmodell

und das dazugehörige UML-basierte Beschreibungsmittel als orthogonale Elemente der Methode konzipiert. Basis des Beschreibungsmittels sind hierbei die acht Diagrammtypen der UML und ihre Diagrammtyp-Rollen (siehe Tabelle 0.2 und 0.3 sowie die Bilder 0.1 bis 0.4). Zur Übertragung der Modelle und Modellvorstellungen und zur Integration des Engineering von Automatisierungssystemen werden die standardisierten Erweiterungsmechanismen der UML (*Eigenschaftswerte*, *Stereotypen* und *Einschränkungen*) angewandt. In Kapitel 6 werden dann die spezifischen UML-Diagrammtypen und Entwicklungsschritte ausgearbeitet und anschließend in Kapitel 7 anhand der *Referenzfallstudie Produktionstechnik* angewendet und validiert (Beschreibung der Referenzfallstudie, siehe Kapitel 10).

Die nach der Aufgabenstellung dieser Arbeit zu entwickelnde Methode wurde bereits unter der Bezeichnung ODEMA (*Object-oriented Method for Developing Technical Multi-Agent-Systems*) in die Literatur eingeführt und wird auch im Kontext dieser Arbeit so bezeichnet [BRAATZ_00, BRAATZ_01a,b, BRAATZ_03a]. Die ODEMA-Methode wurde im Rahmen des Schwerpunktprogramms *Softwarespezifikationen* (SPP-Nr. 1064) der Deutschen Forschungsgemeinschaft (DFG) entwickelt [DFG_98].

2 Beschreibungsmittel und Methoden zur Spezifikation von Steuerungen – Stand der Technik

In diesem Kapitel wird der Stand der Technik bezüglich der Methoden zur Spezifikation von Steuerungen dargestellt. Hierbei stehen Steuerungen von automatisierten Produktionssystemen im Mittelpunkt. Technologische Basis von Steuerungen sind Automatisierungsgeräte und Steuerungssoftware, die die anwendungsspezifischen Funktionen beschreibt. Die Spezifikation von Steuerungen ist somit als anwendungsspezifische Variante der Software-Spezifikation zu verstehen [POHLKE_94, REINHA_96, LAUBER_99, SCHNIED_99].

Nach der Darstellung der elementaren Begriffe und Definitionen wird mit den Methoden-Axiomen und den Steuerungsfunktionen ein Instrumentarium eingeführt, mit dem die Methoden und Beschreibungsmittel bewertet werden.

2.1 Begriffe und Definitionen

Spezifikation

Nach BENDER ist allgemein unter einer Spezifikation die Formulierung dessen zu verstehen, was ein System tun soll [BENDER_92]. Der im Kontext dieser Arbeit verwendete erweiterte Spezifikationsbegriff ist der Softwaretechnik entliehen und beschreibt eine Entwicklungstechnik für Software, die mittels einer bestimmten Anzahl definierter Entwicklungsschritte schwerpunktmäßig grafische Beschreibungen der zu entwickelnden Software erzeugen [MUEHL_92, SOMMER_96].

Modell, Notation und Beschreibungsmittel

In der Automatisierungstechnik repräsentiert ein *Beschreibungsmittel* in graphischer Form Sachverhalte zur visuellen Wahrnehmung und Speicherung in elektronischen Dokumenten (z.B. Lastenheft). Hierbei besteht ein Beschreibungsmittel aus Darstellungselementen (*Semiotik*), Konventionen über ihre Kombination (*Syntax*) und Zuordnungen entsprechender automatisierungstechnischer Konzepte (*Semantik*). Besitzt ein Beschreibungsmittel eine mathematische Basis und eine definierte vollständige Syntax wird es als *formal* bezeichnet. Ist die Basis nicht mathematisch beschreibbar, ist es *semiformal*. Ist die Syntax nicht vollständig definiert, spricht man von einem *informalen* Beschreibungsmittel [SCHNIED_98]. Werden Sachverhalte allgemein und nicht mit dem Ziel der Spezifikation eines Systems beschrieben, so spricht man von Modellen. Informale Modelle sind in der Automatisierungstechnik insbesondere durch sog. *Begriffsmodelle* verbreitet [SCHNIED_01].

Aus Sicht der Software-Entwicklung ist die Notation der UML als semiformales Beschreibungsmittel für Software-basierte Systeme anzusehen. Die Diagrammtypen der UML bilden hierbei die Syntax und Semiotik und das Meta-Modell der UML die Semantik [BOOCH_99]. Der Begriff der *präzisen Semantik* ist ebenfalls auf die UML anzuwenden. Hierunter wird verstanden, dass ein Beschreibungsmittel zwar semiformal ist, aber die Semantik trotzdem eindeutig beschrieben ist, ohne dabei auf mathematische Beschreibungen zurückzugreifen.

Dekomposition und Verfeinerung

Allgemein wird unter der Dekomposition die Zerlegung eines größeren Ganzen in kleinere Einzelteile verstanden. Die Komposition ist im Sinne einer Umkehr-Operation als Konstruktion zu verstehen. Bezogen auf ein Modell oder eine Spezifikation wird mittels Strukturelementen durch Komposition ein abstraktes Modell konstruiert. In diesem Zusammenhang ist die Verfeinerung die Vorschrift, auf welche Weise die Komposition und Dekomposition vorgenommen werden kann. Da die Verfeinerung i. d. R. Voraussetzung für Komposition und Dekomposition ist, wird sie im Rahmen dieser Arbeit nicht von diesen Begriffen getrennt, sondern implizit eingeschlossen (*Strukturelle Verfeinerung*).

Steuerung

Nach DIN 19226 ist der Begriff *Steuerung* als Vorgang in einem System definiert, bei dem eine oder mehrere Größen (*Prozessdaten*) als Eingangsgrößen die Ausgangsgrößen auf Grund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Erweitern lässt sich dieser Begriff, wenn unter einer Steuerung eine Anzahl informationstechnischer Geräte, Sensoren, Aktoren und Kommunikationssysteme (*Verteiltes System*, s.u.) verstanden wird, die fest zu einem physischen Automatisierungssystem gehören. Ergänzend zu dieser physischen Sicht existiert eine logische Sicht auf die Steuerung, die diese als eine Sammlung von Steuerungsfunktionen auffasst, welche über Befehl bzw. Meldungen mit den Anlagenfunktionen (*Technischer Prozess*, s.u.) verbunden ist [DUBBEL_97, SCHNIED_99].

Validation, Verifikation und Prüfung

Validation heißt, durch Vergleichen von Ergebnissen der Steuerungsentwicklung bzw. Eigenschaften eines realisierten Automatisierungssystems mit vorher spezifizierten Anforderungen, die Gültigkeit und Plausibilität zu belegen. Typische Verfahren der Validation sind *Simulation* und *Test*. Die Verifikation ist ein mathematisch-analytisches Verfahren, das die Existenz bestimmter Eigenschaften eines Modells beweist. Hierzu wird ein formales Beschreibungsmittel benötigt, das bei erfolgreicher Verifikation eine korrekte Spezifikation liefert. [SCHNIED_99, LAUBER_99, MOIK_02]. Wird die semiformale UML als Beschreibungsmittel eingesetzt, so ist die Verifikation in Prüfungen der syntaktischen und der semantischen Konsistenz zu unterteilen. Syntaktische Konsistenz bedeutet i.d.R. eine Überprüfung der Namensräume von Spezifikationselementen. Soll funktionale und temporale Widerspruchsfreiheit geprüft werden, so ist dies dem Bereich der semantischen Konsistenz zuzuordnen.

Werkzeug

Ein Werkzeug (*Tool*) ist im Kontext dieser Arbeit als ein Instrumentarium zur Unterstützung von Menschen bei der Erstellung und Verifikation bzw. Prüfung und Validation von Modellen zu verstehen [SCHNIED_98]. Beim Einsatz der UML als Beschreibungsmittel hat sich der Begriff des *Computer Aided Software-Engineering (CASE)-Tools* durchgesetzt, das die beschriebenen Funktionen eines Werkzeuges durch die Programmcodegenerierung aus dem Modell heraus ergänzt.

Technischer Prozess

Ein *Prozess* ist die Gesamtheit von aufeinander einwirkenden Vorgängen in einem System, durch die Materie, Energie oder Information umgeformt, transportiert oder gespeichert werden. Ein *Technischer Prozess* ist ein Prozess, dessen physikalische Größen mit technischen Mitteln erfasst und beeinflusst werden [BENDER_92, VDI3694]. Ein Produktionsprozess ist ein technischer Prozess, der sich in einem Produktionssystem vollzieht [DUBBEL_97].

Anwendungsprozess

Anwendungsprozesse bilden das System zur Informationserfassung, -verarbeitung, -darstellung und -ausgabe zum Zweck der Steuerung eines Automatisierungssystems. Anwendungsprozesse sind Laufzeitvorgänge in einem Automatisierungsgerät, die explizit vom Kommunikationsprozess unabhängig sind, aber auf diesen über eine definierte Schnittstelle zugreifen [BENDER_92, REISSEN_98]. Anwendungsprozesse können aus Sicht der Software-Entwicklung in *leicht*- und in schwergewichtige Anwendungsprozesse unterschieden werden. Im Kontext der UML sind Anwendungsprozesse Instanzen (*Objekte*) von *aktiven Klassen*.

Kommunikationsprozess und -system

In der Informationstechnik sind Kommunikationsprozesse dynamische Vorgänge, die durch Abarbeiten von Kommunikationsprogrammen Informationen zwischen Anwendungsprozessen austauschen. Die über ein gemeinsames Verbindungsmedium (Kabel, Lichtwellenleiter, Funkstrecke, etc.) zusammenwirkenden, lokalen Kommunikationsprozesse werden in ihrer Gesamtheit als Kommunikationssystem bezeichnet [BENDER_92].

Verteiltes System

Im Rahmen dieser Arbeit wird der Begriff *Verteiltes System* in Anlehnung an die Arbeiten von SOLVIE, ENSLOW, HERRTWICH wie folgt gebraucht: Ein verteiltes System ist ein Datenverarbeitungssystem, dessen Daten- oder auch Steuerungsfunktionen auf mehrere, mittels eines Kommunikationssystems miteinander verbundene Automatisierungsgeräte verteilt sind [SOLVIE_95, ENSLOW_78, HERRT_89].

Softwarekomponenten-Technik

Komponententechnik in der objektorientierten Softwaretechnik ist der zusammenfassende Begriff von Softwarekomponenten-Implementierungen und Softwarekomponentenmodellen bzw. –Spezifikationen [GRUHN_00, CHEES_02].

Komponenten-Implementierungen sind als ein Stück Software in binärer Form zu verstehen, deren Implementierung gekapselt ist. Der Zugriff auf die gekapselten Funktionen von außen ist ausschließlich über Schnittstellen möglich. Komponenten-Implementierungen sind in binärer Form zwischen Plattformen (auch *Komponenten-Container* genannt) austauschbar.

Komponentenmodelle legen einen Rahmen für die Entwicklung und Ausführung von Komponenten-Implementierungen fest, der die Kompositions- und Kollaborationsmöglichkeiten von Komponenten-Implementierungen beschreibt.
Die Komponenten-Instanz wird als Ansammlung von Objekten verstanden, die zur Laufzeit die Instanzen der Klassen bildet, die im Komponenten-Modell spezifiziert wurden. [GRUHN_00].

Echtzeit-Prozessdaten

Echtzeit-Prozessdaten sind Prozessvariablen des technischen Prozesses, die einen Prozesswert (Wert und physikalische Einheit) und die zu ihm gehörige Zeitinformation (*Zeitstempel*) zusammenfassen. Diese Zeitinformation beinhaltet den Erstellungszeitpunkt und kann zusätzlich den Gültigkeitszeitraum des Prozesswertes und Datenkonsistenzattribute enthalten [SOLVIE_95, LAUBER_99].

Nachrichten und Ereignisse

In objektorientierten Modellen beschreiben Nachrichten die Kommunikation zwischen Objekten. Dabei ist ein Objekt der Sender und eines oder mehrere andere Objekte sind der bzw. die Empfänger einer Nachricht. Hierbei ist der Empfang einer Nachricht als Ereignis des Empfänger-Objektes zu verstehen. Die Übergabe von Daten als Parameter der Nachricht an den Empfänger erweitert das Ereignis zu einer empfangenen Nachricht [SELIC_94].

Echtzeitdienst, Echtzeitschranken und Echtzeitsystem

Ein *Echtzeitdienst* ist ein Dienst, der innerhalb einer bestimmten, durch die Umgebungsanforderungen vorgegebenen Zeitspanne ausgeführt wird. Diese Zeitspanne erstreckt sich über das Intervall $[R_S, D_S]$, wobei R_S (Release) den frühesten erlaubten und D_S (*Deadline*) den spätestens noch gültigen Zeitpunkt für die Beendigung des Dienstes angibt. In Automatisierungssystemen ist das Intervall bestimmt von der Dynamik des technischen Prozesses.
Eine harte *Zeitschranke* ist in diesem Zusammenhang ein Zeitpunkt D_S, bei dessen Verfehlung die zugeordnete Aktion keinen Nutzwert mehr hat und sogar Schaden anrichten kann. Eine weiche Zeitschranke ist ein Zeitpunkt D_S, bei dessen Verfehlung die zugeordnete Aktion weniger Wert hat bzw. höhere Kosten verursacht [SOLVIE_95]. Ein *Echtzeitsystem* ist ein System, das mindestens einen Echtzeitdienst nutzt oder anbietet. Automatisierungssysteme sind Echtzeitsysteme, die in der Regel nur die Spezifikation von D_S erfordern. Zeitschranken gehören aus Sicht der Software-Spezifikation zum *Quality of Service* (QoS) eines Dienstes [PROFILE_02].

2.2 Definition des Begriffes Spezifikationsmethode

Der Begriff *Spezifikationsmethode* bzw. Methode zur Spezifikation von Steuerungen lässt sich aus Sicht der objektorientierten Software-Entwicklung und aus Sicht der Automatisierungstechnik unterschiedlich definieren.

Die objektorientierte Software-Entwicklung versteht im Allgemeinen unter einer Methode bzw. Entwicklungsmethode die Kombination der beiden orthogonalen Dimensionen *Beschreibungsmittel bzw. Notation* und dem *Vorgehensmodell*. Ergänzen lässt sich diese Kombination durch sog. *Management-Techniken* der Projektdurchführung als eine dritte Dimension. Diese kann aber für die hier zu entwickelnde Methode zur Spezifikation von Steuerungen vernachlässigt werden, da sie abhängig ist Projektgröße und unternehmerischem Umfeld eines Projektes [OESTER_99a, REINHO_97]. Aus den gleichen Gründen können die in der Software-Entwicklung sonst üblichen parallelen Entwicklungsstränge, wie z.B. das Projektmanagement und die Qualitätssicherung, ebenfalls für die hier gestellte Aufgabe vernachlässigt werden [REINHO_97, REINHO_00].

Nach SCHNIEDER ist allgemein unter einer Methode zur Entwicklung von Automatisierungssystemen ein planmäßiges Verfahren zur Erlangung von Zielen zu verstehen, das im Verbund mit einem Werkzeug und einem Beschreibungsmittel angewendet wird. Die Anwendung der Methode orientiert sich dabei an einem Phasenmodell, ähnlich dem Vorgehensmodell in der Software-Entwicklung, das sich in die Phasen *Planung*, *Systementwurf*, *Realisierung*, *Test* und *Einsatz* gliedert (*BMW-Prinzip*) [SCHNIED_99].

Da sich zunehmend die Auffassung durchsetzt, dass die Entwicklung von Steuerungen eine anwendungsspezifische Form der Software-Entwicklung ist, soll dieser Arbeit der Methoden-Begriff der objektorientierten Software-Entwicklung zu Grunde liegen [BRAATZ_01b, BENDER_00]. LAUBER definiert im Verständnis der objektorientierten Software-Entwicklung hierfür ein Vorgehensmodell für die Entwicklung von Steuerungen [LAUBER _99]. Dies beschreibt eine Entwicklung als sequentiellen Durchlauf von Phasen in Vorwärtsrichtung. Rücksprünge in vorgelagerte Phasen sind ebenfalls möglich, was eine Verifikation, Prüfung oder Validation der Spezifikation voraussetzt (siehe Bild 2.1). Parallele Projekt-Aktivitäten (z.B. Qualitätssicherung) werden hierbei nicht berücksichtigt.

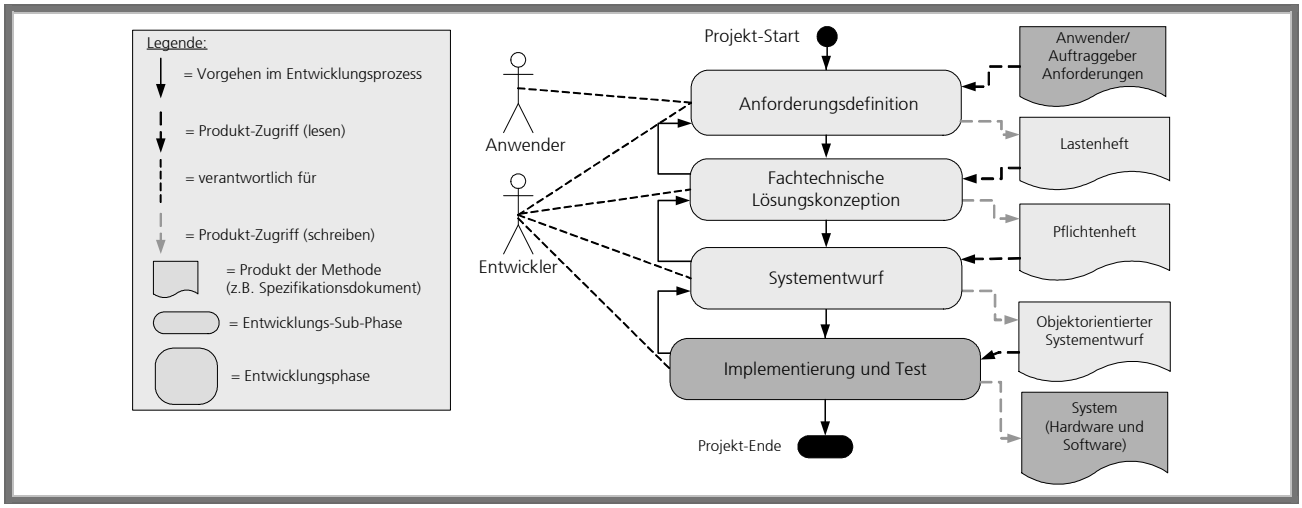

Bild 2.1: Vorgehensmodell und Produkte der Spezifikationsmethode (Phasen und Produkte, die nicht zur Spezifikation gehören, sind dunkel gefärbt)

Wird der Begriff der Spezifikation aus Sicht der Automatisierungstechnik ergänzt, ergibt sich die Spezifikationsmethode als Abschnitt der Entwicklungsmethode, der die Phasen *Anforderungsdefinition*, *Fachtechnische Lösungskonzeption* und *Systementwurf* umfasst (siehe Bild 2.1). Darüber hinaus wer-

den die Phasen und Produkte (Lastenheft, Pflichtenheft, Objektorientierter Steuerungsentwurf) dieses Vorgehensmodells mit den elementaren Rollen *Anwender* (Betreiber des Automatisierungssystems und Auftraggeber) und *Entwickler* (Verantwortlicher für die Entwicklung der Steuerung) ergänzt. *Rollen* werden in der Automatisierungstechnik und Software-Entwicklung als generische Beschreibungen von Verantwortlichkeiten und Kompetenzen der an einem Projekt Beteiligten verstanden [OESTER_99a, SCHNIED_99].

Der hier definierte Begriff der Spezifikationsmethode ist zwar der objektorientierten Software- bzw. Steuerungsentwicklung entnommen, aber doch ausreichend generisch, dass auch die in diesem Kapitel analysierten nicht-objektorientierten Beschreibungsmittel unter dieser Definition betrachtet werden können.

2.3 Methoden-Axiome und Steuerungsfunktionen zur Bewertung von Spezifikationsmethoden

Um die in Kapitel 2 vorgestellten Beschreibungsmittel bzw. Methoden zur Spezifikation von Steuerungen vergleichen und bewerten zu können, werden zunächst die *Systemaxiome* zur Klassifizierung eines Beschreibungsmittels eingeführt (siehe Tabelle 2.1).

Nach SCHNIEDER lässt sich prinzipiell jedes technische System, also auch eine Steuerung, mit Hilfe der Axiome des *Struktur-*, *Dekompositions-*, *Kausal-* und *Temporalprinzips* beschreiben, bzw. ein Beschreibungsmittel muss diese Axiome umfassen, um eine Steuerung zu spezifizieren [SCHNIED_99]. Die Eignung eines Beschreibungsmittels für Steuerungen ist dabei abhängig vom Erfüllungsgrad dieser Systemaxiome.

Die Systemaxiome lassen sich aber ausschließlich auf ein Beschreibungsmittel anwenden. Da im Rahmen dieser Arbeit eine Methode zur Spezifikation von Steuerungen mit dem Ziel der Implementierung von Software entwickelt werden soll, wird zunächst das Dekompositionsprinzip in *System-*, *Software-* und *Projekt-Dekompositionsprinzip* unterschieden und somit die Systemaxiome erweitert. Diese erweiterten Axiome sind nicht orthogonal zueinander, sondern sind als unterschiedliche Sichten auf die Dekomposition anzusehen. Das Paradigma *Objektorientierung,* und damit auch die Sichtenkonzeption der UML, ist in diesem Zusammenhang durchgängig auf alle erweiterten Systemaxiome anwendbar.

Nach SOMMERVILLE ist ein Beschreibungsmittel auch dadurch gekennzeichnet, dass bezüglich der methodischen Anwendung ein Widerspruch in der Forderung nach maximaler Formalität und ebenfalls größtmöglicher Verstehbarkeit und Anwendbarkeit durch den Entwickler besteht [SOMMER_96]. In der Praxis müssen *Anwendbarkeit* und *Prüfbarkeit*, also die Definition von Verfahren zur Verifikation, Prüfung und Validation, bei der Auslegung des Erfüllungsgrades dieser Axiome gegeneinander abgewogen werden.

Die auf die dargestellte Weise erweiterten Systemaxiome werden als *Methoden-Axiome* eingeführt. Sie dienen zur Bewertung und zudem als Anforderung für die in dieser Arbeit zu entwickelnde Methode. Zusammenfassend werden die Methoden-Axiome in Tabelle 2.1 detailliert beschreiben.

Methoden-Axiom	Beschreibung
Strukturprinzip	Eine Steuerung als System verstanden besteht aus einer Menge von Teilen, die untereinander und mit der Umgebung in wechselseitiger Beziehung stehen. Die Teile der Steuerung werden durch Größen beschrieben, deren Werte als Zustände der Steuerung angesehen werden können. Die Steuerung ist gegenüber ihrer Umgebung als eigenständig anzusehen [SCHNIED_99]. Eine Steuerung ist prinzipiell als ein *verteiltes System* anzusehen [BENDER_92]. Aus Sicht der Softwareentwicklung wird die Struktur der Steuerung als Makro-Architektur bezeichnet [MEIER_01].
System-Dekompositionsprinzip	Eine Steuerung besteht aus einer Menge von Teilen, die ihrerseits wieder in eine Anzahl von in wechselseitiger Beziehung stehender Unterteile zerlegt werden können. Im Detail betrachtet können die Unterteile wieder Systemmerkmale aufweisen [SCHNIED99].
Software-Dekompositionsprinzip	Aus Sicht der Software besteht eine Steuerung aus einer Menge von Software-Teilen und Unterteilen (Software-Komponenten, Klassen, Anwendungsprozesse, etc.) und ihre wechselseitigen Beziehungen untereinander werden als *Mikro-* bzw. *Software-Architektur* eines Systems bezeichnet [MEIER_01].
Kausalprinzip	Eine Steuerung besteht aus einer Menge von Teilen, deren Zustände und Beziehungen untereinander determiniert sind, d.h. im Sinne eines kausalen Wirkzusammenhanges können spätere Zustände nur von vorangegangenen abhängen. Eine Steuerung befindet sich zu jeder Zeit in einem definierten Zustand bzw. ist durch eine definierte Menge von verteilten Zuständen zu beschreiben [SCHNIED_99].
Temporalprinzip	Eine Steuerung besteht aus einer Menge von Teilen, deren Struktur und Zustände zeitlichen Veränderungen unterliegen. Es gibt eine zeitliche Folge von Abläufen und Veränderungen (z.B. sequentiell oder nebenläufig) [SCHNIED_99].
Projekt-Dekompositionsprinzip	Die Definition der Spezifikationsmethode setzt die Verteilung einer Spezifikation bzw. einer Beschreibung längs des zeitlichen Vorgehens in einem Entwicklungsprojekt voraus. Hierbei ist davon auszugehen, dass der Kenntnisstand im Verlauf des Entwicklungsprojektes wächst (siehe Kapitel 2.2). Hieraus folgt, dass die Spezifikation bzw. die Beschreibung einer Steuerung, nicht die Steuerung selbst, entlang des Entwicklungsprozesses bzw. des Vorgehensmodells und seiner Produkte dekomponiert werden muss [LAUBER_99, VOGHEU_02].
Anwendbarkeit	• Anwendbarkeit beschreibt zum einen die prinzipielle Verstehbarkeit einer Methode in einem bestimmten Anwendungsgebiet. Für die Spezifikation von Steuerungen ist dies die Automatisierungstechnik. Hierzu gehört die Verwendung durchgängig gleicher Beschreibungstechniken, eine möglichst kleine Menge von zu erlernender Syntax und die Integration von Begriffsmodellen und Modellvorstellungen der Automatisierungstechnik [SOMMER_96, SCHNIED_01]. • Vom Standpunkt der Wirtschaftlichkeit beschreibt Anwendbarkeit darüber hinaus, das Maß der Verfügbarkeit und den Verbreitungsgrad von kommerziellen Werkzeugen [VOGHEU_02].
Prüfbarkeit	• Die Prüfbarkeit verlangt von einer Spezifikationsmethode, dass für ein formales Beschreibungsmittel das entsprechende mathematisch-analytische Verfahren zur *Verifikation* beschrieben ist. Im Falle eines semiformalen Beschreibungsmittels ist eine mathematisch-analytische oder informale Beschreibung der Verfahren zur *Prüfung der syntaktischen und semantischen Konsistenz* sowie der *Validation* gefordert. • Aus Sicht des Vorgehensmodells sind *Verifikation*, *Validation* und *Prüfung* als Entwicklungs-Sub-Phase zu definieren.

Tabelle 2.1: Methoden-Axiome zur Bewertung einer Spezifikationsmethode für Steuerungen

Grundsätzlich ist eine Steuerung mit Funktionen des Produktionssystems verbunden, die durch einen technischer Prozesstyp charakterisiert sind. Vereinfacht gesagt kann dieser Prozesstyp über die Art seiner *Prozessvariablen* bestimmt werden, denn Prozessvariablen können ereignis-diskret oder kontinuierlich beschrieben werden. Die Notwendigkeit der Beschreibung beider Varianten von Prozessvariablen im Kontext von Produktionsprozessen führt zu so genannten *hybriden Beschreibungsmitteln*. Im Folgenden wird allerdings statt *kontinuierlich* der Begriff *quasi-kontinuierlich* verwendet, da Steuerungen,

bedingt durch die Verwendung digitaler Datenverarbeitungssysteme, Prozessvariablen zeitdiskret abtasten [SCHNIED_99, ENSTE_00].

Eine geeignete Variante der Klassifizierung von Steuerungsfunktionen ist anhand der zu verarbeitenden Prozessvariablen vorzunehmen. Im *OSACA*-Projekt wurde eine solche Klassifizierung erarbeitet [OSACA_02, SPERL_97]. Daran angeschlossen werden kann ebenfalls eine Typisierung der Beschreibungsmittel für eine bestimmte Steuerungsfunktion. Nach SCHNIEDER sind die Beschreibungsmittel in der Automatisierungstechnik in V*erknüpfungs-*, *Programmiertechnisch-*, *Zustandsorientierte* und *Aufbauorientierte Beschreibungsmittel* zu unterscheiden (*Beschreibungsorientierung*, [SCHNIED_99]). Tabelle 2.2 stellt zusammenfassend die Abhängigkeiten von Steuerungsfunktion, Prozessvariablen und Beschreibungsorientierung gegenüber (siehe auch [VOGHEU_02]). Die Klassifizierung der Steuerungsfunktionen wird hierbei als verfeinerte Betrachtung der Anforderungen *Kausal-* und *Temporalprinzips* betrachtet.

Steuerungsfunktionen nach OSACA (Deutsch / *Englisch*)	**Prozessvariablentyp**	**Beschreibungsorientierung**
Achssteuerung / *Axis Control*	Quasi-kontinuierlich	Programmiertechnisch-orientiert
Bewegungssteuerung / *Motion-Control*	Quasi-kontinuierlich	Programmiertechnisch-orientiert, Aufbauorientiert
Ablaufsteuerung / *Logic Control*	Ereignis-diskret	Verküpfnungsorientiert, Aufbauorientiert, Zustandsorientiert
Steuerung der Neben-Funktionen / *Process-Control*	Unabhängig von Prozessvariablen (MDE, BDE, Diagnose, aber auch Nebenfunktionen des technischen Prozesses etc.)	Programmiertechnisch-orientiert
Bedienersteuerung / *Man-Machine Control*	Ereignis-diskret als Teil der Ablaufsteuerung, sonst unabhängig vom Prozesstyp	Programmiertechnisch-orientiert, Zustandsorientiert, grafische Beschreibung der Bediener-Schnittstelle

Tabelle 2.2: Klassifizierung von Steuerungsfunktionen

2.4 Verknüpfungsorientierte Beschreibungsmittel

Die klassische und grundlegende Form der Spezifikation von Steuerungen ist die mittels algebraisch-logischen Beschreibungsmitteln. In der Regel geschieht dies mittels Gleichungen und Ungleichungen. Prozessvariablen sind *Aussagen*, die sich als *Operanden* mittels *Junktoren* (z.B. *Nicht-*, *Oder-* und *Und-*Verknüpfung) zu neuen Aussagen verknüpfen lassen [DIN66000].

Methodisch kann diese Art der Spezifikation durch Wahrheitstafeln und *Karnaugh-Tafeln* unterstützt werden. Weitere Varianten der Beschreibung auf dieser Basis, die sich zusätzlich am physischen Aufbau eines Automatisierungssystems orientiert, sind der *Logikplan* (DIN 19239, IEC 117-15) und die Teil-Sprache der IEC-61131-3, der *Kontaktplan* (KOP).

Verknüpfungsorientierte Beschreibungen unterstützen sehr gut das Kausalitätsprinzip in Bezug auf die formale Beschreibbarkeit von Ablaufsteuerungen bzw. Verriegelungen. Sie sind verifizierbar und in der Steuerungstechnik sehr weit verbreitet (*Anwendbarkeit* und *Prüfbarkeit*). Die Beschreibung des *Strukturprinzips* und die verschiedenen Varianten des Dekompositionsprinzips von Steuerungen werden

nicht unterstützt. Das Temporal-Prinzip und die Spezifikation komplexer Prozessvariablen sind kaum berücksichtigt [SCHNIED_99].

2.5 Programmiertechnisch-orientierte Beschreibungsmittel

Auch die in der Informationstechnik verwendeten Beschreibungsmittel, die sich am prozeduralen bzw. funktionalen Programmierparadigma (Aktionen, bedingte Verzweigungen und Schleifen) orientieren, werden zum Entwurf von Steuerungen in der Automatisierungstechnik verwandt. Zu diesen Beschreibungsmitteln gehören die grafischen Beschreibungsmittel *Programmablaufplan* (DIN 66001 und DIN 66262) sowie das *Struktogramm* nach *Nassi* und *Shneidermann* (DIN 66261). Im Kontext der UML kann das Aktivitätsdiagramm als programmiertechnisch-orientiert charakterisiert werden.

Prinzipiell weisen programmiertechnisch-orientierte Beschreibungsmittel die größte Nähe zu Programmiersprachen auf. Zur Unterscheidung soll hier dienen, dass zu einem Beschreibungsmittel eine grafische Syntax gehört. Textuelle Beschreibungen von Steuerungen, z.B. der *Strukturierte Text* (ST) als Teil-Sprache der IEC-61131-3, sind als Geräte-bezogene Programmiersprachen aufzufassen.

Mit einem programmiertechnisch-orientierten Beschreibungsmittel lassen sich besonders Algorithmen und zyklisch ablaufende Programme darstellen, wie sie z.B. für diskrete Regel-Algorithmen notwendig sind (*Achs-* und *Bewegungssteuerung*, siehe Tabelle 2.2). Grundsätzlich weisen diese Beschreibungsmittel verglichen mit verknüpfungsorientierten Beschreibungsmitteln eine bessere Beschreibbarkeit des Temporalprinzips auf. Darüber hinaus aber sind sie durch die gleichen Schwächen gekennzeichnet wie diese [REINHA_96, POLKE_94, SCHNIED_99].

2.6 Zustandsorientierte Beschreibungsmittel

Die Beschreibung des Verhaltens einer Steuerung durch die diskrete Abfolge von globalen und lokalen Zuständen ist die Grundlage von zustandsorientierten Beschreibungsmitteln. Die mathematische Grundlage hierfür ist das Automatenmodell [SCHNIED_99].

Als Beschreibungsmittel für Steuerungen im Bereich der eingebetteten Systeme werden *Automatengraphen* und vor allem die *Statecharts* nach HAREL angewandt [HAREL_87]. Für Statecharts gilt dies vor allem deshalb, weil sie als *Zustandsdiagramm* in die UML übernommen wurden. Eine wesentliche Eigenschaft von Statecharts bzw. von Zustandsdiagrammen ist es, dass sie neben der zustandsorientierten Beschreibung mittels Nebenläufigkeit und Hierarchisierbarkeit der Zustände auch Beschreibungstechniken zur Dekomposition bereit stellen. Diese beziehen sich aber ausschließlich auf eine verfeinerte Beschreibung des Verhaltens und spezifizieren somit weder das *System-* noch das *Software-Dekompositionsprinzip*.

Für den Anwendungsbereich der Spezifikation von Steuerungen in der Automatisierungstechnik lässt sich feststellen, dass zwar zunehmend zustandsorientierte Beschreibungsmittel eingesetzt werden, sich aber noch keine verbindliche Syntax bzw. kein standardisierter Diagrammtyp durchgesetzt hat [VOGHEU_02, PREHN_02].

Prinzipiell sind zustandsorientierte Beschreibungsmittel ausschließlich zur Beschreibung von Ereignisdiskretem Verhalten geeignet, also zur Beschreibung von Ablauf- und Bedienersteuerungen. *Kausal-* und *Temporalprinzip* sind dabei ausreichend beschreibbar. Die Möglichkeit zur Formulierung von Nebenläufigkeit und hierarchischer Verfeinerung ist aber nicht ausreichend, um die Methoden-Axiome *Struktur-* und *System-Dekompositionsprinzip* von Steuerungen zu erfüllen. Das *Software-Dekompositionsprinzip* wird nicht berücksichtigt.

2.7 Aufbauorientierte Beschreibungsmittel

Für nahezu jede Steuerungsfunktion ist ein eigenes Beschreibungsmittel bzw. eine eigene Programmiersprache ist in der Automatisierungstechnik entwickelt worden. Hierbei orientiert sich die grafische Syntax am physikalischen Aufbau der Steuerung und am Typ des technischen Prozesses [SCHNIED_99]. Zu den am weitesten verbreiteten aufbauorientierten Beschreibungsmitteln in der Automatisierungstechnik gehört der *Kontaktplan* als Teil der IEC-61131, welcher aber auch als verknüpfungsorientiert verstanden werden kann (siehe Kapitel 2.4). Als Beispiel für aufbauorientierte Programmiersprachen lassen sich z.B. die Roboter-Programmiersprachen *Industrial Robot Language* (IRL) und *VAL II* aufzählen (*Bewegungssteuerung*, [SIEG_96]).

Da sich viele Steuerungsgeräte auch mittels objektorientierter Programmiersprachen (z.B. JAVA, C++) programmieren lassen, entsteht auf diese Weise eine hohe Anzahl aufbauorientierter Programmiersprachen, die als Beleg für die Heterogenität von Automatisierungsgeräten bezüglich ihrer Programmierung gewertet werden kann [GRUND_02]. Tabelle 2.3 zeigt eine Auswahl der wichtigsten aufbauorientierten Programmiersprachen im Zusammenhang mit den entsprechenden Steuerungsfunktionen. Aufbauorientierte Beschreibungsmittel verhalten sich bezüglich der Erfüllung der Methoden-Axiome genauso wie die Verknüpfungsorientierten.

Steuerungsfunktionen	Programmiersprachen	Objektorientiert
Achssteuerung	DIN 66025, EXAPT, IEC 61131, etc.	Nein
Bewegungssteuerung	VAL II, RAPT, ROBEX, Industrial Robot Language	Nein
Ablaufsteuerung	IEC 61131-3	Nein
	C++, JAVA	Ja
Steuerung der Neben-Funktionen	C++, JAVA	Ja
Bedienersteuerung	Visual Basic, C++, JAVA	Ja

Tabelle 2.3: Aubauorientierte Programmiersprachen und Bescheibungsmittel [SIEG_96, PLCOPEN_02, DIN66312, DIN66025, IEC61131-1]

2.8 Softwarekomponentenmodell als Beschreibungsmittel

Die Nutzung der Softwarekomponententechnik ist eine Methode der Softwareentwicklung, die auf Dekomposition bzw. Komposition gleichartiger Teile eines Software-basierten Systems beruht, wobei ihre interne Spezifikation (*Kausal-* und *Temporalprinzip*) gekapselt und dem Entwickler verborgen bleibt. Eine wesentliche Motivation für ihren Einsatz ist die Wiederverwendung bereits bestehender Softwarefunktionalitäten in Form von *Dynamic linked Libraries* (DLL), *Static Linked Libraries* (SLL) oder ausführbaren Programmen (*Komponenten-Implementierungen*). In der Automatisierungstechnik ein-

gesetzte Softwarekomponenten-Techniken sind das ActiveX/*Distributed Component Object Model*® (DCOM) und *JavaBeans*®. Beide werden zur Entwicklung von Bedienersteuerungen eingesetzt. Darüber hinaus ist die DCOM-Technologie die Basis für den Software-Bus *OLE for Process Control* (OPC) [HOANG_99]. Komponenten-Implementierungen in Form sog. *Komponentenbasierter Software-Frameworks* werden in der Steuerungstechnik ebenso eingesetzt [TRAUB_02, DUJMO_02].

KE	Ausprägung	Beschreibung
Komponenten-Implementierung	Ja/Nein	Ist eine Komponenten-Implementierung spezifiziert (Ja), so besteht die Komponenten-Technologie aus einem Komponentenmodell und einer Komponenten-Implementierung.
Kommunikations-konzept	Nachrichtenkanal/Bus	*Nachrichtenkanäle* sind gepufferte Sammelbecken für Nachrichten bzw. für Daten, die gelesen oder geschrieben werden können. Ein *Bus* ermöglicht die Auswahl genau eines Empfängers (Komponente) durch einen Sender.
Schnittstellenbe-schreibung	Methoden/Ereignisse/Attribute/ Nachrichten	Beschreibt die verfügbaren Eigenschaften einer Komponenten-Schnittstelle (Komponentenmodell).
Ortstransparenz	Ja/Nein	Verfügt eine Komponenten-Technologie über die Eigenschaft *Ortstransparenz*, so kann sie in verteilten Systemen eingesetzt werden.
Aktivität	Aktiv/Passiv	Von aktiven Komponenten geht ein Kontrollfluss aus. Passive Komponenten werden von außen aufgerufen.
Verhaltensbe-schreibung	Programmiersprache/ Zustands-beschreibung / Keine	Diese Eigenschaft beschreibt die Art und Weise, wie und ob das Verhalten einer Komponente beschrieben wird (Komponentenmodell).
Plattform	Laufzeitsystem	Komponententechnologien können an bestimmte Laufzeitsysteme, d.h. z.B. Betriebssysteme, Middleware, gebunden sein.
Granularität	Fein- / Mittel- / Grobkörnig	Komponenten können über einen verschieden großen Umfang an Funktionalitäten verfügen (z.B. grobkörnig = Applikation).

Tabelle 2.4: Beschreibung der Eigenschaften von Softwarekomponenten (KE=Komponenten-Eigenschaften) [GRIFFEL_98]

Beschreibungsmittel von Komponenten-Techniken sind die Komponentenmodelle (*Softwaredekompositions-Prinzip)*. Üblich sind hierbei Beschreibungen der Komponenten-Schnittstellen (Schnittstellen-Klassen) durch *Interface Definition Languages* (IDL). Durch das Fehlen von Verhaltensbeschreibungen (*Kausal-* und *Temporal-Prinzip*) und den Einsatz einer IDL als textuelle Beschreibung der Schnittstellen verfügen die relevanten Komponententechnologien über kein implementierungs-unabhängiges Beschreibungsmittel.

Typische Eigenschaften von Softwarekomponenten sind zusammenfassend in Tabelle 2.4 dargestellt. Eine Gegenüberstellung der Komponenten-Technologien anhand der Komponenten-Eigenschaften ist in Tabelle 2.5 beschrieben.

Allgemein werden Automatisierungsgeräte als Gerätekomponenten bezeichnet. Im Kontext der Definition von Softwarekomponenten aber sind Gerätekomponenten von dem hier verwendeten Komponenten-Begriff zu unterscheiden. Ein Automatisierungsgerät enthält eine eingebettete Steuerungsfunktionalität bzw. sensorische oder aktorische Funktionalität, die als eingebettete Softwarekomponenten-Implementierung zu verstehen ist. In diesem Kontext ist das Automatisierungsgerät selbst die Plattform dieser Softwarekomponenten-Implementierung.

KE \ KT	ActiveX/DCOM	Java-Beans
Komponenten-Implementierung	ja	ja
Kommunikationskonzept	Bus	Bus
Schnittstellenbeschreibung	Methoden, Ereignisse (*Interface Definition Language*)	Methoden, Ereignisse, Attribute
Ortstransparenz	ja	nein (lokal)
Aktivität	passiv und aktiv	passiv und aktiv
Verhaltensbeschreibung	Keine	Keine
Plattform	Windows-Betriebssystem (C++, Visual Basic)	JAVA Virtual Machine
Granularität	fein- bis mittelkörnig (komponierbar)	feinkörnig

Tabelle 2.5: Übersicht der Eigenschaften von Komponententechniken (Komponenten-Eigenschaften = KE, Komponententechnologien = KT) [GRIFFEL_98]

2.9 Objektorientierte Methoden für die Entwicklung von Steuerungen

In diesem Kapitel werden die objektorientierten Methoden vorgestellt, die mit dem Fokus auf die Entwicklung und Spezifikation sog. *Embedded Systems* (Eingebettete Systeme) entstanden sind. Da diese Methoden Software mit Echtzeitdiensten (*Temporal-Prinzip*) spezifizieren, werden sie auch prototypisch zur Spezifikation von Steuerungen in der Automatisierungstechnik eingesetzt. Große Bedeutung haben hierbei die Methoden *Realtime Objektoriented Modeling Method* (ROOM) und die von DOUGLASS entwickelte Methode (*Douglass-Methode*) erlangt [DOUG_99]. Das objektorientierte Beschreibungsmittel *Specification and Description Language* (SDL) soll in diesem Zusammenhang ebenfalls betrachtet werden [RÜFFER_02, VOGHEU_02, POSCH_02, SDL_02, SELIC_94].

Die SDL ist zunächst als ein eigenständiges, objektorientiertes Beschreibungsmittel neben der UML aufzufassen. Sie wurde zur Beschreibung von verteilten Telekommunikationssystemen entwickelt. Sie wird aber auch im Rahmen der Entwicklung von Steuergeräten für Kraftfahrzeuge angewandt. Bezogen auf die Sichten-Schichten-Darstellung der UML (siehe Kapitel 0.4) fokussiert die SDL die Prozesssicht mit einem Schwerpunkt auf die Formalität des Beschreibungsmittels [GRABO_02, KOWAL_01]. Die prinzipielle Ähnlichkeit der SDL zur UML hat dazu geführt, dass wesentliche Beschreibungselemente der SDL in die nächsten Version der UML (UML 2.0) integriert werden und somit die SDL für die hier verfolgte Aufgabenstellung ihre eigenständige Bedeutung verliert [GRABO_02, OMG_03].

Im Rahmen dieser Arbeit soll im Folgenden aufgrund der geschilderten Entwicklung unter einem objektorientierten Beschreibungsmittel ausschließlich die UML verstanden werden. Die im Folgenden vorgestellte ROOM- und DOUGLASS-Methode sind daher als objektorientierte Spezifikationsmethoden zu verstehen.

2.9.1 ROOM-Methode und das Profil UML/RT

Die ROOM-Methode wurde Ende der Achtziger-Jahre des vorigen Jahrhunderts von SELIC entwickelt, also noch bevor die UML als Beschreibungsmittel spezifiziert wurde. Sie verfügt über eine eigene Notation und ein eigenes Vorgehensmodell. Im Zuge der Verbreitung der UML wurde die Notation der

Methode aber in Form eines UML-Profils (*UML/Realtime=UML/RT*) auf die UML abgebildet [SELIC_94, SELIC_98]. Es sind verschiedene CASE-Tools zur Erstellung von Spezifikationen und zur automatischen Programmcodegenerierung verfügbar (z.B. *Rose RealTime*®) [RUMPE_01, VOGHEU_02, FISCHER_02, RATIO_02]. Für die ROOM-Methode gilt, was prinzipiell für alle UML-basierten Methoden gilt, dass bedingt durch die Verfügbarkeit unterschiedlicher Diagrammtypen mit abgestuften Abstraktionsgraden (*Schichten*, siehe Kapitel 0.4) das *Projekt-Dekompositionsprinzip* erfüllt wird.

UML/RT verwendet nicht alle Diagrammtypen der UML, sondern lediglich Klassen- und Kollaborationsdiagramme zur Architekturbeschreibung (Statische Modellierung) und UML-Zustandsdiagramme zur Modellierung der Kausalität und Temporalität. Nachrichten zwischen *Kapseln* (Instanzen der Kapsel-Klassen, Stereotyp *capsule*) werden asynchron und gepuffert modelliert. Kapseln verfügen über *Ports*, die mittels *Protokollen* (Instanzen der *protocol*-Klassen, Stereotyp *protocol*) eingehende und ausgehende Nachrichten spezifizieren. Ports werden über *Konnektoren* im Kollaborationsdiagramm verbunden, wobei zwei oder mehrere Ports miteinander verbunden werden können (siehe Bild 2.1).

Aufgrund der kommerziellen Verfügbarkeit von entsprechenden CASE-Tools wird aus wirtschaftlicher Sicht das Axiom *Anwendbarkeit* erfüllt (siehe Tabelle 2.1). Begriffsmodelle und Modellvorstellungen aus der Automatisierungstechnik sind aber nicht berücksichtigt. *Kausal-* und *Temporalprinzip* werden bezüglich der Spezifikation von Ablaufsteuerungen und der Steuerung von Nebenfunktionen unterstützt. Da Aktivitätsdiagramme nicht verwendet werden, muss zur Spezifikation der übrigen Steuerungsfunktionen auf die Programmiersprache C zurückgegriffen werden. Weiterhin fehlt die Möglichkeit zur Beschreibung einer Steuerung aus der physischen Sicht (Implementierungs- und Einsatzsicht). *Struktur-* und *System-Dekompositionsprinzip* sind somit aus Sicht der Steuerung nicht erfüllt. Verfahren zur Verifikation sind dagegen definiert und das Axiom der *Prüfbarkeit* ist damit erfüllt.

Da Kapseln als dezentrale Anwendungsprozesse aufgefasst werden können, wird UML/RT zur Zeit für die Spezifikation von Maschinen-Steuerungen evaluiert [BENDER_99, FISCHER_02, RÜFFER_02, VOGHEU_02]. Auch eine Entwurfsmuster-basierte Kopplung zwischen einem IEC-61131 Programm mit einem UML/RT-Modell wurde bereits entwickelt und beschrieben [HEVER_02].

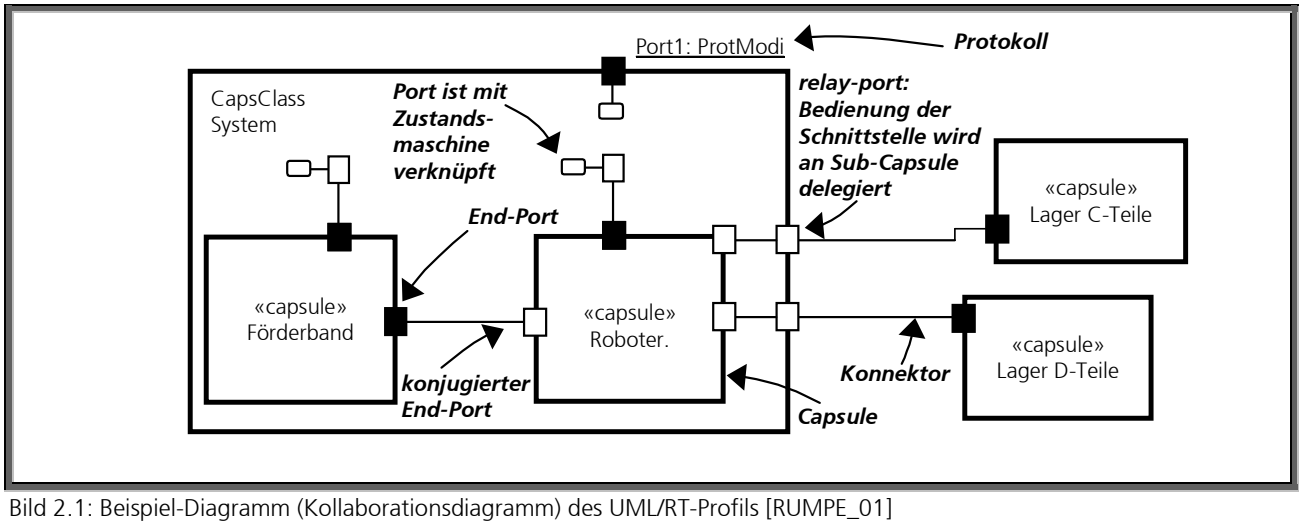

Bild 2.1: Beispiel-Diagramm (Kollaborationsdiagramm) des UML/RT-Profils [RUMPE_01]

2.9.2 DOUGLASS-Methode

DOUGLASS entwickelte in den 90er Jahren des vorigen Jahrhunderts eine Methode zur Entwicklung von Steuerungen für eingebettete und sicherheitskritische Systeme auf Basis der UML [DOUG_99, UMLprofil_03]. Hierbei wird die Methode als Kombination des Vorgehensmodells ROPES (*Rapid Object-oriented Process for Embedded Systems*) und der UML als zentrales Beschreibungsmittel verstanden. Ebenso ist ein UML-CASE-Tool kommerziell verfügbar (*Rhapsody* [DOUG_99]), welches diese Methode unterstützt und eine vollständige Programmcode-Generierung unterstützt.

Im Unterschied zur ROOM-Methode werden in der Douglass-Methode alle Diagrammtypen der UML verwendet, wobei der physische Teil der Beschreibung (Komponenten- und Einsatzdiagramme) nur informativ verwendet wird. Kern der Beschreibung von Steuerungen sind die aus der Beschreibung von Anwendungsfällen gewonnenen Nachrichten und Ereignisse, die die Steuerung verarbeiten muss und die daraus hervorgehende Architektur von Anwendungsprozessen, die als Instanzen aktiver Klassen modelliert werden. Kennzeichen hierbei ist die detaillierte Modellierung der Nachrichten-Kommunikation in Form der Spezifikation von harten und weichen Echtzeitschranken eines Ereignisses (siehe Kapitel 2.1) und die Entwurfsmuster-basierte Beschreibung der Steuerungs- und Kommunikationsprozesse. Die Software-Architektur ist aber prinzipiell zentral orientiert, d.h. die Steuerung reagiert ausschließlich auf Ereignisse, die von außerhalb auf diese wirken.

Die wichtigsten Spezifikationsdokumente dieser Methode sind das Analyse-Modell (*Use Cases*, *Object Behavioral Model*, *Object Structural Model*) und das Design-Modell (*Architectural Design Model, Mechanistic Design Model*, *Detailed Design Model*). Ergänzend existiert eine große Anzahl von Spezifikationsdokumenten in tabellarischer und anderer informaler Form, die die Methode sehr unübersichtlich und die UML-Diagramme für den Anwender nur schwer interpretierbar machen. Außerdem führen diese Dokumente zwar zu einer durchgängigen Spezifikation der Steuerung, sind aber nicht konform zu den gängigen Spezifikationsdokumenten (*Produkte*) der Automatisierungstechnik (siehe Kapitel 2.2). In Verbindung mit der mangelnden Berücksichtigung der Modellvorstellungen bzw. Begriffsmodellen aus der Automatisierungstechnik hat dies dazu geführt, dass die Douglass-Methode in diesen Anwendungsbereich im vernachlässigbaren Umfang zum Einsatz kommt (*Anwendbarkeit*). Für die Erfüllung der Methodenaxiome gelten die gleichen Feststellungen, die auch bezüglich der ROOM-Methode gemacht wurden. Hierbei ist das Axiom *Prüfbarkeit* aber deutlich weniger ausgeprägt.

Zusammenfassend lässt sich sagen, dass die Douglass-Methode sich stärker an der Software-Architektur von Echtzeitbetriebssystemen (RTOS) orientiert als am *System-Dekompositionsprinzip* von Steuerungen in Produktionssystemen.

2.9.3 Agentenorientierte Methode zur Entwicklung von dezentralen Steuerungen

Die zunehmende Anforderung an Flexibilisierung und Wandlungsfähigkeit von Automatisierungssystemen in der Produktionstechnik, d.h. kürzere Umrüstzeiten, kleine Losgrößen bei niedrigen Kosten und *built-to-order*-Geschäftsprozesse, haben zu neuen Paradigmen (z.B. Holonische Systeme) in der

Produktionstechnik geführt, deren technische Grundlage im Allgemeinen in der Agententechnik zu sehen ist [KLEMM_02, HOEPF_97, RITBRA_00, RITTER_99, WEST_99].

Ein Agent wird in der Informationstechnik allgemein als eine Entität mit den Merkmalen: *Flexibilität*, *Autonomie* und *Interaktivität* beschrieben [WEISS_01]. Kooperieren mehrere Agenten innerhalb einer sog. *Agentenplattform* miteinander, so wird dieses System als Multi-Agenten-System (MAS) bezeichnet [RITBRA_01, RITTER_00, FIPA_98]. Agenten kommunizieren mittels Handlungsanweisungen (Agenten-Telegramme), die einen anderen Agenten auffordern, einen Auftrag durchzuführen. Hierbei ist lediglich eine *Kommunikationsprotokoll* spezifiziert. Eine Zwangssteuerung durch z.B. externe Ereignisse gibt es nicht, weshalb ein MAS prinzipiell ein nicht-deterministisches Verhalten aufweist. Ein wesentliches Instrument der Umsetzung von Interaktivität ist die Verhandlung. Ursprünglich wurde der Agenten-Begriff für mobile Software-Agenten definiert. Erweitert man diesen Begriff auf mechatronische Systeme, so wird von *technischen Agenten* gesprochen [LUETH_98, SUND_01].

Übergreifend wird der Agent auch als ein Muster verstanden, das die Grundlage einer Methode zur Softwareentwicklung bildet, *Agenten-Orientiertes Software-Engineering* genannt [WEIS_01, GRIFFEL_98]. Am Fraunhofer IPA wurde eine solche Methode zur Entwicklung von dezentralen Steuerungen für Produktionssysteme entwickelt (*Produktionsagenten-Methode*). Basis dieser Methode ist das Begriffsmodell des *Produktionsagent* (PA) als spezielle Ausprägung des technischen Agenten (siehe Bild 2.2). Das Begriffsmodell des Produktionsagenten basiert auf dem Objekt- und Klassenbegriff der UML und ist unabhängig von bestimmten Agententechnologien. Erste prototypische Szenarien - bestehend aus mobilen Transportrobotern und Bearbeitungsstationen - wurden bereits implementiert [RITBRA_01, RITBRA_02].

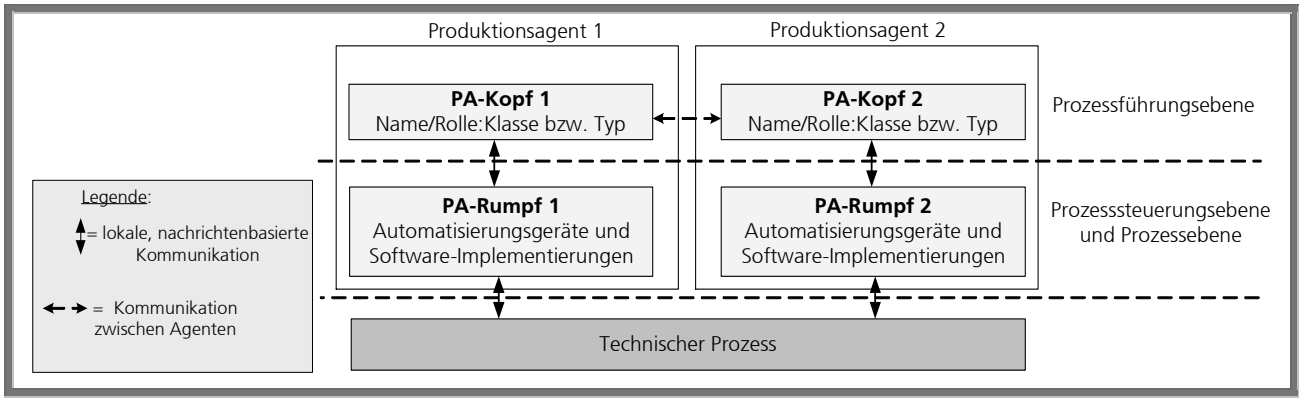

Bild 2.2: Begriffsmodell des Produktionsagenten im Kontext des Ebenenmodells (siehe auch Kapitel 3.1.1)

Ein Produktionsagent ist als Instanz in der logischen Sicht die zusammenfassende Beschreibung von Steuerungsfunktionen (Software- und Gerätefunktionen), die notwendig sind, autonom die spezifizierten Produktionsfunktionen wahrnehmen zu können (*a-Aggregation*). Hierbei wird zwischen dem *PA-Rumpf* und der eigentlichen Agentensteuerung, dem *PA-Kopf*, unterschieden. Voraussetzung hierfür ist, dass alle physischen Teile, die zur Erfüllung der vorgesehenen Produktionsfunktion notwendig sind, dem PA-Rumpf zugeordnet werden. Hierbei ist die mechanische Unabhängigkeit dieser Teile nach außen, also zu anderen PA-Rümpfen hin, gefordert (*p-Aggregation*). Auf diese Weise ist der PA aus logischer und physischer Sicht eine flexible und autonome Einheit.

Zu dem PA-Modell gehört ein Vorgehensmodell, die sog. *Agentifizierung*, welches die Methode komplettiert. Grundsätzlich wird unter der Agentifizierung der Vorgang bezeichnet, ein bestehendes Produktionssystem in eine endliche Menge elementarer Produktionsagenten zu zergliedern. Es werden auf diese Weise lediglich die PA-Köpfe beschrieben und die PA-Rümpfe bleiben transparent. Diese Instanzen können in einem weiteren Schritt mittels Klassen typisiert und durch Zuweisung von Rollen und entsprechenden Kommunikationsprotokollen in ihrem Verhalten beschrieben werden. Es sind mit dem *Bottom-Up-* und dem *Top-Down*-Ansatz zwei Varianten des Vorgehens zur Agentifizierung beschrieben, die bezüglich der Definition der Spezifikationsmethode der Phase *Anforderungsdefinition* zugeordnet sind [RITBRA_02].

2.10 Eigenschaften objektorientierter Vorgehensmodelle

Nach OESTEREICH haben sich durch die Anwendung der UML auch objektorientierte Vorgehensmodelle entwickelt, die zwar prinzipiell mit beliebigen anderen Beschreibungsmitteln angewendet werden könnten, sich aber besonders effizient mit der UML als Beschreibungsmittel ergänzen [REINHO_97, OESTER_99a, REINHO_00]. Diese Eigenschaften sind *Anwendungsfall-getrieben*, *Architektur-zentriert* und *Iterativ-inkrementell*. Ergänzt werden können diese durch die *Komponentenorientierung*. Komponentenorientierte Vorgehensmodelle können als spezielle Ausprägung von Architektur-zentrierten Vorgehensmodellen verstanden werden [ELTING_99, ELTING_02a, CHEES_02]. Eine detaillierte Darstellung diese Eigenschaften findet sich in Tabelle 2.6.

Eigenschaften	Beschreibung
Anwendungsfall-getrieben	Anwendungsfall-getrieben meint, dass die am Anfang eines Projekts erarbeiteten Anwendungsfälle, d.h. die Klartext-Beschreibungen grundsätzlicher Abläufe aus Sicht des Anwenders, Grundlage der Anforderungen an das zu entwickelnde System sind.
Architektur-zentriert	Architektur-zentriert meint, dass die technische Architektur der Steuerung und die angestrebte Software-Architektur im Vorgehen und in den Produkten berücksichtigt werden müssen (*System-* und *Software-Dekomposition*). Sie ermöglichen erst, dass Inkremente entwickelt und verändert werden können, ohne das umgebende System zu verändern.
Iterativ-inkrementell	Das Konzept des iterativ-inkrementellen Vorgehensmodells wurde eingeführt, um Softwaresysteme in einem dynamischen Umfeld entwickeln zu können. Eine Iteration umfasst den immer gleichen, wasserfallartigen Ablauf von Phasen. Am Ende einer jeden Iteration steht ein lauffähiges System mit einer bestimmten Anzahl an Funktionalitäten (*Inkrement*). Durch jede Iteration wächst das System inkrementell bis zur vollen Funktionalität.
Komponentenorientiert	Komponentenorientierte Vorgehensmodelle beschreiben das System- und Software-Dekompositionsprinzip durchgängig mittels verschieden granularer Softwarekomponentenmodelle. Ziel ist hierbei, die funktionalen Anforderungen an ein System mit Hilfe wieder zu verwendender Softwarekomponenten umzusetzen. Die Integration bereits bestehender Software-Komponenten ist aus Gründen der Wirtschaftlichkeit explizit gewünscht (Komponenten-Implementierungen, siehe Kapitel 2.8)

Tabelle 2.6: Eigenschaften objektorientierter Vorgehensmodelle

2.11 Lasten- und Pflichtenheft als Dokumenten-Standard

Lastenheft und Pflichtenheft sind Spezifikationsdokumente eines Entwicklungsprojektes für Automatisierungssysteme (siehe Kapitel 2.2). Sie dienen zugleich auch als Rechtsgrundlage eines Vertrages zwischen Auftraggeber (Rolle *Anwender*) und Auftragnehmers (Rolle *Entwickler*). Allgemein wird unter einem Lastenheft die Zusammenstellung aller funktionalen und nicht-funktionalen Anforderungen des

Auftraggebers hinsichtlich Liefer- und Leistungsumfang verstanden. Das Pflichtenheft beschreibt dagegen, wie und womit die Anforderungen des Lastenheftes realisiert werden sollen. Die Umsetzung der Anforderungen ist nicht eindeutig und Aufgabe des Auftragnehmers [REINH_96, STETT_00, LAUBER_99].

Prinzipiell können alle geeignet erscheinenden Beschreibungsmittel in diesen Spezifikationsdokumenten eingesetzt werden, üblich sind jedoch überwiegend textuelle Spezifikationen (*Informale Beschreibungsmittel*). Als Standard in der praktischen Anwendung von Lasten- und Pflichtenheft kann die VDI/VDE-Richtlinie 3694 angesehen werden, die eine Gliederung dieser Dokumente vorgibt [VDI3694].

Abschnitte des Lastenheftes nach VDI 3694	Abschnitte des Pflichtenheftes nach VDI 3694
1. Einführung in das Projekt	*Abschnitte 1. bis 8. aus dem Lastenheft*
2. Beschreibung der Ausgangssituation (Ist-Zustand)	9. Systemtechnische Lösung
3. Aufgabenstellung (Soll-Zustand)	10. Systemtechnik (Ausprägung)
4. Schnittstellen	Anhang zum Pflichtenheft
5. Anforderungen an die Systemtechnik	
6. Anforderungen für die Inbetriebnahme und Einsatz	
7. Anforderungen an die Qualität	
8. Anforderungen an die Projektabwicklung	
Anhang zum Lastenheft	

Tabelle 2.7: Gliederung von Lasten- und Pflichtenheft nach VDI/VDE-Richtlinie 3694

2.12 Analyse der Methoden und Beschreibungsmittel der Automatisierungstechnik

Zusammenfassend lassen sich folgende Feststellungen bezüglich der Analyse der bisher vorgestellten Beschreibungsmittel und Methoden treffen (siehe auch Tabelle 2.8):

1. Da die Methoden-Axiome *Kausal-* und *Temporalprinzip* im Zusammenhang mit der in dieser Arbeit zu entwickelnden Methode nur dann als erfüllt gelten, wenn alle Steuerungsfunktionen beschrieben werden können, kann somit nur eine Kombination der in Tabelle 2.2 aufgeführten Beschreibungsmittel diese Methoden-Axiome erfüllen. Die einzelnen Beschreibungsmittel erfüllen somit diese Methoden-Axiome nur eingeschränkt.
2. Alle vorgestellten Beschreibungsmittel erfüllen das Methoden-Axiom *Anwendbarkeit*, da sie in der Praxis angewandt werden und entsprechende Werkzeuge kommerziell verfügbar sind.
3. Keines der vorgestellten Beschreibungsmittel erfüllt die Methodenaxiome *Strukturprinzip*, *System-*, *Software-* und *Projekt-Dekompositionsprinzip*. Ausnahme ist das Softwarekomponentenmodell, welches aber nur in Verbindung mit einem weiteren Beschreibungsmittel wie die UML das Methoden-Axiom *Software-Dekompositionsprinzip* erfüllt.
4. Die vorgestellten UML-basierten Methoden zeigen die Stärken der UML bezüglich der Erfüllung des *Dekompositionsprinzip* (*Projekt-* und *Software-Dekompositionsprinzip*). Unberücksichtigt bleibt hierbei lediglich die physische Sicht auf die Steuerung. Die mangelnde Berücksichtigung von Aktivitätsdiagrammen (*programmiertechnisch-orientiert*) und Zustandsdiagrammen (*zustandsorientiert*)

im Sinn eines integrierten Beschreibungsmittels führt dazu, dass diese Methoden die Beschreibbarkeit der Steuerungsfunktionen nur eingeschränkt erfüllen.

5. Die Produktionsagenten-Methode ist die einzige Methode, die das *Strukturprinzip* bezüglich Steurungen erfüllt.

Prinzipiell zeigen diese Feststellungen (z.B. Festellung 1. und 4.), dass prinzipiell nur eine Kombination von Beschreibungsmitteln und Methoden zu der hier zu entwickelnden Methode zur Spezifikation von Steuerungen führt, die die gestellten Anforderungen aus den Tabellen 2.1 und 2.2 erfüllt.

Der Standard IEC-61131 basiert auf diesem konzeptionellen Ansatz. Bisher haben sich die Teilsprachen dieses Standards allerdings an den Programmiersprachen zur Implementierung von Ablaufsteuerungen orientiert. Zudem sind diese Teilsprachen substituierend angelegt und nicht ergänzend wie z.B. das Sichten-Konzept der UML. Auch aktuelle Erweiterungen zur Programmierung von Achssteuerungen führen aufgrund der geringfügigen Abstraktion nicht zu einem Beschreibungsmittel zur Spezifikation von Steuerungen, sondern zu einer stärkeren Integration von aufbau-orientierten Beschreibungsmitteln in eine gemeinsame Programmiersprache [PLCOPEN_03].

Einen anderen Ansatz stellt die UML dar, die explizit unterschiedlich orientierte Beschreibungsmittel zu einem integrierten Beschreibungsmittel zusammengefasst hat. Feststellung 3 und 4 (s.o.) führen unmittelbar zu dem Schluss, dass von den hier diskutierten Beschreibungsmitteln nur die UML das Potenzial hat, alle Methoden-Axiome zu erfüllen. Die auf der UML basierenden Methoden zeigen, dass gerade die Erweiterungsmechanismen der UML hiefür die beste Voraussetzung bieten.

Da aber das Methoden-Axiom *System-Dekomposition* von keinem Beschreibungsmittel bzw. keiner Methode berücksichtigt wurde und das Softwarekomponentenmodell sowie die PA-Methode in eine zu entwickelnde Methode zu integrieren wären, sind weder die ROOM- noch die DOUGLASS-Methode als Grundlage dieser Methodenentwicklung geeignet. Die durchzuführenden Änderungen an diesen etablierten Methoden wären so fundamental, dass dies einer Methoden-Neuentwicklung gleich käme.

Methodenaxiome **Beschreibungsmittel/ *Methoden***	**Struktur**	**Dekomposition**			**Temporal**	**Kausal**	**Anwend-barkeit**	**Prüf-barkeit**
		System	**Soft-ware**	**Projekt**				
Verknüpfungsorientiert	-	-	-	-	-	(X)	X	X
Programmiertechnisch-orientiert	-	-	-	-	(X)	(X)	X	X
Zustandsorientiert	-	-	-	-	(X)	(X)	X	X
Aufbauorientiert	-	-	-	-	(-)*	(X)	X	X
Softwarekomponentenmodell	-	-	X	-	-	-	X	-
ROOM-Methode (UML)	-	-	(X)	X	(X)	(X)	(X)	X
Douglass-Methode (UML)	-	-	(X)	X	(X)	(X)	-	(X)
Produktionsagenten-Methode	X	-	-	(X)	-	-	(X)	-

Tabelle 2.8: Bewertung der vorgestellten Methoden bzw. Beschreibungsmittel anhand der Systemaxiome (-=nicht erfüllt, (x)=eingeschränkt erfüllt, x=erfüllt, *=kann aufgrund der Vielzahl der Beschreibungsmittel nur im Mittel bewertet werden)

Aus diesem Grund wird hinsichtlich der Konzeption der Methode ausschließlich die UML als ihr Beschreibungsmittel gefordert. Hierbei wird die UML als implementierungsunabhängiges Beschreibungsmittel (*Platform Independed Model*) verstanden, da nur auf diese Weise der Heterogenität bezüglich der Programmierung von Automatisierungsgeräten Rechnung getragen werden kann.
Darüber hinaus wird die PA-Methode der Ausgangspunkt der Konzeption sein, da sie als einzige das Methoden-Axiom *Strukturprinzip* erfüllt und zudem nur ein Begriffsmodell für den Produktionsagenten definiert ist, welches eine flexible Basis für ein neu zu entwickelndes Beschreibungsmittel bildet. Feststellung 1 und 4 fordern unmittelbar die Integration von Zustands- und Aktivitätsdiagrammen zur vollständigen Beschreibbarkeit der Steuerungsfunktionen (*Temporal-* und *Kausal-Prinzip*). Zu Erfüllung der Prüfbarkeit wird die Entwicklung von Verfahren zur Prüfung der syntaktischen Konsistenz und der semantischen Widerspruchfreiheit gefordert, wie sie auch in ROOM- und Douglass-Methode zu finden sind.
Um das Softwarekomponentenmodell integrieren zu können, müssen Modelle für das des *System-Dekompositionsprinzips* gefunden werden. Um gleichzeitig die Anwendbarkeit zu berücksichtigen, sind im Folgenden Begriffsmodelle und Modellvorstellungen aus der Steuerungstechnik bzw. Automatisierungstechnik zu finden, die dieses Methoden-Axiom beinhalten.
In den Kapiteln 2.10 und 2.11 wurden die Eigenschaften objektorientierter Vorgehensmodelle bzw. die Spezifikationsdokumente Lasten- und Pflichtenheft vorgestellt. Diese sind als zusätzliche Anforderungen für die Konzeption der zu entwickelnden Methode zu verstehen und beziehen sich ausschließlich auf das Vorgehensmodell. Sie sind somit als Teil der Methoden-Axiome *Projekt-Dekompositionsprinzip* und *Anwendbarkeit* anzusehen.

3 Wissenschaftliche Grundlagen zur Beschreibung von Steuerungen

Wie in Kapitel 2 festgestellt und gefordert, sollen in diesem Kapitel die Modellvorstellungen und Begriffsmodelle zur Beschreibung des *System–Dekompositionsprinzips* von Steuerungen vorgestellt werden, die als Grundlage der Erweiterung des zu entwickelnden Beschreibungsmittels dienen. Zentrale Begriffsmodelle sind in diesem Zusammenhang das *Ebenenmodell*, das *Steuerungsprozessmodell*, das *Funktionsblockmodell* und das *Modulare Automatisierungsgerät*. Darüber hinaus werden von diesen Modellen abhängige Modelle wie das Begriffsmodell des *Kommunikationssystems* und das Echtz*eitmodell* dargestellt.

3.1 Begriffsmodelle zur Beschreibung des System-Dekompositionsprinzips von Steuerungen

3.1.1 Das *Ebenenmodell*

Die System-Dekomposition von Automatisierungssystemen bzw. deren Steuerungen lässt sich durch ein Begriffsmodell aus fünf Ebenen bestehend beschreiben [VDI_3687, VDI_3694, LAUBER_99, SCHERFF_99]. Diese Ebenen sind gekennzeichnet durch die Funktionen, die durch Anwendungsprozesse realisiert werden. Dies sind zum einen die Steuerungsfunktionen (siehe Tabelle 2.2) und zum anderen die sensorischen bzw. aktorischen Funktionen in der *Prozessebene*. Darüber hinaus gibt es die Funktionen der überlagerten Produktionssteuerung in der *Fertigungs*- und in der *Fabrikleitebene* (z.B. *Manufacturing Execution System*). Hierbei geht das Ebenenmodell von einem hierarchischen Ansatz aus, d.h. dass die Anwendungsprozesse einer höheren Ebene Dienste von Anwendungsprozessen aus der nächst tieferen Ebene nutzen (siehe auch *Begriffsmodell des Produktionsagenten* in Bild 2.2).
Die Funktionen von Anwendungsprozessen in diesem Kontext benötigen zur Erfüllung ihrer Funktionalität bestimmte Eigenschaften der Automatisierungsgeräte. Somit werden durch das Ebenenmodell auch Typen von Automatisierungsgeräten definiert (*Gerätekomponenten bzw. -typen*).

<table>
<tr><th>Ebenen</th><th>Funktionen</th><th>Gerätetypen</th><th>Kommunikations-systeme</th><th>Kommunikations-objekte</th></tr>
<tr><td>Fabrikleitebene</td><td rowspan="2">z.B. ERP, MES</td><td rowspan="2">Standard IT-Geräte</td><td rowspan="2">Fabrikbus/
z.B. Ethernet</td><td rowspan="2">Elektronische Dokumente, Dateien</td></tr>
<tr><td>Fertigungsebene</td></tr>
<tr><td>Prozessführungs-ebene</td><td>Maschinen- und Betriebsdatenverarbeitung (Nebenfunktion), Bedienersteuerung</td><td>PFK</td><td rowspan="2">Systembus/
z.B. Profibus,
Ethernet</td><td rowspan="2">Agententelegramme, Nachrichten/Ereignisse, Status- und Konfigurationswerte</td></tr>
<tr><td>Prozesssteuerungs-ebene</td><td>Achs-, Bewegungs- Bediener- und Ablaufsteuerung</td><td>PNK, Intelligente Feldgeräte</td></tr>
<tr><td>Prozessebene (Feldebene)</td><td>Ein- und Ausgabe von Prozessvariablen</td><td>Sensoren, Aktoren, Bus-Klemmen</td><td>Feldbus/
z.B. Profibus, Controller Area Network (CAN), RS232/485</td><td>Ereignisse, Echtzeit-Prozessdaten, Synchronisationssignale, Alarmmeldungen, Status- und Konfigurationswerte</td></tr>
</table>

Tabelle 3.1: Ebenenmodell der Automatisierungstechnik nach VDI/VDE 3687 [SCHERFF_99, HOANG_99]

Prozessnahe Komponenten (PNK) sind Gerätekomponenten, die über ein sog. Echtzeit-Betriebssystem verfügen, das für ein deterministisches Verhalten von Anwendungsprozessen sorgt (z.B. das Einhalten

von Echtzeitschranken). Darüber hinaus können Visualisierungsfunktionen vorhanden sein. In der Regel ist aber der Speicherplatz stark eingeschränkt (z.B. SPS, Computer Numerical Control (CNC)). *Prozessferne Komponenten* (PFK) verfügen in der Regel über ein Standard IT-Betriebssystem (z.B. Windows®) und zusätzlich über Datenbank- und Visualisierungsfunktionen (z.B. IPC).

Einfache Feldgeräte (z.B. Sensoren, Aktoren, Bus-Klemmen) haben keine Steuerungsfunktion, sind aber elementar für die Ausführung von Steuerungen, da sie die Schnittstelle zum technischen Prozess darstellen. So genannte *Intelligente Feldgeräte* gehören zu den Gerätekomponenten, die Ebenenübergreifend zu verstehen sind. Sensorische bzw. aktorische Funktionen sind verbunden mit Steuerungsfunktionen (z.B. Abstandssensor mit eigener Grenzwertüberwachung).

Steuerungen sind prinzipiell als *verteilte Systeme* zu verstehen (siehe *Strukturprinzip* in Tabelle 2.1). Daraus folgt, dass die Gerätekomponenten durch unterschiedliche Kommunikationssysteme miteinander verbunden sind, deren Eigenschaften ebenfalls durch das Ebenenmodell bestimmt werden. Typischerweise wird von einem *Feldbus* (z.B. Profibus) erwartet, dass er unter Beachtung von harten Echtzeitschranken relativ kleine Datenmengen deterministisch überträgt [BENDER_92, REISSEN_98]. Der *Fabrikbus* (i.d.R. Ethernet mit TCP/IP-Protokoll) dagegen überträgt sehr große Datenmengen ohne signifikante zeitliche Einschränkungen [BEN_00, TERW_01, DUMSKY_02].

3.1.2 Das *Steuerungsprozess-Modell*

Unter einem *Steuerungsprozess* soll im Folgenden ein Anwendungsprozess verstanden werden, der eine oder mehrere Steuerungsfunktion realisiert. Steuerungsfunktionen sind in Tabelle 2.2 dargestellt und werden hier aufgrund der geforderten Berücksichtigung der PA-Methode durch die *Agentensteuerung* ergänzt. Die Agentensteuerung ist die Steuerung, die durch den *Agenten-Kopf* bzw. PA-Kopf realisiert wird (siehe Kapitel 2.9.3 und 2.12).

Da Steuerungsprozesse über so genannte *Kommunikationsbeziehungen* Datenobjekte (*Kommunikationsobjekte*) untereinander austauschen können, ist das Steuerungsprozess-Modell ein wichtiges Begriffsmodell zur Beschreibung des *System-Dekompositionsprinzips* von Steuerungen (siehe Bild 3.1). Kommunikationsverbindungen sind im Regelfall *Punkt-zu-Punkt-Beziehungen, d.h.* zwei Steuerungsprozesse tauschen exklusiv Datenobjekte miteinander aus. Gerade aber Multi-Agenten-Systeme erfordern es, dass Steuerungsprozesse - also Agenten-Köpfe - mittels *Mehrpunkt-Verbindungen* (*Multicast* und *Broadcast*) kommunizieren können (siehe *Referenzfallstudie* in Kapitel 10). Kommunikationsbeziehungen sind in diesem Kontext als Abstraktion eines spezifischen Kommunikationssystems zu verstehen (*Transparenz*).

Aktiviert werden Steuerungsprozesse durch das Empfangen von Nachrichten (*Ereignisgesteuert*) oder zyklisch mittels fester Zeiteinstellungen (*Zeitgesteuert*). Als Beispiel kann hier das *Task*-Konzept der IEC-61131-3 dienen, wobei ein *Task* als Steuerungsprozess zu verstehen ist [REISSEN_98, HOANG_99].

Eine erweiterte Betrachtung des Steuerungsprozess-Modells weist diesem eine *Schnittstelle* und eine *Rolle* zu. Eine Schnittstelle beschreibt die verfügbaren Dienste bzw. die zu empfangenen Nachrichten

eines Steuerungsprozesses, die er anderen Steuerungsprozessen anbietet. Die Rolle beschreibt ein bestimmtes Verhalten eines Steuerungsprozesses, das dieser für einen bestimmten Zeitraum einnimmt und dabei eine bestimmte Schnittstelle anbietet. Elementare Rollen sind z.B. die des *Clients* und des *Servers*. Die Rolle, die ein Automatisierungsgerät in einem Kommunikationssystem einnimmt, ist von der Rolle des Steuerungsprozesses zu unterscheiden (*Geräterolle*) [BENDER_92, TRAUB_02]. Der *Rollen*-Begriff findet sich auch in der UML wieder, z.B. kann jede Art von Objekt bzw. Instanz in einem Kollaborations- bzw. Sequenzdiagramm eine bestimmte Rolle einnehmen. Semantisch besitzt die Rolle die Eigenschaften einer Klasse und eines Objektes (siehe Bild 0.2).

Die Schnittstelle eines Steuerungsprozesses zum Kommunikationsprozess wird *Application Layer Interface* (ALI) genannt und ist technologie-abhängig. Das *Abstract Communication Service Interface* (ACSI) ist durch die Norm IEC 61850-1 als technologie-unabhängig definiert, aber bisher in der Praxis nicht umgesetzt [REISSEN_98, BENDER_92].

Zusammenfassend kann festgestellt werden, dass das Steuerungsprozessmodell aufgrund seiner Ableitung aus dem Begriff des Anwendungsprozesses nicht nur das *System*-, sondern auch das *Software-Dekompositionsprinzip* einer Steuerung beinhaltet.

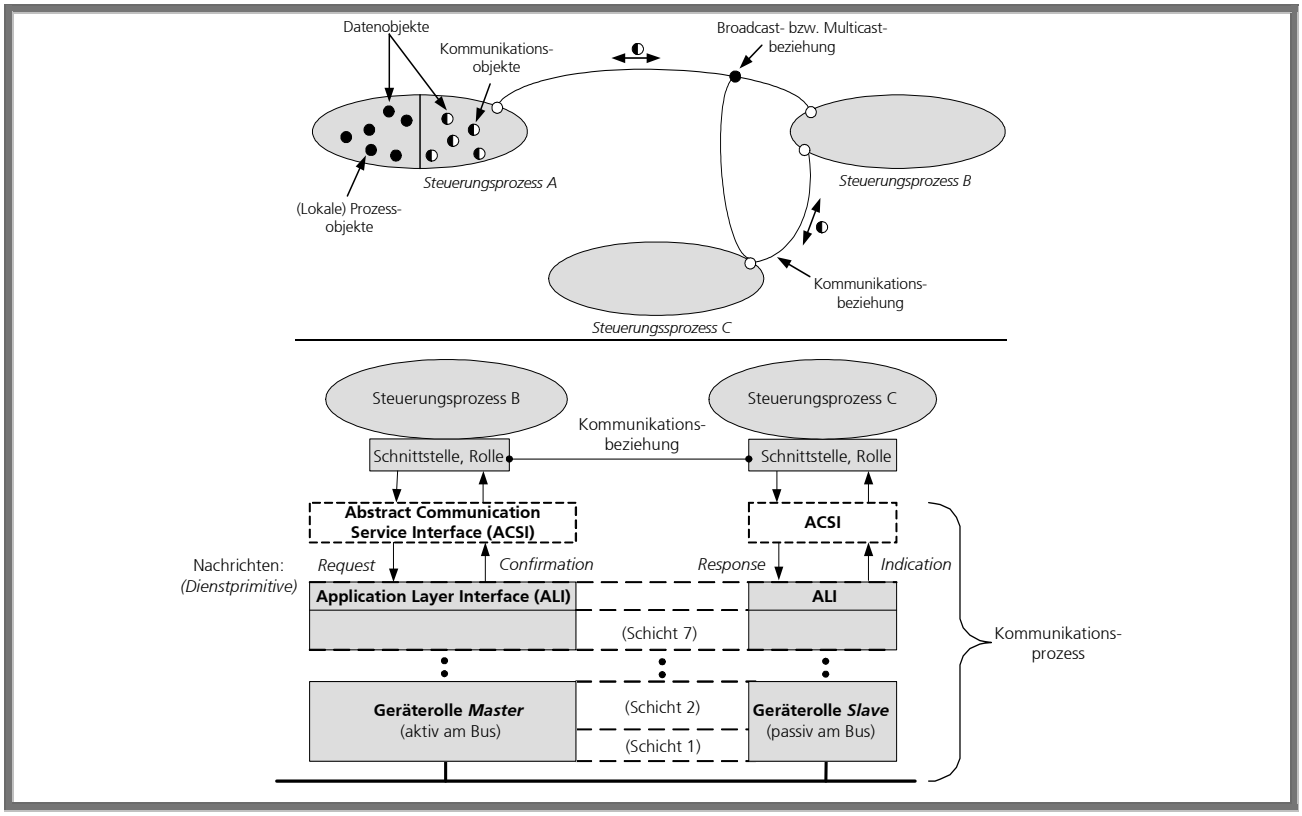

Bild 3.1: Steuerungsprozesse und Kommunikationsbeziehungen im Kontext des Schichtenmodells nach ISO 7498 [BENDER_92, REISSEN_98, HOANG_99]

3.1.3 Analyse der Datenmodellierung in der Steuerungstechnik

Kommunikationsbeziehungen beschreiben die statische Verbindung zwischen Steuerungsprozessen, über die bidirektional Datenobjekte ausgetauscht werden können. Ist die Kommunikation zwischen Steuerungsprozessen Nachrichtenbasiert, so werden Ereignisse und Datenobjekte gemeinsam übertra-

gen, d.h. in einem Telegramm des darunter liegenden Kommunikationssystems [BENDER_92, SELIC_94]. Als elementare Nachrichten eines Kommunikationssystems in einer Steuerung sind die sog. *Dienstprimitiven* anzusehen (siehe Bild 3.1).

Prinzipiell sind Datenobjekte in *Kommunikationsobjekte*, also Datenobjekte, die zwischen Steuerungsprozessen ausgetauscht werden und deren Datentypen diesen Prozessen bekannt sein müssen, und ausschließlich von einem Steuerungsprozess verwaltete *Prozessobjekte* zu unterscheiden [OSACA_96]. Dabei nimmt die Datengröße der Datenobjekte zur Prozessebene hin ab und ihre Anforderung an den zeitlichen Determinismus des Kommunikationssystems nimmt zu (siehe Tabelle 3.2).

Darüber hinaus sind die Datentypen von Datenobjekten in *elementare* und aus diesen *zusammengesetzte Datentypen* zu unterscheiden (siehe z.B. Datentypen der Norm IEC 61131). Bezüglich der elementaren Datentypen ist weiterhin in plattformunabhängige Datentypen (z.B. OCL-Datentyp *Integer*) und implementierungsabhängige (z.B. IEC-61131-Datentyp *INT*, *DINT*, *LINT*, *USINT*) zu differenzieren.

Aus Sicht der UML lässt sich ein Kommunikationsobjekt als Instanz einer Klasse verstehen, die ausschließlich Attribute enthält (*Transportobjekte bzw. -klassen* [IHNS_00]). Diese Attribute können wiederum durch Transportklassen beschrieben sein oder durch elementare Datentypen der UML. Prozessobjekte dagegen können auch über eigene Operationen verfügen.

Nachrichtenarten		**Echtzeitschranke**	**Zusätzliche Spezifikation**	**Ebenen**
Kommunikationsobjekte	Ereignisse (z.B. Synchronisation, Alarm)	Hart	Zeitstempel	Prozesssteuerung, Prozess
	Agenten-Telegramme	Weich	-	Prozessführung
Kommunikations- und Prozessobjekte	Dateien (z.B. Arbeitspläne)	Keine Echtzeitschranke	-	Prozessführung
	Konfigurations- und Statuswerte	Weich	Zeitstempel	Prozesssteuerung, Prozess
	Prozessvariablen (Mess- und Stellgrößen)	Hart	Physikalische Einheiten, Wertebereich, Zeitstempel	Prozesssteuerung, Prozess

Tabelle 3.2: Kommunikations- und Prozessobjekte für Steuerungen in Fertigungssystemen [BENDER_92, IEC61850-7-2, OSACA_96, WEISSHA_99]

3.1.4 Analyse des Beschreibungsmittels *Funktionsblock* und Ableitung des Begriffsmodells

Funktionsblöcke bzw. *–bausteine* sind die am weitesten verbreiteten Beschreibungsmittel bezüglich des Steuerungsentwurfs, d.h. zur Beschreibung des *System-Dekompositionsprinzips* von Steuerungen. Hierbei gibt es aber eine Reihe spezifischer Ausprägungen des Funktionsblockes bzw. –bausteins in Form von unterschiedlichen Standards und Techniken (z.B. die Norm IEC-61499 [IEC61499-1, ENSTE_01, BRAATZ_03].

Diese Ausprägungen sind in ein gemeinsames *Begriffsmodell des Funktionsblocks* als kleinsten gemeinsamen Nenner zu überführen, das dann die Grundlage zur Integration in ein zu entwickelndes Beschreibungsmittel bildet. BRAATZ hat gezeigt, dass sich dieses Begriffsmodell auf Basis der Eigenschaften von Softwarekomponententechniken (Tabelle 2.4) als aktives Softwarekomponentenmodell mittelkörniger Granularität beschreiben lässt. (siehe Tabelle 3.3, [BRAATZ_03])

Aufgrund der Beschreibung als aktives Softwarekomponentenmodell lässt sich das Begriffsmodell des Funktionsblocks als dekomponiertes Modell des Steuerungsprozesses verstehen. Die Schnittstellenbeschreibung des Funktionsblocks in Kombination mit der inneren Strukturierung (Ausführungskontrolle und Algorithmen) geben der *Black-Box*-Betrachtung des Steuerungsprozesses eine dekomponierte Beschreibung. Somit beschreibt das Begriffsmodell des Funktionsblocks den Übergang vom *System*- zum *Software-Dekompositionsprinzip*.

FBT / KE		IEC-61131	ACPLT/FB	IDA-Block	IEC-61499
Komponenten-Implementierung (Binärcode)		Nein	**Nein**	**Nein**	**Nein**
Kommunikationskonzept		*keine Verteilung*	Nachrichtenkanäle	Nachrichtenkanäle	Nachrichtenkanäle
Schnittstellenbeschreibung		Attribute (Daten)	**Nachrichten, Daten**, **Ereignisse**	**Nachrichten**, **Daten**, Methoden	**Daten**, **Ereignisse**
Ortstransparenz		*Keine Verteilung*	**ja**	**ja**	**ja**
Aktivität		passiv	**aktiv**	**aktiv**	**aktiv**
Verhaltensbeschreibung	**Ausführungskontrolle**	extern durch SFC	generisch	*nicht spezifiziert*	Execution Control Chart (ECC)
	Algorithmen	Structured Text, Instruction List	ANSI C	*nicht spezifiziert*	Structured Text
Plattform		SPS	ACPLT/KS, -/OV	IDA-Laufzeitsystem	*nicht spezifiziert*
Granularität		feinkörnig	**mittelkörnig**	**mittelkörnig**	**mittelkörnig** (hierarchisierbar)

Tabelle 3.3: Ableitung des Begriffsmodell des Funktionsblocks (FBT=Funktionsblocktechniken und –Spezifikationen, KE=Komponenteneigenschaften, Eigenschaften des Begriffsmodells des Funktionsblocks sind fett gedruckt)

3.1.5 *Strukturprinzip* und *System-Dekompositionsprinzip* eines Automatisierungssystems in der physischen Sicht

In der physischen Sicht beschreibt die *Modellvorstellung des Automatisierungssystems* im Kontext dieser Arbeit ein Produktionssystem bzw. einen Teil eines solchen Systems (z.B. eine Werkzeugmaschine, ein Transportfahrzeug). Es wird also unterschieden in *Produktionssystem* und *Produktionssubsystem*. Eine weitere Stufe der System-Dekomposition in *Automatisierungsgeräte* bzw. *Gerätekomponenten* ist notwendig, wenn die Steuerung als das zu spezifizierende System betrachtet wird. Diese Modellvorstellung beschreibt somit zusätzlich die Verteilung von Gerätekomponenten im Raum bzw. im räumlich verteilten, technischen Prozess. Dies ist aus Sicht der Steuerung eine wichtige Eigenschaft des Systems, da diese Verteilung darüber entscheidet, welche Prozessvariablen durch die entsprechenden Feldgeräte gelesen bzw. ausgegeben werden können.

Aktuelle Forschungsarbeiten fokussieren einen weiteren Schritt der Dekomposition in Sinne des *Software-Dekompositionsprinzips*. Zu diesem Zweck soll ein technologie-unabhängiges *Gerätemodell* entwickelt werden, das neben den grundlegenden Funktionen des Gerätes auch die durch Software implementierten und gegebenenfalls zu programmierenden Funktionalitäten beschreibt (siehe hierzu z.B. [SIMON_02, RIEDL_02]). Ansatz dieser noch nicht abgeschlossenen Arbeiten ist die Zusammenführung

von parameterorientierten Beschreibungen (z.B. Geräte-Stamm-Dateien (GSD)) und komplexen Geräte-Beschreibungen (z.B. *Device Description Language* (EDDL) und *Field Device Configuration Markup Language* (FDCML)) [EDDL_03, FDCML_03]. Vereinfachend lässt sich aber bereits feststellen, dass dieses Gerätemodell durch eine modulare Beschreibung der Hardware und eine Kapselung der Software-Funktionen durch Funktionsblöcke gekennzeichnet ist. Nach DOEBRICH lässt sich hieraus eine einfache grafische Darstellung ableiten, die als Begriffsmodell *Modulares Automatisierungsgerät* eingeführt werden soll [DOEB_00].

Aus Sicht der Spezifikation einer Steuerung ist das *Verarbeitungs- und Rechenmodul* entscheidend, da dieses Modul den Funktionsblock aufnimmt und ihm, je nach Gerätekomponente, entsprechende Schnittstellen (*Anwender-Schnittstellen* und ALI) zur Verfügung stellt. Darüber hinaus definieren die Module auch den *offered QoS* der Gerätekomponente (siehe Tabelle 3.4 und Bild 3.2).

Gerätemodell	Modul	Funktion
Modulares Automatisierungsgerät (Begriffsmodell)	Kommunikationsmodul	Stellt dem Funktionsblock die Dienste der angeschlossenen Kommunikationssysteme zur Verfügung (Schnittstellen ALI oder ACSI) und definiert $\mathbf{t_{V,SP}}$ (offered QoS) als maximale Verzögerungszeit, d.h. Wandlungs- und Verarbeitungszeit eines Gerätes, das eine Nachricht als Eingang erhält und nach der Verzögerungszeit mit einer weiteren Nachricht antwortet.
	Verarbeitungs- und Rechenmodul	Speicher und Prozessorleistung, die dem Funktionsblock zur Verfügung steht. Diese kann programmierbar oder fest eingebettet sein.
	Prozessgrößen-Verarbeitungsmodul	Stellt dem Funktionsblock die Dienste zum Einlesen und Ausgeben der Prozessvariablen (sensorisch und aktorisch) zur Verfügung.
	Organisationsmodul	Beinhaltet im Wesentlichen den *Prozess-Scheduler* und definiert die minimale Zykluszeit $\mathbf{t_{Zykl,SP}}$ (offered QoS) eines Steuerungsprozesses.

Tabelle 3.4: Beschreibung des Begriffsmodells *Modulares Automatisierungsgerät* [DOEB_00, SCHERFF_99]

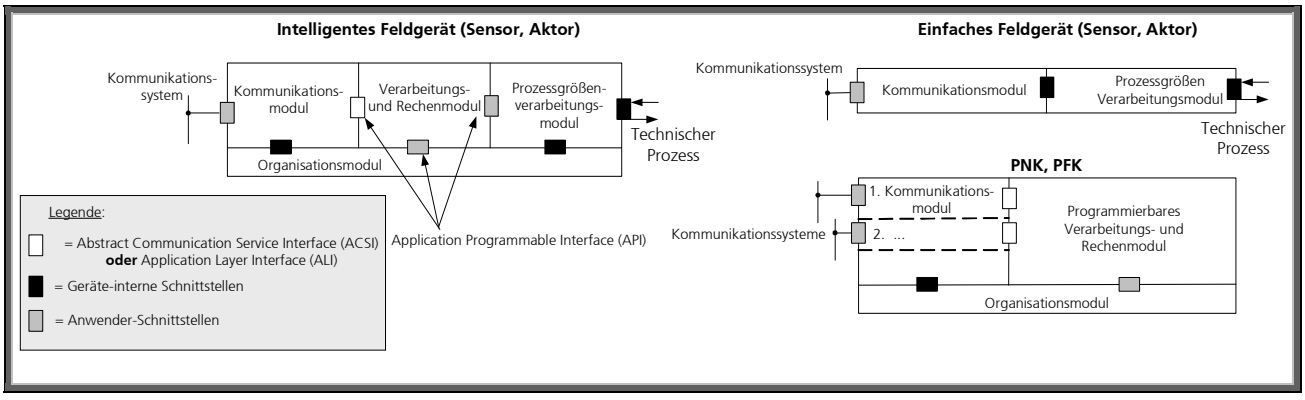

Bild 3.2: Grafische Darstellung des *Modularen Automatisierungsgeräts* [DOEB_00]

3.1.6 Begriffsmodell des Kommunikationssystems

Bedingt durch die räumliche Verteilung von Gerätekomponenten im Produktionssystem beschreibt das hier eingeführte *Begriffsmodell des Kommunikationssystems* das System-Dekompositionsprinzip einer entsprechenden Steuerung. Dieses Begriffsmodell hat dabei keinerlei Einfluss auf das *Kausalprinzip* einer Steuerung, aber durch entsprechende Verzögerungszeiten (*Offered QoS*) der Telegramme wird das *Temporalprinzip* beeinflusst [REISSEN_98, HOANG_99, SCHERFF_99].

Aus Sicht eines zu entwickelnden Beschreibungsmittels für Steuerungen ist ein Kommunikationssystem anhand seiner Kopplung mit dem entsprechendem Steuerungsprozess bzw. Funktionsblock zu unter-

scheiden (siehe Kapitel 3.1.2 und 3.1.4). Werden von einem Steuerungsprozess bzw. Funktionsblock direkt die Dienste (*Dienstprimitive*) eines Kommunikationssystems aufgerufen, um entsprechende Kommunikationsobjekte von den angeschlossenen Gerätekomponenten zu lesen oder auf sie schreibend zuzugreifen, wird vom *Synchronen Zugriff* gesprochen (*Synchrones Kommunikationssystem*). Hierbei ist der Zugriff exklusiv und mit der Zeitverzögerung *Buszykluszeit* $\mathbf{t_{Zykl,Kom}}$ behaftet [REISSEN_98, ROSTAN_02]. Dies ist typisch für Feldbusse (siehe Tabelle 3.1).

Die Kommunikationsbeziehung im Kontext des Steuerungsprozess-Modells beschreibt dagegen den *transparenten Zugriff* auf ein Kommunikationssystem (siehe Bild 3.1). Das Begriffsmodell des Funktionsblock setzt dies mittels Nachrichtenkanälen um, wobei diese Teil der Plattform sind und somit in der Prozesssicht des Funktionsblocks transparent bleiben (siehe Kapitel 3.1.4). Dies setzt aber eine entsprechende Middleware-Implementierung voraus (z.B. IDA-Laufzeitsystem), die eine Eigenschaft der physischen Sicht ist.

Tabelle 3.5 stellt das Begriffsmodell des Kommunikationssystems mit seinen Eigenschaften dar, wobei der Tabelle eine Gliederung nach Art des Zugriffes und nach der relevanten Sicht bezüglich einer Eigenschaft zu Grunde liegt.

Methoden-Axiome / **Sicht/Zugriff**		**System-Dekompositionsprinzip**	**Temporalprinzip**
		Eigenschaft	**Eigenschaft**
Logisch	**Synchroner Zugriff**	Application Layer Interface *oder* Abstract Communication Service Interface	Maximale Ausführungszeit eines Dienstes $\mathbf{t_{V,OP}}$
	Transparenter Zugriff	Kommunikationsbeziehung (Punkt-zu-Punkt, Multicast bzw. Broadcast)	-
Physisch	**Synchroner Zugriff**	Anzahl der Geräte am Kommunikationssystem (n_{AG})	Buszykluszeit $\mathbf{t_{Zykl,Kom}}$
		Maximal erlaubte Anzahl der Geräte (n_{AGmax})	
		Feldbustyp	
		Topologie (Mehr- oder Zweipunkttopologie)	
		Geräterolle (Master, Slave)	
	Transparenter Zugriff	Nachrichtenkanal	Max. Zeitverzögerung einer Nachricht auf einem Nachrichtenkanal $\mathbf{t_{V,Kanal}}$

Tabelle 3.5: Begriffsmodell des Kommunikationssytems

3.1.7 *Echtzeit-Modell* der System-Dekomposition

Da Steuerungen als *Echtzeitsysteme* verstanden werden, ist das Echtzeit*modell* im Kontext des System-Dekompositionsprinzips essentiell zur Beschreibung des Temporalprinzips. Die bisher im Zusammenhang mit der Beschreibung des *Modularen Automatisierungsgerätes* und des *Begriffsmodell des Kommunikationssystems* eingeführten temporalen Größen (z.B. $\mathbf{t_{Zykl,SP}}$) werden durch das Echtzeitmodell algebraisch und damit formal verknüpft.

Es gelten hierbei zwei grundsätzliche Voraussetzungen bezüglich der Interpretation von Zeit in einer Steuerung [REISSEN_98, SCHERFF_99, REINHA_96]:

1. Das Echtzeitmodell ist quantifiziert beschreibbar durch konstante, zum Spezifikationszeitpunkt feststehende, maximale bzw. minimale Zeitangaben. Der *Jitter*, d.h. die maximale Schwankungsbreite einer zugesicherten Zeitangabe, wird nicht berücksichtigt.
2. Das Echtzeit-Modell lässt sich aus Sicht des technischen Prozesses als maximal zulässige Reaktionszeit $\mathbf{t_{Prozess}}$ einer Steuerung bezüglich der Veränderungen im technischen Prozess und als Eigenschaft der Geräte und Kommunikationssysteme im Sinne einer maximalen zeitlichen Verzögerung $\mathbf{t_V}$ beschreiben (*Worst-Case*). Die daraus zu formulierende Ungleichung muss zum Spezifikationszeitpunkt erfüllt sein:

$$\mathbf{t_{Prozess} \geq t_V} \qquad \text{(Gl. 1)}$$

Aufgabe einer Verifikation bzw. einer Prüfung auf semantische Konsistenz ist es somit, zu jedem Zeitpunkt der Spezifikation die Gültigkeit dieser Ungleichung sicherzustellen. Bezieht man nun die bereits eingeführten temporalen Größen ($\mathbf{t_{V,Kanal}}$, $\mathbf{t_{Zykl,SP}}$, $\mathbf{t_{V,SP}}$, $\mathbf{t_{Zykl,Kom}}$, siehe Tabellen 3.4 und 3.5) in diese Ungleichung ein, ergeben sich folgende drei weiteren Ungleichungen. Geräte-interne Verzögerungen, wie z.B. das Laden eines Wertes in den lokalen Speicher können hierbei vernachlässigt werden [REISSEN_98, SCHERFF_99, ROSTAN_02]:

1. *Synchroner Zugriff*: Ein zyklisch gesteuerter Steuerungsprozess (Geräterolle *Master*) und das Kommunikationssystem sind synchronisiert. Zwischen zwei Zyklen des Steuerungsprozesses liegt ein Buszyklus. Es gilt:

$$\mathbf{t_{Prozess} \geq 2 * t_{Zykl,SP} + 3 * t_{Zykl,Kom}} \qquad \text{(Gl. 2)}$$

2. *Transparenter Zugriff*: Ein sendender Steuerungsprozess (z.B. Intelligenter Sensor) greift verzögerungsfrei auf seine Sensorwerte zu und schickt aufgrund des kritischen Wertes einer Prozessvariablen eine Nachricht über einen Kommunikationskanal. Nach Ablauf der Zeit $\mathbf{t_{V,Kanal}}$ erhält der ausgebende Steuerungsprozess (z.B. Intelligenter Aktor) diese Nachricht. Nach Ablauf der Zeit $\mathbf{t_{V,SP}}$ wird dieser Wert im technischen Prozess wirksam und eine Antwort-Nachricht wird gesandt. Folgende Ungleichung beschreibt die Forderung von Gl. 1:

$$\mathbf{t_{Prozess} \geq t_{V,Kanal} + t_{V,SP}} \qquad \text{(Gl.3)}$$

3. Der unter 2. beschriebene *Transparente Zugriff* spezifiziert den elementaren Fall. Zwischen dem sendenden und dem ausgebenden Steuerungsprozess kann aber eine beliebige Anzahl weiterer Steuerungsprozesse in Reihe geschaltet sein bzw. weitere Kommunikationskanäle. Die Verzögerungszeiten addieren sich entsprechend nach Gl. 3:

$$\mathbf{t_{Prozess}} \geq \sum_{\mathbf{x=1}}^{\mathbf{x=m_{K+SP}}} \mathbf{t_{V,Kanal,x} + t_{V,SP,x}} \qquad \text{(GL.4)}$$

Die geschilderten Varianten betrachten ausschließlich harte Echtzeitschranken, da mit $\mathbf{t_{Prozess}}$ ausschließlich die Abläufe im technischen Prozess beschrieben werden. Für den Fall, dass ein Prozess durch weiche Zeitschranken gekennzeichnet ist, können die identischen Ungleichungen angewendet werden. Im Fall der Verletzung einer dieser Ungleichungen ist aber vom Entwickler fallweise zu ent-

scheiden, ob dies zu akzeptieren ist oder ob die Echtzeitschranke bzw. der spezifizierte Mittelwert als verfehlt gilt und entsprechende Massnahmen ergriffen werden müssen.

3.2 Analyse der Begriffsmodelle und Modellvorstellungen und Schlussfolgerungen

Gemäß der Aufgabenstellung wurden in diesem Kapitel Begriffsmodelle und Modellvorstellungen eingeführt, die das System-Dekompositionsprinzip beinhalten. Hierbei kann für alle Modelle festgestellt werden, dass zum einen das System-Dekompositionsprinzip aus zwei verschiedenen Sichten - der physischen Sicht und der Prozesssicht - beschrieben werden kann und dass zum anderen diese Modell ergänzend weitere Methoden-Axiome berücksichtigen. Ausnahme ist hierbei die Analyse der Datenmodellierung aus Kapitel 3.1.3. Diese ist als Detaillierung des Steuerungsprozess-Modells zu verstehen.
Zunächst ist signifikant, dass die *Modellvorstellung des Automatisierungssystems* als einziges Modell das Strukturprinzip berücksichtigt. Allerdings ist diese Modellvorstellung so grundlegend, dass sie in der physischen Sicht des Begriffsmodells des Produktionsagenten bereits enthalten ist (siehe Kapitel 2.9.3 und Tabelle 2.8). Darüber hinaus lassen sich durch die Auswertung der verschiedenen Modelle folgende Feststellungen treffen (siehe auch Tabelle 3.6):

1. Ebenenmodell, Steuerungsprozessmodell und das Begriffsmodell des Funktionsblocks modellieren das System-Dekompositionsprinzip einer Steuerung in der Prozesssicht. Da das Ebenenmodell die Menge aller Steuerungsprozesse in einer Steuerung betrachtet und das Funktionsblockmodell als die nächste Dekompositionsstufe des Steuerungsprozess-Modells gilt, werden alle drei Modelle in der angegebenen Reihenfolge als Modelle der System-Dekomposition in unterschiedlichen Dekompositionsstufen angesehen (siehe Kapitel 3.1.1 , 3.1.2 und 3.1.4).

2. Mit der Einführung unterschiedlicher Dekompositionsstufen durch das Steuerungsprozess-Modell und das Begriffsmodell des Funktionsblocks werden auch zwei Dekompositionsstufen bezüglich des Software-Dekompositionsprinzips eingeführt (siehe Kapitel 3.1.2 und 3.1.4).

3. Bezüglich der physischen Sicht ist das Ebenen-Modell als integratives Modell des System-Dekompositionsprinzips anzusehen, welches also ebenfalls die logische Sicht umfaßt. Das Begriffsmodell *Modulares Automatisierungsgerät* und das Begriffsmodell des Kommunikationssystems werden somit gemeinsam als eine Dekompositionsstufe im Ebenenmodell definiert.

4. Das Echtzeitmodell ist bezüglich des Temporalprinzips als integrierendes Modell zu verstehen. Da es beide Sichten beinhaltet, integriert es zum einen die Modellanteile des Temporalprinzips aus dem Begriffsmodell *Modulares Automatisierungsgerät* und dem Begriffsmodell des Kommunikationssystems und zum anderen wird das Temporalprinzip in das Steuerungsprozess-Modell eingebracht.

Methoden-Axiome/ Sichten / Modelle	Struktur		Dekomposition			Temporal	
			System		Soft-ware		
	Logisch	Physisch	Logisch	Physisch		Logisch	Physisch
Ebenenmodell	-	-	X	X	-	-	-
Steuerungsprozess-Modell	-	-	X	-	(X)	-	-
Begriffsmodell des Funktionsblocks	-	-	X	-	X	-	-
Modellvorstellung des Automatisierungs-systems	-	X	-	-	-	-	-
Modulares Automatisierungsgerät	-	-	-	X	-	-	(X)
Begriffsmodell des Kommunikationssys-tems	-	-	-	X	-	-	(X)
Echtzeitmodell	-	-	-	-	-	X	X

Tabelle 3.6: Auswertung der Modellvorstellungen und Begriffsmodelle (-=nicht erfüllt, (X)=eingeschränkt erfüllt, X=erfüllt)

Aus den aus der Analyse der Modelle abgeleiteten Feststellungen und aus den Schlussfolgerungen bezüglich der Analyse der Beschreibungsmittel und Methoden (siehe Kapitel 2.12), lässt sich die Konzeption der zu entwickelnden Methode ableiten:

- Mit der Integration der PA-Methode in die zu entwickelnde Methode zur Spezifikation von Steuerungen ist das *Strukturprinzip* für die physische und die logische Sicht erfüllt. Dazu wird für die Entwicklungsphase *Anforderungsdefinition* (siehe Kapitel 2.2) ein entsprechender Entwicklungsschritt definiert.

- Mit der UML als Beschreibungsmittel der zu entwickelnden Methode ist die Berücksichtigung des *Projekt-Dekompositionsprinzips* implizit gegeben (siehe Tabelle 2.8). Hierzu gehören auch die in den Kapiteln 2.10 und 2.11 beschriebenen Aspekte. Das semiformale Beschreibungsmittel ermöglicht es darüber hinaus, Verfahren zur *Prüfung* zu formulieren und anzuwenden (*Prüfbarkeit*).

- Die geplante Abbildung der eingeführten Modellvorstellungen und Begriffsmodelle auf die UML schließt nicht nur die Lücke bezüglich der Beschreibung des *System-Dekompositionsprinzips*, sondern erhöht auch den Grad der *Anwendbarkeit* durch die Verwendung von Modellen aus dem Anwendungsgebiet Automatisierungstechnik.

- Zur vollständigen Beschreibung aller Methoden-Axiome sind *Software-Dekompositions-*, *Temporal-* und *Kausalprinzip* in das zu entwickelnde Beschreibungsmittel der Methode zu integrieren. Das *Kausalprinzip* wird durch Verwendung von Zustands- und Aktivitätsdiagrammen berücksichtigt (siehe Kapitel 2.12). Die Integration des Echtzeitmodells in das UML-basierte Beschreibungsmittel erfüllt dann das *Temporalprinzip* (siehe Festellung 4). Festellung 2 führt zur Einbindung des Softwarekomponentenmodells in Form des Begriffsmodells des Funktionsblocks und somit zur Berücksichtigung des *Software-Dekompositionsprinzips* (siehe auch Tabelle 2.8)

4 Konzeption des Vorgehensmodells der Methode zur Spezifikation von Steuerungen

Basierend auf dem Begriff der *Spezifikationsmethode* aus Kapitel 2.2 soll in diesem Kapitel das Vorgehensmodell der objektorientierten Methode zur Spezifikation von Steuerungen konzeptioniert werden. Hierbei werden ausschließlich die Methoden-Axiome bzw. die mit ihnen zusammenhängenden Schlussfolgerungen berücksichtigt, die die Konzeption des Vorgehensmodells beeinflussen bzw. die Methode als Ganzes betreffen (siehe Kapitel 2.12 und Bild 4.1). Für die in dieser Arbeit zu entwickelnde Methode wird der Name ODEMA (*Object-oriented method for Developing Technical Multi-Agent-Systems)* bzw. ODEMA-Methode verwendet, da dieser Namen bereits in die Literatur eingeführt wurde (siehe Kapitel 1.2).

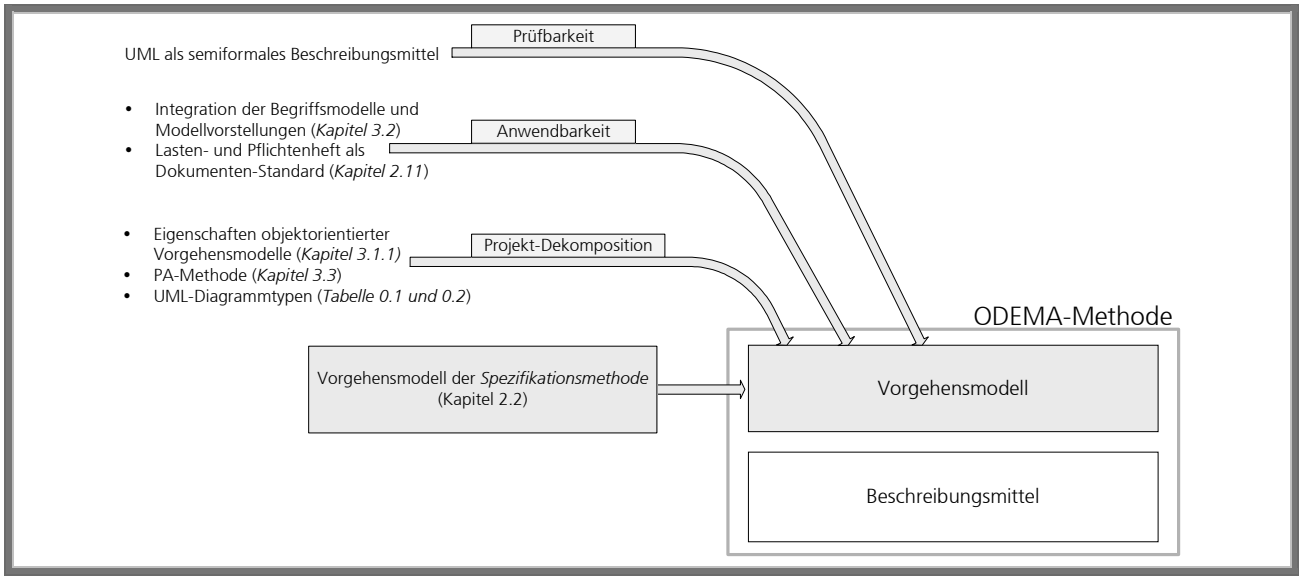

Bild 4.1: Einflussfaktoren zur Konzeption des Vorgehensmodells der ODEMA-Methode

4.1 Konzeption des Projekt-Dekompositionsprinzips

4.1.1 Konzeption eines iterativ-inkrementellen Vorgehensmodells

Das Vorgehensmodell auf Basis der Definition der *Spezifikationsmethode* aus Kapitel 2.2 ist im Kontext der objektorientierten Vorgehensmodelle als sog. *Wasserfallmodell* anzusehen, da die Spezifikation bzw. Entwicklung nacheinander - durch vollständiges Ausführen einer Entwicklungsphase - erfolgt. Die Möglichkeit des Rücksprunges ist hierbei nur der Ausnahmefall. Um die Vorteile eines iterativ-inkrementellen Vorgehens zu nutzen, muss zunächst die Iteration in zwei verschiedenen Varianten eingebracht werden.

Als erstes ist eine Iteration so zu verstehen, das an ihrem Ende ein Inkrement der Steuerung abgeschlossen wird (*Makro-Iteration*). Hierzu ist es notwendig, dass ein Rücksprung aus der Entwicklungsphase *Implementierung und Test* in die Entwicklungsphase der *Anforderungsdefinition* zugelassen wird, um nach Abschluss eines Inkrementes die Definition eines weiteren zu ermöglichen. Die direkten Rücksprünge aus Entwicklungsphasen entfallen hierbei.

Um innerhalb der Makro-Iteration durchgängig mit korrekten Spezifikationen zu arbeiten, ist vor Verlassen einer Phase zu prüfen, ob die Spezifikationen im Rahmen eines semiformalen Beschreibungsmittels korrekt sind (*Prüfbarkeit*). Ist dies nicht der Fall, so muss innerhalb der Entwicklungsphase eine entsprechende Überarbeitung erfolgen (*Mikro-Iteration).* Damit wird das Verfahren der Prüfung als ein Entwicklungsschritt innerhalb einer jeden Entwicklungsphase des Vorgehensmodells eingeführt (siehe Bild 4.2).

Da die Spezifikationsmethode nicht die Entwicklungsphase *Implementierung und Test* definiert, muss der Abschluss eines Inkrementes neu definiert werden. Zunächst sind zwei verschiedene Varianten des Überganges zwischen den Entwicklungsphasen *Systementwurf* und *Implementierung und Test* zu unterscheiden. Eine Variante versteht diesen Übergang als die manuelle Programmierung des Systems auf Grundlage des Spezifikationsdokumentes *Objektorientierter Steuerungsentwurf*. Automatische Generierung des Programmcodes ist möglich, wenn ein entsprechendes Werkzeug zur Verfügung steht. Unabhängig davon wird ein neues Inkrement erst dann definiert, wenn ein implementiertes Inkrement im Zielsystem oder im Zusammenspiel mit einer Simulation validiert wurde (siehe Kapitel 2.1).

4.1.2 Anwendungsfall-getriebenes Vorgehensmodell durch die PA-Methode

Produktionsagenten werden im Kontext eines Produktionssystems (PA-MAS) als Akteure verstanden, da ihre Interaktionen die funktionalen Anforderungen einer Steuerung beschreiben (siehe Kapitel 2.9.3). Sie sind somit die *internen Akteure* im Kontext der Formulierung von Anwendungsfällen. Der interne Akteur wird hier in Ergänzung zum *externen Akteur* der UML eingeführt und als technischer Teil der Steuerung verstanden, vom dem die Interaktionen des Systems bzw. der Steuerung initiiert werden. Im Unterschied zu den Akteuren der UML dienen interne Akteure nicht zur Definition der zu spezifizierenden Hauptklassen, sondern sind selbst die Hauptklassen des zu spezifizierenden Systems bzw. der Steuerung. Produktionsagenten als interne Akteure für die Beschreibung von Anwendungsfällen sind somit Bestandteil der Beschreibung des Lastenheftes (Entwicklungsphase *Anforderungsdefinition*).

4.1.3 Konzeption des *Projekt-Dekompositionsprinzips* durch die Eigenschaften *Architektur-zentriert* und *Komponentenorientiert*

Mit der Einführung des Begriffsmodells des Funktionsblocks als spezifisches Softwarekomponentenmodell mittelkörniger Granularität ist der Ansatz zur Konzeption eines Architektur-zentrierten Vorgehensmodells gegeben, der zugleich eine Komponentenorientierung aufweist (siehe Kapitel 3.1.4). Die Definition der Komponentenorientierung fordert aber darüber hinaus, die Möglichkeit der Wiederverwendung von Softwarekomponenten-Implementierungen (siehe Kapitel 2.10). Hieraus folgt, dass eine unmittelbare Dekomposition des Begriffsmodells des Funktionsblocks in seine *White-Box*-Beschreibung (Verhaltensbeschreibung, Schnittstellen usw.) nicht vorgesehen werden kann. Eine weitere Stufe der Software-Dekomposition ist in Form eines *feinkörnigen Softwarekomponentenmodells* mit einer ergänzenden Beschreibung der Komponenten-Implementierung einzuführen. Diese Beschreibung muss

so ausgeführt werden, dass sie die in der Automatisierungstechnik verwendeten Komponententechniken spezifizieren kann (siehe Bild 4.2).

Zur Vervollständigung der Konzeption eines Architektur-zentrierten und komponentenorientierten Vorgehensmodells ist das Begriffsmodell des Funktionsblocks mit dem bereits eingeführten internen Akteur *Produktionsagent* zu verbinden. Gemäß der Analyse der Modellvorstellungen und Begriffsmodelle ist das Begriffsmodell des Funktionsblocks als Stufe der System-Dekomposition unterhalb des Steuerungsprozessmodells anzusehen (siehe Kapitel 3.2). Gleichzeitig aber ist der interne Akteur PA in der Prozesssicht als PA-Kopf und PA-Rumpf modelliert, wobei der Agentenkopf selbst wiederum als ein Steuerungsprozess anzusehen ist. Der Rumpf eines PAs wird hierbei als aus mindestens einem weiteren Steuerungsprozess bestehend modelliert. Weiterhin ist aus Sicht des Projekt-Dekompositionsprinzips der PA-Rumpf transparent, d.h. die funktionalen Anforderungen werden durch den PA-Kopf umgesetzt (siehe Kapitel 3.1.2). Berücksichtigt man also die Prozesssicht des Produktionsagenten, so ist das Begriffsmodell des Funktionsblocks bezüglich des System-Dekompositionsprinzips die nächst feinere Dekompositionsstufe des internen Akteurs *Produktionsagent* (siehe Bild 4.2).

Der in Bild 4.2 beschriebene Aspekt der Projekt-Dekomposition bezüglich des feinkörnigen Softwarekomponentenmodells ist aus Tabelle 4.1 abgeleitet. Das Begriffsmodell des Funktionsblocks inklusive seiner Dekomposition wird als Beschreibungsmittel in Lasten- und Pflichtenheft verwendet und ist somit als Schritt zur Software- und Projekt-Dekomposition anzusehen.

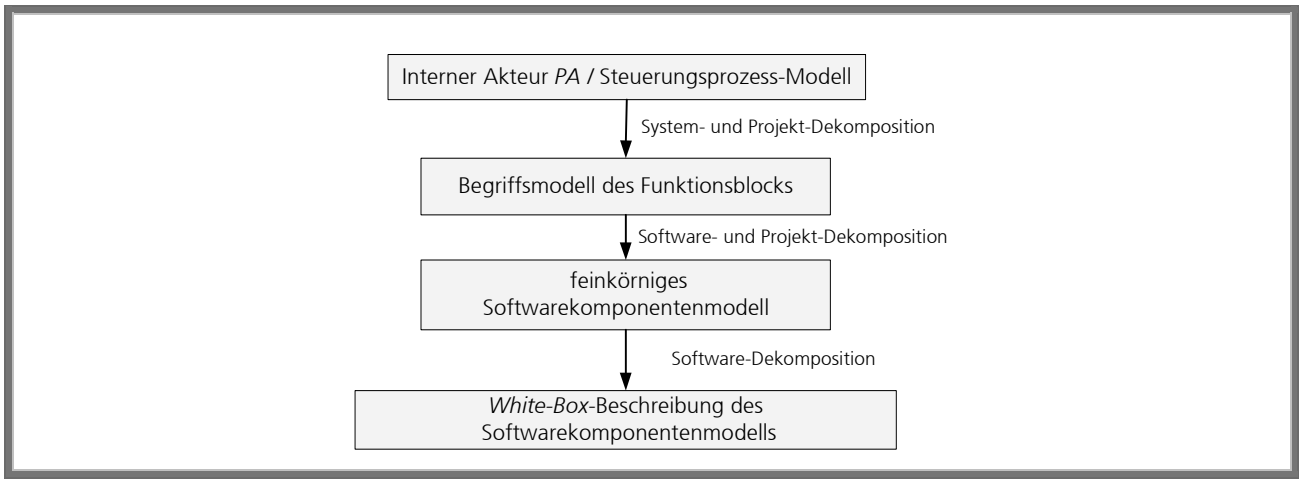

Bild 4.2: Stufen der System- und Software-Dekomposition im Kontext der Konzeption des *Projekt-Dekompositionsprinzips*

4.2 Konzeption der Anwendbarkeit des Vorgehensmodells

4.2.1 Integration des Dokumentenstandards *VDI/VDE-Richtlinie 3694* und Abbildung der Modellvorstellungen und Begriffsmodelle

Die Integration des Dokumentenstandards VDI/VDE-Richtlinie 3694 bedeutet für das Lasten- und Pflichtenheft zunächst eine inhaltliche Strukturierung der bezüglich des Begriffes der Spezifikationsmethode als generisch angenommenen Spezifikationsdokumente (siehe Kapitel 2.2 und 2.11). Berücksichtigt man weiterhin das noch zu entwickelnde Beschreibungsmittel der ODEMA-Methode, so ist ei-

ne Zuweisung der fundamentalen Beschreibungstechniken in den Kontext der Spezifikationsdokumente notwendig. Diese Beschreibungstechniken sind durch die Abbildung der bereits eingeführten Modellvorstellungen und Begriffsmodelle definiert und basieren konzeptionsgemäß auf der UML (siehe Kapitel 3.2). Tabelle 4.1 weist diesen Modellen entsprechende UML-Beschreibungstechniken und ordnet diese den dazugehörigen Spezifikationsdokumenten zu. Hierbei ist zu beachten, dass lediglich für Lasten- und Pflichtenheft eine detaillierte Strukturierung im Sinn von Abschnitten definiert ist. Das Spezifikationsdokument *Objektorientierter Steuerungsentwurf* ist ein aus Sicht des Entwicklers internes Dokument, das unternehmensspezifisch strukturiert ist. Darüber hinaus gilt für jedes Spezifikationsdokument, dass nur ein Teil dieser Dokumente durch das Beschreibungsmittel der ODEMA-Methode spezifiziert wird. Der übrige Teil wird weiterhin informal, d.h. i.d.R. textuell beschrieben (z.B. die nicht funktionalen Anforderungen des Lastenheftes). Hierbei sind das Ebenenmodell und das Echtzeitmodell als integrative Modelle zu verstehen, die nicht einer einzelnen UML-Beschreibungstechnik zugeordnet werden können. Sie sind durchgängig in das gesamte Beschreibungsmittel der ODEMA-Methode zu integrieren (siehe Kapitel 3.2).

Begriffsmodell oder Modellvorstellung	UML-Beschreibungstechnik	Spezifikationsdokument/Abschnitt
Steuerungsprozess-Modell	Steuerungsprozess (aktive Klasse)	Lastenheft / 3. Aufgabenstellung (3.1 bis 3.5)
Produktionsagent	Interner Akteur und Steuerungsprozess	Lastenheft/ 2. Beschreibung der Ausgangssituation (2.2) 3. Aufgabenstellung (3.1 bis 3.5)
Begriffsmodell des Funktionsblocks	UML-Funktionsblock	Lastenheft/ 4. Schnittstellen (4.1 bis 4.6) Pflichtenheft/ 9. Systemtechnische Lösung (9.1 bis 9.3)
Modellvorstellung des Automatisierungssystems	Physischer Rahmen und Agenten-Rahmen	Lastenheft/ 2. Beschreibung der Ausgangssituation (2.2) Pflichtenheft/ 10. Systemtechnik (10.1 bis 10.2)
Modulares Automatisierungsgerät	UML-Geräte-Beschreibung	Lastenheft/ 2. Beschreibung der Ausgangssituation (2.2) Pflichtenheft/ 10. Systemtechnik (10.3 bis 10.4)
Begriffsmodell des Kommunikationssystems	UML-Kommunikationssystem-Beschreibung	Lastenheft/ 2. Beschreibung der Ausgangssituation (2.2) Pflichtenheft/ 10. Systemtechnik (10.3 bis 10.4)

Tabelle 4.1: Abbildung der Modelle auf UML-Beschreibungstechniken und Abschnitte der Spezifikationsdokumente

4.2.2 Verknüpfung des Vorgehensmodell der PA-Methode mit den Spezifikationsdokumenten des Vorgehensmodells

Das Begriffsmodell des Produktionsagenten wurde im Kapitel 4.1.2 als interner Akteur und Steuerungsprozess bereits eingeführt, ohne dass dabei das Vorgehen vollständig in die Spezifikationsmethode integriert wurde. Zunächst ist die Agentifizierung (*Top-Down*- und *Bottom-Up*-Ansatz, siehe Kapitel 2.9.3) als Entwicklungsschritt innerhalb der Entwicklungsphase *Anforderungsdefinition* einzuführen. Hierbei ist die Wahl des Ansatzes nicht festgelegt, da unabhängig von der angewandten Me-

thodik als Ergebnis dieses Entwicklungsschrittes stets entsprechende Beschreibungen des Produktionsagenten als interner Akteur bzw. Steuerungsprozess vorliegen. Der Entwicklungsschritt *Modellierung PA* hat in diesem Zusammenhang die Aufgabe, die durch die Agentifizierung gefundenen Produktionsagenten (Spezifikation: *Rolle bzw. Instanz/Rollen:Typ*) in den Kontext entsprechender UML-Diagrammtypen zu überführen (Beschreibungsmittel).

Darüber hinaus erfordert die Integration der Richtlinie VDI/VDE 3694, dass die Agentifizierung eines Produktionssystems unterschiedliche Ausgangsbedingungen bezüglich des Entwicklungsprojektes berücksichtigen muss. Abschnitt 2 und 3 gehen von einem Soll- und einem Ist-Zustand eines Produktionssystems aus, d.h. die Entwicklung bzw. Spezifikation einer Steuerung muss berücksichtigen, dass die neu zu entwickelnden Steuerungsfunktionen in eine bestehende Steuerung integriert werden müssen. Dies führt zu unterschiedlichen Varianten bezüglich der Agentifizierung (siehe Tabelle 4.2).

Da unabhängig von der Anwendung der verschiedenen Ansätze zur Agentifizierung nach ihrem Abschluß die gleichen Spezifikationen vorliegen, hat ausschließlich der triviale Fall (Fall 1b) Auswirkung auf die Konzeption der Spezifikationsdokumente bzw. des Beschreibungsmittels (siehe Bild 4.3).

Nr.	Ausgangssituation	Konzeption des Vorgehensmodells
1.	Die Entwicklung einer Steuerung für ein Produktionssystem ist eine Neu-Entwicklung, d.h. es existiert kein zu erweiterndes Produktionssystem.	a) Zur Agentifizierung wird der *Top-Down-Ansatz* angewandt.
		b) Es ist gefordert, eine Maschine (*Produktionssubsystem*) zu entwickeln. Folglich wird die Agentifizierung nicht angewandt (*Trivialer Fall*).
2.	Die Entwicklung einer Steuerung für ein Produktionssystem muss ein bestehendes Produktionssystem und seine Steuerung berücksichtigen, d.h. es existiert ein zu erweiterndes Produktionssystem.	c) Es ist gefordert, dass die bestehende Steuerung auf ein PA-MAS umgestellt wird. Hieraus folgt, dass zunächst das bestehende Produktionssystem mittels des *Bottom-up-Ansatzes* agentifiziert wird. Danach werden mittels des *Top-Down-Ansatzes* die zu erweiternden PA spezifiziert.
		d) Es ist gefordert, dass die bestehende Steuerung erhalten bleibt. Hieraus folgt, dass die bestehende Steuerung bzw. das bestehende Produktionssystem als ein einziger PA angesehen wird und keine weitere Agentifizierung erfolgt (*Stellvertreter-PA*).

Tabelle 4.2: Konzeption des Vorgehensmodells aufgrund unterschiedlicher Ausgangssituationen des Entwicklungsprojektes

4.3 Integration der Prüfbarkeit

Das Verfahren *Prüfung* ist als Entwicklungsschritt bereits durch die Konzeption des iterativ-inkrementellen Vorgehensmodells eingeführt worden (siehe Kapitel 4.1.1). Definitionsgemäß ist im Kontext der Anwendung der UML als Beschreibungsmittel die Prüfung semantischer und syntaktischer Konsistenz zu unterscheiden (siehe Kapitel 2.1). Darüber hinaus ist zur Validation ein Verfahren einzuführen, das mit einem UML-basierten Beschreibungsmittel angewendet werden kann.

Die Prüfung der syntaktischen Konsistenz bedeutet im Kontext der hier zu entwickelnden Methode, dass UML-Erweiterungsmechanismen zur Entwicklung des Beschreibungsmittels, bedingt durch den informalen Anteil der UML-Semantik (d.h. z.B. der fehlende strukturelle Zusammenhang im UML-Metamodell zwischen einer *Operation* und einem *Zustand*) und die Notwendigkeit der Integration der Modellvorstellungen und Begriffsmodelle, genutzt werden müssen (siehe Kapitel 4.2.1). Die daraus resultierenden syntaktischen Lücken müssen somit auf Beschreibungsebene überbrückt werden und durch ein entsprechendes Verfahren geprüft werden.

Ursächlich für diese Lücken (*Syntaktische Inkonsistenz*) ist, dass mittels der Erweiterungsmechanismen zwar erweiterte UML-Beschreibungselemente aus dem Metamodell abgeleitet werden können, ihre strukturellen Zusammenhänge untereinander aber können innerhalb der Beschreibung bzw. spezifikation nicht beschrieben werden und somit von einem CASE-Tool nicht konsistent gehalten werden [UML_03]. Die UML sieht hierfür ausschließlich eine externe tabellarische Beschreibung vor (UML-Profil).

Um das Verfahren *Prüfung auf syntaktische Konsistenz* mit Unterstützung von CASE-Tools anwenden zu können, sind entsprechend erweiterte UML-Beschreibungselemente und ihre Zusammenhänge in separaten UML-Diagrammtypen zu spezifizieren. Dies erhöht zudem die Übersichtlichkeit der anwendungsbezogenen Beschreibungen (*Anwendbarkeit*). Zu diesem Zweck wird die UML-Diagrammtyp-Rolle *Konsistenzsichernder Diagrammtyp* eingeführt, der als ein beliebiger UML-Diagrammtyp zu verstehen ist, der ausschließlich UML-Beschreibungselemente beinhaltet, die die syntaktische Konsistenz der Spezifikation beschreiben. Diese Diagrammtyprolle ist ergänzend zu den Beschreibungen eines UML-Profils zu sehen und somit kommen beide Beschreibungstechniken im Rahmen dieser Arbeit zum Einsatz.

Zur Prüfung der semantischen Konsistenz wird das Verfahren *Prüfung auf semantische Widerspruchsfreiheit* eingeführt. Dies Verfahren wird ebenfalls durch die Nutzung der Erweiterungsmechanismen notwendig, denn z.B. die Integration des formalen Echtzeitmodells verlangt, dass die korrekte Erfüllung dieser algebraischen Beschreibungen Diagramm-übergreifend geprüft werden kann (siehe Kapitel 4.2.1 und 3.1.7).

Abschließend ist bezüglich der Konzeption des Vorgehensmodells der ODEMA-Methode das Validationsverfahren *Test* einzuführen. Dieses Verfahren setzt im Kontext der UML voraus, dass semantisch ein Zusammenhang zwischen dem Sequenz- und dem Zustands- bzw. Aktivitätsdiagramm besteht. Sequenzdiagramme beschreiben in diesem Kontext die funktionalen Anforderungen als Szenario, die durch die Spezifikation eines Zustandsdiagramms erfüllt werden müssen (siehe auch Kapitel 2.9.1 und 2.9.2). Dies ist als eine Anforderung an das Beschreibungsmittel der UML zu verstehen. Die Konzeption des Verfahrens *Test* ist darüber hinaus nur in der Entwicklungsphase *Systementwurf* anzuwenden, da nur das Spezifikationsdokument *Objektorientierter Steuerungsentwurf* die notwendigen UML-Diagrammtypen enthält (*White-Box*-Beschreibung der Steuerungskomponenten). Das Verfahren *Test* muß aber nicht im Rahmen dieser Arbeit entwicklet werden, da es durch andere UML-basierte Methoden (z.B. Doglass-Methode) bereits bekannt ist.

Prinzipiell sind alle hier eingeführten Verfahren so zu beschreiben, dass sie durch ein entsprechendes Werkzeug automatisch oder durch einen Entwickler mittels manuellen Prüfens durchgeführt werden können (entsprechende Werkzeugkonzepte siehe Kapitel 8). Zusammenfassend beschreibt Tabelle 4.3 die Verfahren der Validation und Prüfung.

Verfahren	Beschreibung	Prüfung	Validation
Prüfung auf syntaktische Konsistenz	1. Durch Nutzung von UML-Erweiterungsmechanismen (z.B. Beschreibung der System-Dekomposition) entstehen syntaktische Lücken 2. Um eine syntaktische Durchgängigkeit auch aus Sicht eines CASE-Tools zu erreichen, werden *Konsistenz-sichernde Diagrammtypen* eingeführt. 3. Bezüglich der Nutzung der Erweiterungsmechanismen sind Prüfkriterien für die syntaktische Konsistenz zu formulieren. Grundlage hierbei ist die Formulierung eines *präzisen* Beschreibungsmittels auf Basis der UML.	x	
Prüfung auf semantische Widerspruchsfreiheit	1. Erfolgreiche Prüfung auf syntaktische Konsistenz ist Voraussetzung. 2. Die UML ist ein semiformales Beschreibungsmittel, d.h. das Metamodell der UML hat informale Anteile, die anwendungspezifische präzisiert werden müssen. 3. Durch die Integration Diagramm-übergreifender Begriffsmodelle und Modellvorstellungen (Nutzung der Erweiterungsmechanismen) entsteht eine durch die UML nicht definierte Semantik. 4. Bezüglich der Nutzung der Erweiterungsmechanismen und des teilweise informalen Metamodells der UML sind Prüfkriterien für die semantische Konsistenz zu formulieren. Grundlage hierbei ist ebenfalls die Formulierung eines *präzisen* Beschreibungsmittels auf Basis der UML.	x	
Test	1. Sequenzdiagramme beschreiben die funktionalen Anforderungen als nachrichtenbasierte Szenarien. 2. Die Whitebox-Beschreibung von Steuerungskomponenten (Zustands- und Aktivitätsdiagramme) muss Anforderungen aus allen relevanten Szenarien erfüllen. 3. Prinzipiell (ohne Aktivitätsdiagramme) ist dieses Verfahren bereits in der Douglass-Methode beschrieben.		x

Tabelle 4.3: Vefahren für die Validation und Prüfung

4.4 Abschließende Integration des Vorgehensmodells

Das Ergebnis der Konzeption des Vorgehensmodells wird in Bild 4.3 zusammenfassend dargestellt. Ergänzend zu der bis hierher dargestellten Konzeption ist noch das Produkt *Modell-Bibliothek* einzuführen und die parallel durchgeführte Entwicklung bezüglich mechanischer und elektrotechnischer Konstruktion zu integrieren (*Basic*- und *Detailed-Engineering,* siehe [METZ_97]).

Die bisherige Konzeption des Vorgehensmodells berücksichtigt die Komponenten-Orientierung ausschließlich hinsichtlich des Beschreibungsmittels bzw. der damit zu spezifizierenden Produkte (siehe Kapitel 4.1.2). Darüber hinaus ist hiermit aber die Wiederverwendung von implementierter und gegebenenfalls kommerziell erworbener Software verbunden. Diese Art der Software ist zwar dem Entwickler bezüglich der *White-Box*-Beschreibung unbekannt, realisiert aber Steuerungsfunktionen und muss somit in der Spezifikation berücksichtigt werden. Aus diesem Grund wird das nur dem Entwickler bekannte Produkt *Modell-Bibliothek* eingeführt, das in strukturierter Form ausschließlich UML-Beschreibungselemente enthält, die zur Spezifikation von Steuerungen wieder verwendet werden. Da aber auch ein bestehendes Produktionssystem oder auch der Einsatz von Automatisierungsgeräten (Gerätekomponenten) implizit die Wiederverwendung von implementierter Software bzw. Steuerungsfunktionalität bedeutet, ist erst im Zusammenhang mit der Entwicklung des Beschreibungsmittels festzulegen, welche UML-Beschreibungselemente zur Modell-Bibliothek gezählt werden können und wel-

che lediglich projektbezogen bleiben. Das Erstellen dieser Modell-Bibliothek ist ein projektübergreifender Prozess, der nicht Teil der Spezifikation einer Steuerung ist (siehe weiterführend [EISEN_00]).
Zur Vervollständigung der Konzeption des Vorgehensmodells gehört, dass die Entwicklungsschritte der mechanischen und elektrotechnischen Konstruktion, z.B. die Planung der Verdrahtung von Automatisierungsgeräten und die damit verbundene Erstellung entsprechender Pläne, berücksichtigt werden. Diese Entwicklungsschritte verlaufen zeitlich parallel zur Spezifikation bzw. Entwicklung der Steuerung, wobei auf die identischen Spezifikationsdokumente zugegriffen wird. Hierbei werden die Entwicklungsschritte der anderen Gewerke als *Basic*- und als *Detailed-Engineering* zusammengefasst (siehe Bild 4.3).

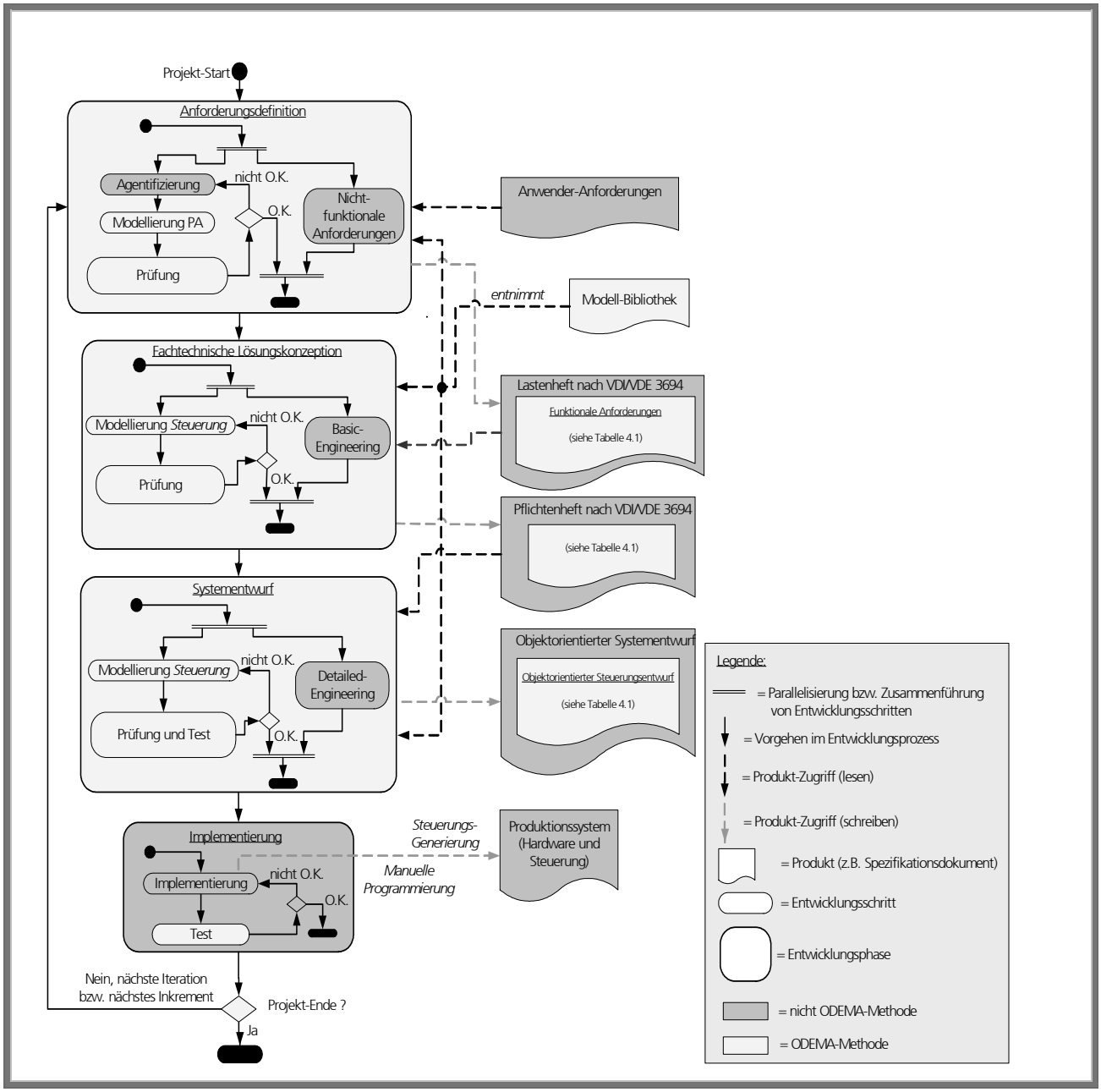

Bild 4.3: Vorgehensmodell der ODEMA-Methode

5 Konzeption des UML-basierten Beschreibungsmittels der Methode zur Spezifikation von Steuerungen

In diesem Kapitel wird die Konzeption des Beschreibungsmittels der ODEMA-Methode auf Basis der UML mit Unterstützung der UML-Erweiterungsmechanismen beschrieben (siehe Kapitel 2.12). Unter Berücksichtigung des konzeptionierten Vorgehensmodells der ODEMA-Methode (UML-Beschreibungstechniken, siehe Kapitel 4.2.1) wird diese Konzeption mit Hilfe der in der UML bereits enthaltenen Diagrammtypen durchgeführt. Zur Erfüllung des *Struktur*- und *Temporalprinzips* sind darüber hinaus das Ebenenmodell, das Begriffsmodell des Produktionsagenten und das Echtzeitmodell zu integrieren (siehe Kapitel 3.2). Obwohl zur Konzeption des Beschreibungsmittels prinzipiell ausschließlich die Systemaxiome (als Teil der Methoden-Axiome verstanden) berücksichtigt werden, ist das Methoden-Axiom *Anwendbarkeit* grundsätzlich ebenso in diesem Teil der Konzeption zu berücksichtigen. Die Anforderung nach einem möglichst niedrigen Umfang zu erlernender Syntax und nach durchgängiger Verwendung gleicher Beschreibungstechniken macht dies erforderlich (siehe Tabelle 2.1). Die zu berücksichtigenden Einflussfaktoren bezüglich der Konzeption sind in Bild 5.1 dargestellt.

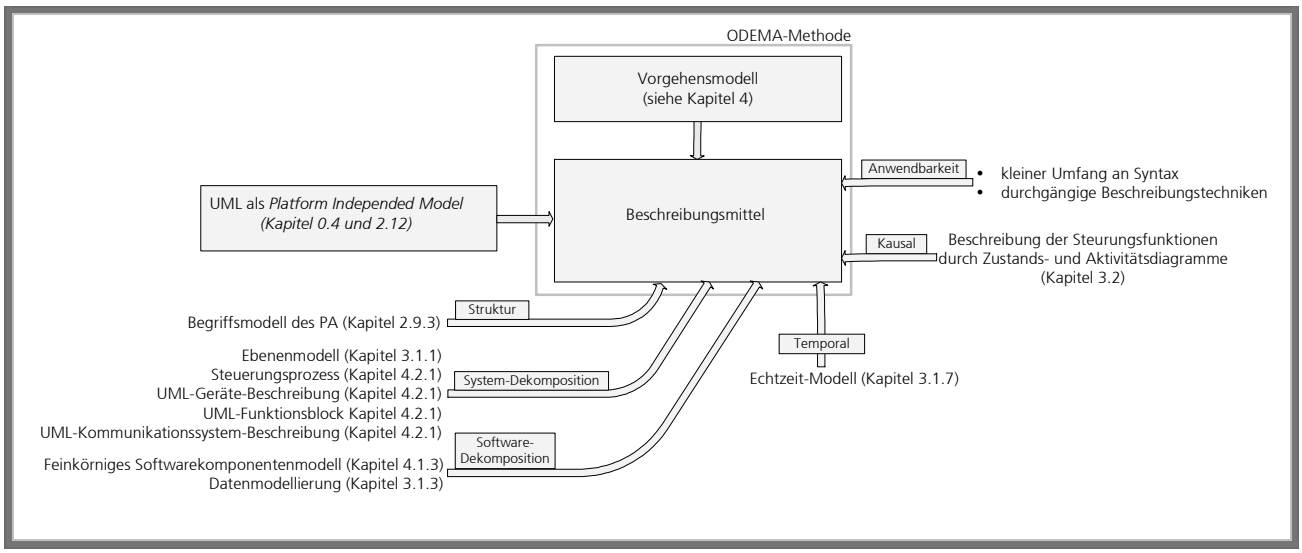

Bild 5.1: Einflussfaktoren zur Konzeption des Beschreibungsmittels der ODEMA-Methode

Für die UML bedeutet die Konzeption bzw. Entwicklung eines neuen Beschreibungsmittels die Anwendung der UML-Erweiterungsmechanismen *Stereotypen*, *Eigenschaftswerte* und *Einschränkungen*. Zusammenfassend werden die erweiterten UML-Beschreibungselemente als *UML-Profil* bezeichnet. Ein sUML-Profil ist ein UML-Paket (Stereotyp *profile*), welches die Elemente des Metamodells der UML beschreibt, die durch diese Mechanismen erweitert wurden (*ODEMA-Profil*, siehe Bild 5.2). Dies kann durch ein spezielles Klassendiagramm oder auch tabellarisch beschrieben werden [UML_03]. Die tabellarische Darstellung wird in dieser Arbeit verwendet.

Bezüglich der Definition von Stereotypen werden englische Bezeichnungen verwendet, da die UML ein internationaler Standard ist. In den Fällen, wo die Namen neu eingeführter Stereotypen oder Eigenschaftswerte ähnlich oder identisch zu Stereotypen der UML oder anderer Profile sind, wird der Präfix `OD` für den Namen verwendet.

Darüber hinaus werden in dieser Arbeit für die UML-Beschreibungselemente die deutsche Namen verwendet, die auch im Kapitel 0.4 eingeführt wurden. Ausnahmen hierbei sind die auch außerhalb der UML verwendeten Begriffe (z.B. Komponente und Knoten), die das Präfix `UML` erhalten.

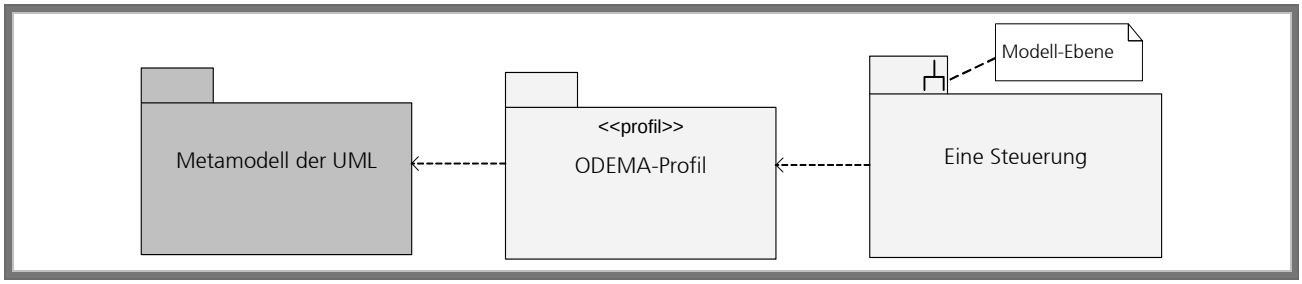

Bild 5.2: Darstellung des ODEMA-Profils im Kontext der UML

5.1 Herleitung der Systemgrenzen der Steuerung aus dem Strukturprinzip

Der erste Schritt zur Konzeption des Beschreibungsmittels ist die Zusammenführung des *Begriffsmodelsl des Produktionsagenten* (Strukturprinzip) mit dem Beschreibungsmittel UML. Das *Begriffsmodell des Produktionsagenten* ist das einzige Modell des Strukturprinzips und umfasst zudem eine physische und eine logische Sicht. Aus diesem Grund ist die Integration dieses Begriffsmodells der erste Schritt zur Konzeption des Beschreibungsmittels, d.h. zur Bestimmung der Systemgrenzen (siehe Bild 5.1).

Der Produktionsagent als interner Akteur, d.h. als Teil der Steuerung, wurde bereits eingeführt (siehe Kapitel 4.1.2). Durch die Berücksichtigung eines bestehenden Produktionssystems im Zusammenhang mit der Entwicklung bzw. Spezifikation einer Steuerung wurde ebenfalls der Stellvertreter-PA eingeführt (siehe Kapitel 4.2.2). Ergänzt man einen menschlichen Bediener mit entsprechender Eingabemöglichkeit (*Bedienersteuerung*) als einen externen Akteur in der Definition der UML, so lassen sich mit Hilfe der eingeführten Akteure die Systemgrenzen der Steuerung bestimmen (siehe Bild 5.3).

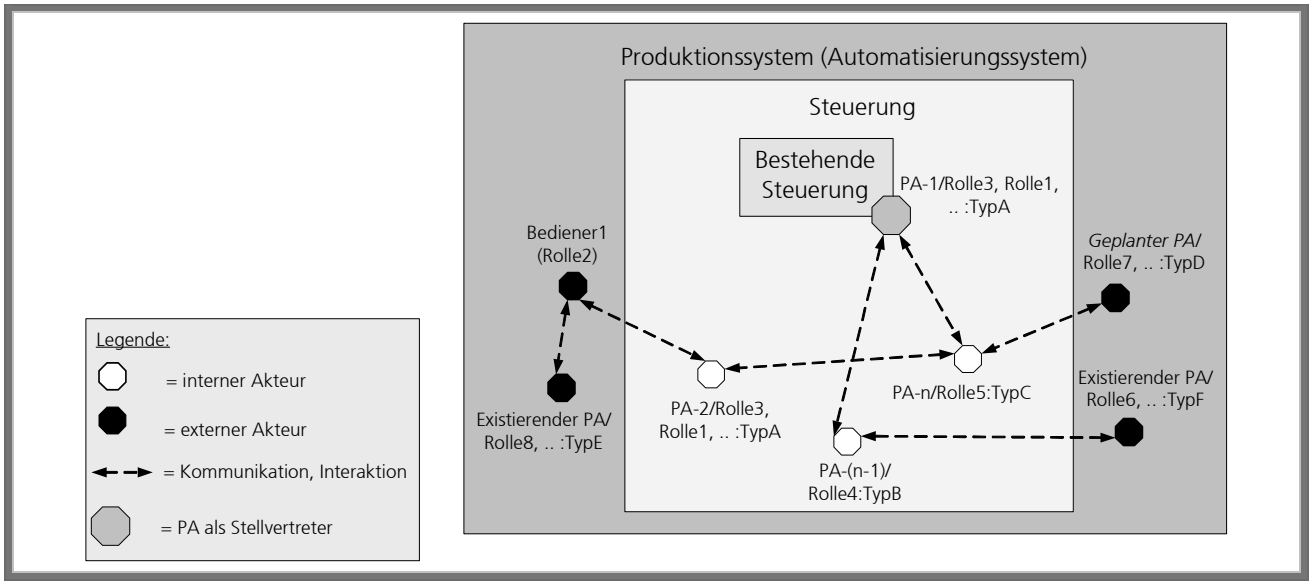

Bild 5.3: Systemgrenzen der Steuerung eines Produktionssystems im Kontext der ODEMA-Methode

Hierbei sind der menschliche Bediener und bereits implementierte oder geplante Produktionsagenten als externer Akteur anzusehen, die zwar Teil des Produktionssystems, aber nicht Teil der zu spezifizierenden Steuerung sind. Die Interaktion dieser externen Akteure mit der Steuerung muss somit bezüg-

lich der Beschreibung der Anwendungsfälle berücksichtigt werden. Das System-Dekompositionsprinzip wird auf die externen Akteure aber nicht angewendet. Dies ist ausschließlich relevant für die internen Akteure. Der Stellvertreter-PA nimmt hierbei eine Sonderstellung ein, da für diesen lediglich der PA-Kopf spezifiziert werden muss, denn der PA-Rumpf (*existierende Steuerung*) besteht bereits und soll nicht verändert werden. Trotzdem gehört dieser Teil der Steuerung zum betrachteten System, da vom praktischen Standpunkt aus gesehen stets von Eingriffen in die PA-Rumpfsteuerung zur Integration des stellvertretenden PA-Kopfes auszugehen ist.

5.2 Herleitung des Grob-Konzeptes des Beschreibungsmittels auf Basis des Ebenenmodells

In Kapitel 3.2 wird bezüglich des Ebenenmodells festgestellt, dass es die oberste Stufe der *Systemdekomposition* modelliert und zudem die logische und physische Sicht beinhaltet. Somit ist das Ebenenmodell die Basis der Integration der Modelle zur Beschreibung des *System-Dekompositionsprinzips*. Da die Darstellung der UML nach dem HRUBY-Ansatz allgemein die Dekomposition eines Systems ausgehend von der System-Schicht in Stufen bis hin zur Methoden-Schicht definiert, ist diese Darstellung, als Modell verstanden, mit dem Ebenenmodell zu verbinden (siehe Kapitel 0.4).

Zunächst kann das Ebenenmodell vereinfacht werden, wenn in Betracht gezogen wird, dass ausschließlich Steuerungsprozesse bzw. Steuerungsfunktionen zu spezifizieren sind. Die Definitionen des Ebenen- sowie des Steuerungsprozessmodells zeigen, dass diese nur in der Prozesssteuerungs- und Prozessführungsebene zu finden sind. Da aber die sensorischen und aktorischen Funktionen die Schnittstellen zum zu steuernden technischen Prozess bilden, ist auch die Prozessebene zu berücksichtigen. Somit müssen ausschließlich diese drei Ebenen in die Sichten-Schichten-Darstellung integriert werden.

Die Integration von Ebenenmodell und Sichten-Schichten-Darstellung erfolgt zum einen auf der Basis der Sichten-übergreifenden Definition des Ebenenmodells, welche eine Orthogonalität von Sichten und Ebenen zur Folge hat. Zum anderen beschreibt Kapitel 4.1.3 das *Dekompositionsprinzip* als Abfolge von System- und Software-Dekomposition, wobei das Ebenenmodell als abstrakteste Stufe der System-Dekomposition anzusehen ist (siehe Kapitel 3.2). Hieraus folgt eine dreidimensionale Ebenen-Sichten-Schichten-Darstellung, die jeder der drei Ebenen eine Sichten-Schichten-Darstellung zuordnet. Für die Systemschicht in allen Ebenen heißt dies, dass Steuerungsfunktionen bzw. sensorische und aktorische Funktionen auf Akteure (*Anwendungsfallsicht*) bzw. Anwendungsprozesse (*Prozesssicht*) abgebildet werden (*ODEMA-Würfel*, siehe Bild 5.4). Die so gewonnene Darstellung umfasst das Grob-Konzept des Beschreibungsmittels und alle UML-Diagrammtyp-Rollen aus Sicht des Struktur- und System-Dekompositions- und Software-Dekompositionsprinzips.

Die Projekt-Dekomposition ist definitionsgemäß nicht orthogonal zur System- und Software-Dekomposition und orientiert sich somit ebenfalls an der Dimension der Schichten in der Sichten-Schichten-Darstellung. Lediglich die Abstufungen der Dekomposition sind unterschiedlich, da die Pro-

jekt-Dekomposition diese Dekompositionsstufen durch die Spezifikationsdokumente definiert (siehe Kapitel 4.2.1).

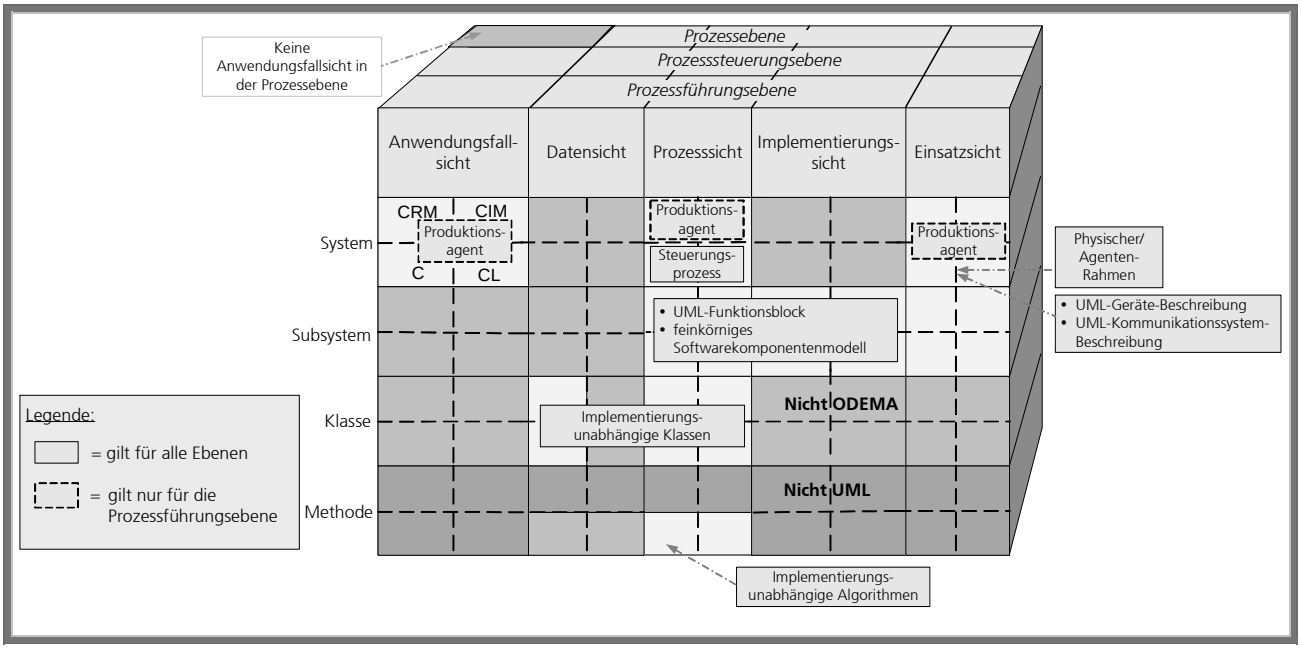

Bild 5.4: Vereinfachte Ebenen-Sichten-Schichten-Darstellung (*ODEMA-Würfel*, CRM = Classifier Relationship Model, CIM = Classifier Interaction Model, C = Classifier, CL = Classifier Lifecycle)

Zur Vervollständigung der Grob-Konzeption sind noch zwei Ergänzungen vorzunehmen. Zum einen ist für das hier zu entwickelnde Beschreibungsmittel die Analysesicht einer Steuerung nicht notwendig. Die Analysesicht ist Teil der logischen Sicht und beschreibt die Spezifikation eines *Software-Prototypen*. Das bedeutet, dass die funktionalen Anforderungen ohne Berücksichtigung der physischen Sicht als Analysemodell beschrieben und in Richtung einer ablauffähigen Software – dem Software-Prototypen - spezifiziert und implementiert werden. Da die physische Sicht nicht beschrieben wird, fehlen wesentliche Informationen bezüglich des *System-Dekompositions-* und *Temporalprinzips* (siehe Kapitel 3.2).

Zum anderen ist die *Datensicht* in die Grob-Konzeption einzuführen. Die Analyse der Datenmodellierung hat gezeigt, dass Datenobjekte wesentliche Elemente zur Spezifikation der Kommunikationsverbindungen zwischen Steuerungsprozessen beschreiben (siehe Kapitel 3.1.3). Ein weiterer Aspekt ist die Berücksichtigung von Datenfluss-orientierten Beschreibungen zur Spezifikation von Software, die im Kontext von Geschäftsprozessen zum Einsatz kommen. Steuerungen werden zwar Prozess-orientiert beschrieben, aber die zukünftige Integration von Funktionen aus den Ebenen oberhalb der Prozessführungsebene wird auf diese Weise ermöglicht.

5.3 Herleitung der vereinfachten Ebenen-Sichten-Schichten-Darstellung

Das Beschreibungsmittel der ODEMA-Methode wurde aufgrund der Heterogenität der Automatisierungsgeräte bezüglich ihrer Programmierung als ein *Platform Independet Model* (PIM) konzeptioniert (siehe Kapitel 2.12). Hieraus ist unmittelbar abzuleiten, dass die Klassen- und die Methodenschicht der Ebenen-Sichten-Schichten-Darstellung als Beschreibung von Architekturklassen und dazugehörigen

Algorithmen verstanden werden muss, die keine programmiersprachlichen Elemente verwenden (*Implementierungs-unabhängige Klassen* und *Algorithmen*, siehe Bild 5.4).

Darüber hinaus sind aber weitere Vereinfachungen der Ebenen-Sichten-Schichten-Darstellung möglich, die auf eine Reduzierung der Anzahl der UML-Diagrammtyp-Rollen zielen. Dies bewirkt die Reduzierung der Komplexität des Beschreibungsmittels, wie das Methoden-Axiom *Anwendbarkeit* es fordert.

Zunächst kann die Anwendungsfallsicht vereinfacht werden, die ausschließlich zur Darstellung der funktionalen Anforderungen dient (siehe Tabelle 0.1). Das Vorgehen der PA-Methode, welches Teil des Vorgehensmodells der ODEMA-Methode ist, transformiert diese Anforderungen in den Kontext der Beschreibung eines PA-MAS. Das Begriffsmodell des Produktionsagenten definiert hierbei den PA-Kopf als Steuerungsprozess in der Prozessführungsebene, der diese Anforderungen in der Prozesssicht beschreibt (siehe Kapitel 2.9.3). Hierbei ist der PA-Rumpf, d.h. die Steuerungsprozesse der Prozesssteuerungsebene, transparent. Ist lediglich eine einzelne Maschinen-Steuerung zu spezifizieren, kann der Bediener der Maschine direkt mit den Steuerungsprozessen der Prozesssteuerungsebene interagieren, wobei hierbei die Prozessebene transparent ist. Dies verringert für diesen die Anzahl der UML-Diagrammtypen. Darüber hinaus kann festgestellt werden, dass die Anwendungsfallsicht ausschließlich bezüglich der Beschreibung der Prozessführungs- und Prozesssteuerungsebene berücksichtigt werden muss.

Die Sichten-Schichten-Darstellung der UML bietet die Möglichkeit, Anwendungsfälle in der *System-Subsystem-* und *Klassen-Schicht* zu beschreiben. Die Konzeption des *System-* und *Software-Dekompositionsprinzip* bezüglich des Vorgehensmodells sieht aber die Dekomposition des internen Akteurs in der Prozesssicht vor. Das heißt, es entstehen durch die Dekomposition keine weiteren internen Akteure und die Beschreibung dekomponierter Anwendungsfälle ist nicht möglich bzw. nicht notwendig. Die Schichten *Subsystem* und *Klassen* können bezogen auf die Anwendungsfallsicht also aus der Konzeption genommen werden.

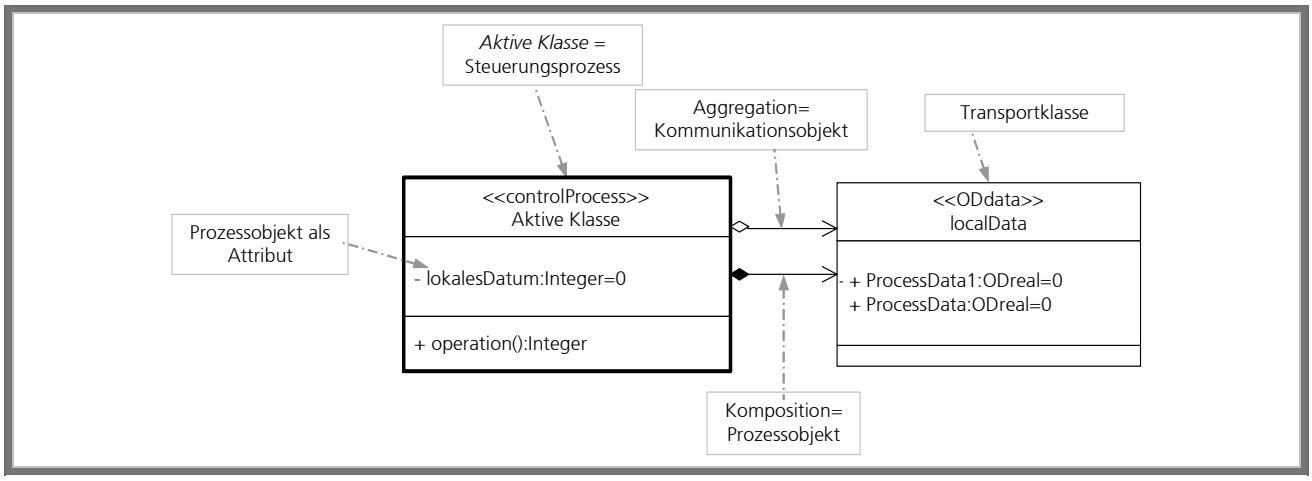

Bild 5.5: Konzept der Datenmodellierung des Beschreibungsmittels

Die Analyse der Datenmodellierung hat gezeigt, dass Kommunikationsobjekte sich aus objektorientierter Sicht geeignet als Transportobjekte bzw. –klassen darstellen lassen (siehe Kapitel 3.1.3). Lokale

Prozessobjekte eines Steuerungsprozesses können in diesem Kontext als Transportobjekte mit einer stärkeren Kopplung an die den Steuerungsprozess repräsentierende *aktive Klasse* verstanden werden. Hierzu eignen sich bei einfachen Datentypen Attribute von Klassen (*Class*). Im Fall von komplexen Datenstrukturen sind Transportklassen am besten geeignet, die mittels einer *Komposition* mit dem Steuerungsprozess (Stereotyp *controlProcess*) verbunden sind (siehe Bild 5.5). Eine *Aggregation* beschreibt hingegen die gemeinsame Nutzung der Transportklassen von Steuerungsprozessen (*Kommunikationsobjekte*). Transportklassen können aufgrund ihrer Einfachheit bezüglich der Anforderungsdefinition ebenso verwendet werden wie im Kontext der Beschreibung implementierungs-unabhängiger Klassen. Es kann somit auf eine Dekomposition verzichtet werden und Transportklassen werden bezüglich der vereinfachten Ebenen-Sichten-Schichten-Darstellung der Klassenschicht zugeordnet (siehe Bild 5.4, *Implementierungs-unabhängige Klassen*). Die Datensicht berücksichtigt ausschließlich *Classifier* und das *Classifier Relationship Model*.

Implementierungs- und Einsatzsicht bilden zusammen die *physische Sicht* der UML. Die Einsatzsicht lässt sich in der Klassenschicht insofern beschreiben, als dass UML-Komponenten auch Quelle-Code, also Implementierungen von Klassen in einer bestimmten Programmiersprache, darstellen können und deren örtliche Verteilung im Zusammenhang mit einem Einsatzdiagramm beschrieben werden kann. Da das Beschreibungsmittel der ODEMA-Methode den Anforderungen eines PIM entsprechen soll, muss diese Eigenschaft auch in der physischen Sicht erhalten bleiben. Die Klassen-Schicht ist für diese Sicht somit nicht zu berücksichtigen.

Durch die Konzeption der Ebenen-Sichten-Schichten-Darstellung auf Basis der PA-Methode ist die Aufgabe der System-Schicht nicht mehr ausschließlich die Darstellung der funktionalen Anforderungen, sondern es wird ebenso das *Strukturprinzip* einer Steuerung spezifiziert. Das Begriffsmodell des Produktionsagenten berücksichtigt hierbei auch die physische Sicht einer Spezifikation in Form der Zusammenfassung von Automatisierungsgeräten und Produktionssubsystemen (*UML-Geräte-Beschreibung*, *Agenten-Rahmen*, siehe Kapitel 4.2.1). Dieser Teil der Spezifikation ist der Einsatzsicht zuzuordnen (siehe Kapitel 0.4). Die Implementierungssicht hat explizit die Aufgabe, UML-Komponenten unabhängig von ihrer Verteilung darzustellen. Somit kann sie für die System-Schicht vernachlässigt werden.

Aus den bisher beschriebenen Einschränkungen und Festlegungen kann auch das bereits eingeführte Beschreibungsmittel des *UML-Funktionsblocks* in die Ebenen-Sichten-Schichten-Darstellung integriert werden. Das Begriffsmodell des Funktionsblocks wird als mittelkörnige Software-Komponente verstanden (siehe Kapitel 3.1.4) und Softwarekomponentenmodelle werden in der UML mittels *UML-Subsystemen* dargestellt [UML_03]. Somit lässt sich das Beschreibungsmittel *UML-Funktionsblock* ebenso wie das *feinkörnige Softwarekomponentenmodell* aus Kapitel 4.1.3 der Subsystem-Schicht zuordnen.

Mit dem bis hierher beschriebenen Verfahren ist die Konzeption der vereinfachten Ebenen-Sichten-Schichten-Darstellung abgeschlossen (siehe Bild 5.4).

5.4 Klassifizierung der UML-Diagrammtyp-Rollen des Beschreibungsmittels

Die Konzeption des Vorgehensmodells der ODEMA-Methode hat gezeigt, dass anwendungsspezifische UML-Diagrammtyp-Rollen, also die zur Spezifikation einer Steuerung durch den Entwickler notwendigen UML-Diagrammtyp-Rollen, von dem *Konsistenz-sichernden Diagrammtyp* und den *Modell-Bibliotheken* zu unterscheiden sind (siehe Kapitel 4.3 und 4.4). Hierbei wurde die Modell-Bibliothek zunächst als Produkt des Vorgehensmodells eingeführt. Da dieses Produkt aber explizit kein Entwicklungs- oder Spezifikationsdokument der hier zu entwickelnden Methode darstellt, soll die Modell-Bibliothek als übergreifende UML-Diagrammtyp-Rolle verstanden werden, die alle noch zu entwickelnden UML-Diagrammtyp-Rollen umfasst, welche ausschließlich wieder zu verwendende UML-Beschreibungselemente enthalten. Der *Konsistenz-sichernde Diagrammtyp* ist im Kontext dieser Arbeit ebenfalls als ein solcher Oberbegriff zu verstehen.

5.4.1 Konzeption der UML-Diagrammtyp-Rolle *Modell-Bibliothek*

Die *Classifier* (C) nehmen in der Sichten-Schichten-Darstellung der UML eine besondere Stellung ein. Sie definieren UML-Diagrammtypen, in denen alle *Classifier* (z.B. Klassen, Anwendungsfälle, und UML-Knoten) unverbunden - also ohne z.B. eine Assoziation oder Abhängigkeit - dargestellt werden. Beispielhaft lässt sich ein Klassendiagramm vorstellen, das ausschließlich Klassen beschreibt und dabei auf die Modellierung von Vererbungen, Assoziationen und Abhängigkeiten vollständig verzichtet. Diese UML-Beschreibungselemente werden dann in einem *Classifier Relationship Model* (CRM) in Form eines Klassendiagramms inklusive aller Verbindungen dargestellt. Gemäß diesem Ansatz lassen sich ein UML-Sequenzdiagramm als *Classifier Interaction Model* (CIM) und ein UML-Zustandsdiagramm als *Classifier Lifecycle* (CL) verstehen (siehe Kapitel 0.4). Voraussetzung für dieses Betrachtung ist aber, dass sich die Diagramme einer Spezifikation im gleichen *Sicht-Schicht-Schnittpunkt* befinden.

Modell-Bibliothek	Ebene/Sicht/Schicht	Modell-Bibliotheks-Paket
Daten-Bibliothek, Prozessvariablen-Bibliothek	Alle Ebenen / Datensicht / Klassenschicht	Daten-Bibliothek
Agenten-Bibliothek, Steuerungen-Bibliothek, Agenten-Protokoll-Bibliothek	Alle Ebenen / Anwendungsfall- und Prozesssicht / Systemschicht	Anforderungs-Bibliothek
PF-Funktionsblock-Bibliothek, PS-Funktionsblock-Bibliothek, PF-Komponenten-Bibliothek, PS-Komponenten-Bibliothek, PF-Implementierungen-Bibliothek, PS-Implementierungen-Bibliothek	PF- und PS-Ebene / Prozesssicht / Subsystemschicht	Prozess-Bibliothek
Maschinen-Bibliothek, PFK-PNK-Bibliothek, Feldgeräte-Bibliothek	Alle Ebenen / Einsatzsicht / Systemschicht	Maschinen- und Geräte-Bibliothek

Tabelle 5.1 : Uberblick über alle Modell-Bibliotheken des Beschreibungsmittels der ODEMA-Methode (PF=Prozessführung, PS=Prozesssteuerung, PFK=Prozessferne Komponenten, PNK=Prozessnahe Komponenten)

Erweitert man diese Konzeption geringfügig dahingehend, dass das Diagramm, das die Classifier beschreibt auch solche UML-Beschreibungselemente enthält, die in einem speziellen Projekt nicht verwendet werden, also beispielsweise im entsprechenden CRM und CIM nicht vorkommen, so lassen sich diese Diagramme als *Modell-Bibliotheken* begreifen.

Tabelle 5.1 beschreibt alle Classifier, die bezüglich des ODEMA-Beschreibungsmittels als Modell-Bibliotheken verstanden werden. Hierbei wird vereinfachend festgelegt, dass im Kontext der Anwendung der ODEMA-Methode eine Bibliothek jeweils als ein einziges Diagramm zu verstehen ist.
Anwendungsfalldiagramme werden der Kategorie *Classifier* zugeordnet. Mit Ausnahme dieses Classifiers sind alle Classifier der Sichten-Schichten-Ebenen-Darstellung als Modell-Bibliotheken zu verstehen. Anwendungsfalldiagramme beschreiben Anforderungen an eine Steuerung. Diese UML-Diagrammtyp-Rolle ist projekt-spezifisch, wird i.d.R. nicht wieder verwendet und kann daher nicht als Modell-Bibliothek verstanden werden. Interpretiert man dagegen den Classifier Lifecycle der Anwendungsfallsicht im Kontext der Prozessführungsebene als UML-Diagrammtyp-Rolle, die ein *Kommunikations-Protokoll* für Produktionsagenten beschreibt, so lässt sich diese *Agenten-Protokoll-Bibliothek* als eine wieder zu verwendende Modell-Bibliothek aufzufassen (siehe *Begriffsmodell des Produktionsagenten* Kapitel 2.9.3).
Da System- und Software-Dekomposition elementarer Bestandteil des ODEMA-Beschreibungsmittels sind, muss prinzipiell davon ausgegangen werden, dass die Verwendung eines UML-Beschreibungselements die Verwendung eines anderen aus einer abhängigen Modell-Bibliothek nach sich ziehen kann. Hieraus folgt, dass die im folgenden Kapitel herzuleitenden *Konsistenz-sichernden-Diagrammtypen* sich auch auf die Modell-Bibliotheken beziehen.
Da in jeder Sicht der Sichten-Schichten-Darstellung mehrere unterschiedliche Classifier enthalten sind, die wieder verwendet werden können, wird eine Zusammenfassung verschiedener zu einer Sicht gehörende Modell-Bibliotheken mit Hilfe des speziellen UML-Paketes *UML-Modell-Paket* vorgenommen. Ihre Ableitung ist ein Vorgriff auf die Ergebnisse der detaillierten Konzeption des Beschreibungsmittels (siehe Tabelle 5.1 und Kapitel 5.6).

5.4.2 Konzeption der UML-Diagrammtyp-Rolle *Konsistenz-sichernder Diagrammtyp*

Neben den Modell-Bibliotheken gehören die *Konsistenz-sichernden Diagrammtypen* zu den nicht-anwendungsspezifischen Diagrammtypen des zu entwickelnden Beschreibungsmittels. Sie sind a priori als notwendig anzusehen, da die Semantik der UML (*Meta-Model*) die Darstellung der UML in Sichten und Schichten nicht durchgängig unterstützt (*Semiformale und nicht präzise Semantik*). Das Methoden-Axiom *Prüfbarkeit* fordert aber die *Prüfung auf syntaktische Konsistenz* als Voraussetzung für die *Prüfung auf semantische Widerspruchsfreiheit* einer erstellten Spezifikation (siehe Kapitel 4.3). Während festgestellt werden kann, dass das Sichten-Prinzip Bestandteil des Meta-Modells der UML ist, muss die Dekomposition der UML-Beschreibungselemente in Schichten auf der Modellebene ausgedrückt werden.
Das Beispiel der UML-Beschreibungselemente *Anwendungsfall*, *Klasse*, *Komponentenimplementierung* und *Knoten* zeigt, dass die Semantik der UML im Zusammenhang mit der Darstellung vom *Classifier Relationship Model* ausreichend beschrieben ist, um alle Sichten in einem einzigen UML-Diagrammtyp darzustellen - in diesem Fall dem Klassendiagramm. Dies gilt allerdings nur für die Beschreibung der

statischen Beziehungen. Wird ein Anwendungsfall durch ein Aktivitätsdiagramm (CL) ergänzt oder eine entsprechende Klasse durch ein Zustandsdiagramm (CL), so ist durch die Semantik der UML lediglich die Abhängigkeit dieser UML-Diagrammtypen definiert. Details wie z.B. der Zusammenhang zwischen der Operation einer Klasse und dem Zustand des Objektes dieser Klasse sind nur informal oder gar nicht definiert. Diese fehlende Semantik muss von einem Entwickler mittels der UML-Erweiterungsmechanismen anwendungsspezifisch festgelegt werden.

Im Gegensatz zu den Sichten kann die Dekomposition als *UML-Verfeinerung* (Abhängigkeit mit dem Stereotyp *refine*) zwischen UML-Paketen bzw. Klassen lediglich auf Modellebene beschrieben werden. In diesem Zusammenhang kann auch die hierarchische Beschreibung von Zuständen und Aktivitäten durch die UML als Dekomposition (präzise: als Verfeinerung) eines Zustandes verstanden werden (siehe Bild 5.6). Die zusammenfassende Organisation von UML-Diagrammen durch UML-Pakete, UML-Subsysteme und UML-Modell-Pakete (*Model*) ergänzen die Beschreibung der Dekomposition. Eine Unterscheidung in System-, Software- und Projekt-Dekomposition wird durch die UML aber nicht unterstützt.

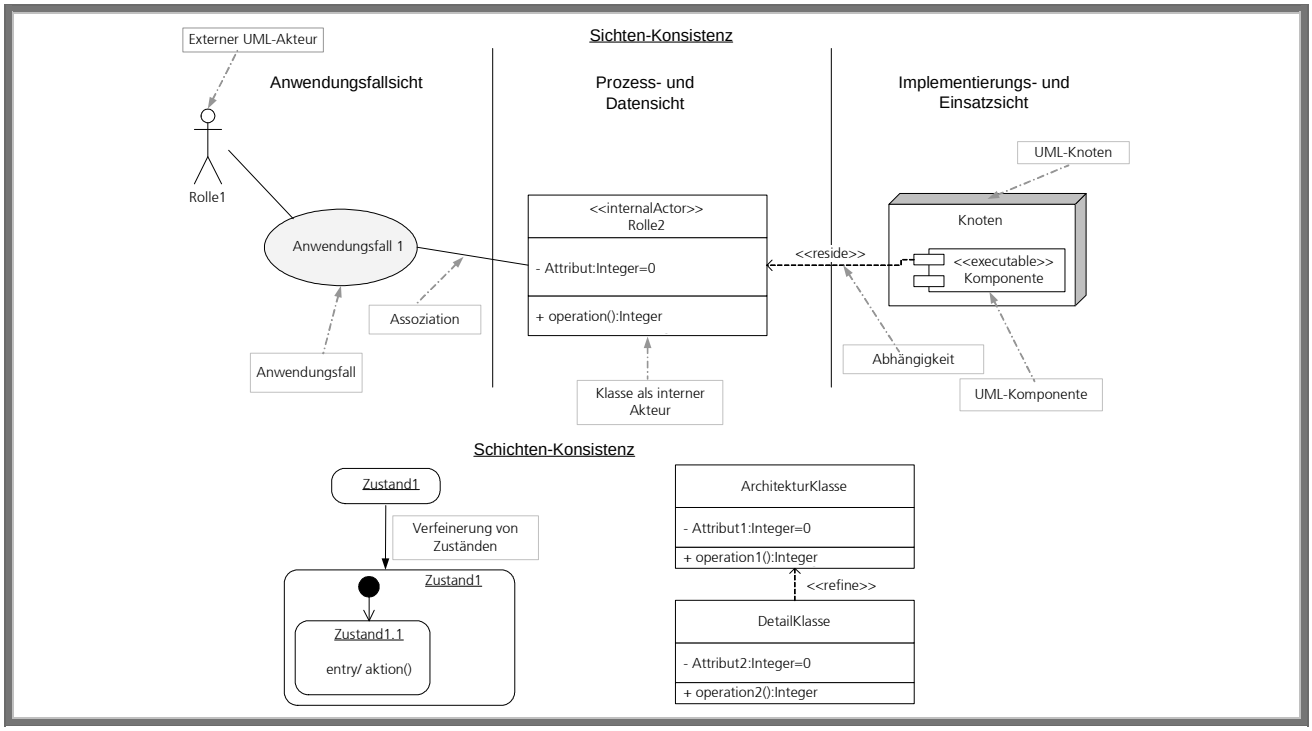

Bild 5.6: Beschreibungstechniken zur Konsistenzsicherung in der UML

Hieraus folgt, dass das ODEMA-Beschreibungsmittel überall dort *Konsistenz-sichernde Diagrammtypen* bereitstellen muss, wo durch die UML selbst keine ausreichende Semantik geliefert wird bzw. anwendungsspezifische UML-Beschreibungselemente eingeführt wurden. Dies gilt auch für die UML-Diagrammtyp-Rollen, die durch die Erweiterung der Sichten-Schichten-Darstellung mittels der Ebenen eingeführt wurden (siehe Bild 5.4). Da hierbei die UML-Erweiterungsmechanismen genutzt werden, kann durch die Erweiterung und Präzisierung der Semantik der UML auf Modellebene keine Formalisierung des ODEMA-Beschreibungsmittels erreicht werden. Die semiformale Eigenschaft der UML bleibt erhalten.

Eine detaillierte Herleitung der spezifischen *Konsistenz-sichernden Diagrammtypen* kann sinnvoll erst nach Abschluss der Detail-Konzeption des ODEMA-Beschreibungsmittels durchgeführt werden (siehe Kapitel 5.5.5).

5.5 Detaillierte Konzeption des Beschreibungsmittels

5.5.1 Konzeption der System- und Software-Dekomposition

Nach der Herleitung der Grob-Konzeption in Form der vereinfachten Ebenen-Sichten-Schichten-Darstellung sollen weitere Reduzierungen der Anzahl von UML-Diagrammtyp-Rollen durchgeführt werden, die sich aus dem Kontext der Beschreibung von Steuerungen ergeben, d.h. ebenenbezogene Vereinfachungen sind. Ergänzend hierzu werden die aus den Begriffsmodellen und Modellvorstellungen hervorgehenden UML-Beschreibungstechniken entsprechenden UML-Diagrammtyp-Rollen zugewiesen.

Basierend auf der Definition der Dekompositionsstufen aus Kapitel 4.1.3 und der Ebenen-Sichten-Schichten-Darstellung kann das detaillierte Dekompositionsschema abgeleitet werden (siehe Bild 5.7).

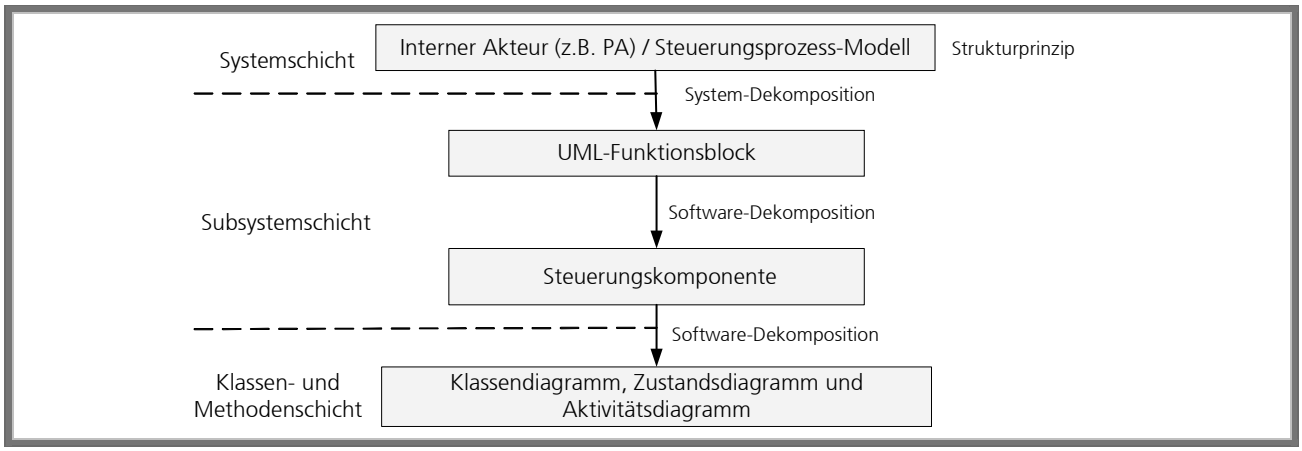

Bild 5.7: Konzeption der System- und Software-Dekomposition bezüglich des ODEMA-Beschreibungsmittels

Zum UML-Funktionsblocks gehört, dass sein gekapseltes Verhalten mittels einer zustandsbasierten Ausführungskontrolle und zusätzlich durch die Beschreibung von Algorithmen und Daten spezifiziert wird (siehe Kapitel 3.1.4). Die Dekomposition des UML-Funktionsblocks in ein feinkörniges Softwarekomponentenmodell macht die Einführung eines anwendungsspezifischen Softwarekomponentenmodells notwendig, das explizit als Realisierung eines UML-Funktionsblocks definiert wird (*Software-Dekompositionsprinzip*). Dieses wird als *Steuerungskomponente* (UML-Subsystem mit Stereotyp *controlComp*, siehe Bild 5.8) eingeführt. Steuerungskomponenten sind durch die Konzeption des ODEMA-Beschreibungsmittels als PIM ebenfalls implementierungs-unabhängig. Sie verfügen aber im Gegensatz zum UML-Funktionsblock über eine Beschreibung ihrer Komponenten-Implementierung (*UML-Komponente, Component*).

UML-Funktionsblöcke verfügen gemäß ihrem Begriffsmodell über Nachrichten-orientierte Schnittstellen-Klassen, die vom UML-Funktionsblock implementiert (*Export*) oder die von einem anderen UML-Funktionsblock angefordert werden (*Import*).

Die Software-Dekomposition des UML-Funktionsblocks geschieht mittels zweier verschiedener Varianten von Steuerungskomponenten. Die *aktive Softwarekomponente* enthält die zustandsbasierte Ausführungskontrolle und die passive die Spezifikation der Algorithmen. Beide Varianten enthalten Spezifikationen von Daten [BRAATZ_03a].

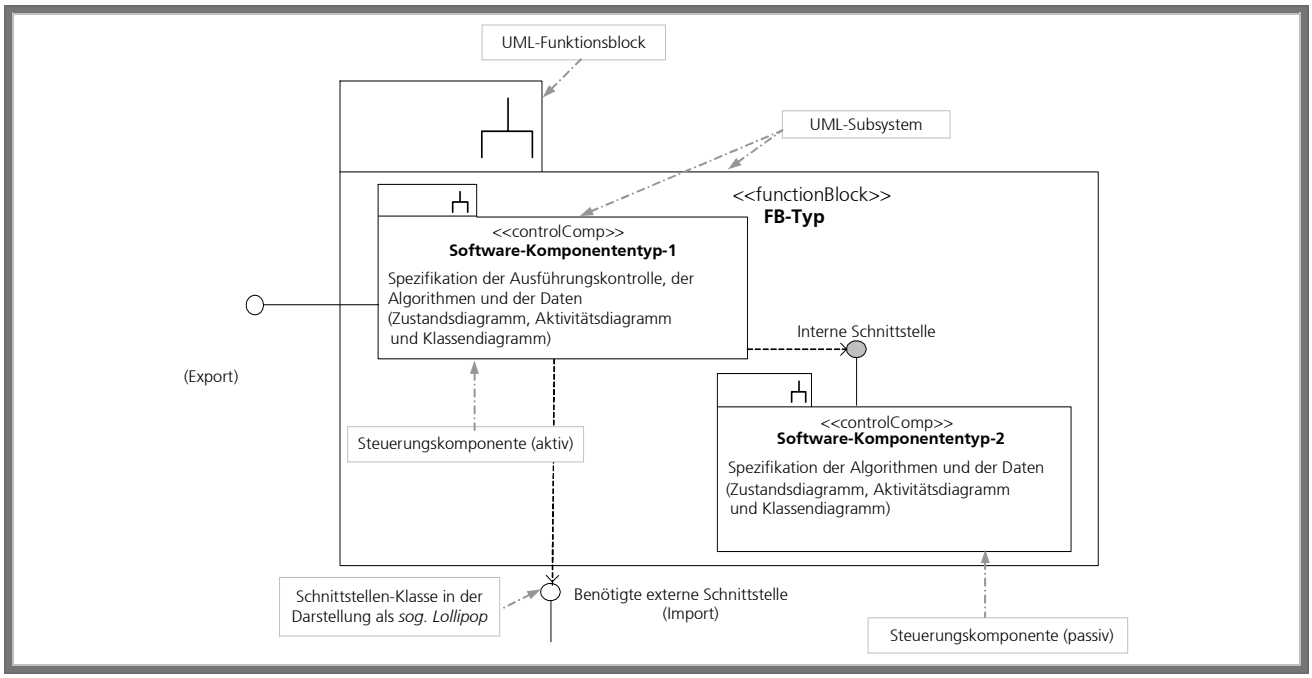

Bild 5.8: Konzeption der Software-Dekomposition des UML-Funktionsblocks (FB = Funktionsblock)

Diese Konzeption des UML-Funktionsblocks berücksichtigt besonders die Forderung nach Komponentenorientierung (siehe Kapitel 4.1.3). Eine aktive Steuerungskomponente kann durchaus auch über zusätzliche Algorithmen-Spezifikationen verfügen, wenn diese aufgrund ihrer Einfachheit den Entwurf einer zusätzlichen passiven Steuerungskomponente nicht geeignet erscheinen lassen. Somit besteht ein UML-Funktionsblock aus mindestens einer aktiven Steuerungskomponente (weitere Details der Konzeption finden sich in [BRAATZ_03a]).

Die abschließende Stufe der Software-Dekomposition bildet die Spezifikation der Steuerungskomponente. Die UML sieht hierfür die Möglichkeit zur Beschreibung des Spezifikations- und Realisierungsabschnitts eines UML-Subsystems vor (siehe [UML_03]). Bedingt durch die Konzeption als PIM kann keine Realisierung z.B. in Form eines Technologie-abhängigen Klassendiagramms erfolgen. Alle zur Beschreibung einer Steuerungskomponente notwendigen UML-Diagrammtypen gehören somit zum Spezifikationsabschnitt des UML-Subsystems (siehe Bild 5.8).

Die aus der Ebenen-Sichten-Schichten-Darstellung abgeleitete Konzeption zur Dekomposition bezüglich des ODEMA-Beschreibungsmittels gilt prinzipiell für alle Ebenen.

5.5.2 Detail-Konzeption der Prozessführungsebene

Durch die in Kapitel 5.5.1 hergeleitete Konzeption der System- und Software-Dekomposition kann bezüglich der Subsystemschicht in der Prozesssicht der Prozessführungsebene auf eine Spezifikation des Verhaltens (Classifier Lifecycle) verzichtet werden, da das Verhalten der UML-Funktionsblöcke und der

Steuerungskomponenten in der Klassenschicht der Prozesssicht durch die Spezifikation der Steuerungskomponenten beschrieben wird.

Die statische Spezifikation der Steuerungskomponente (CRM) wird gemäß der UML durch ein Klassendiagramm modelliert, das bezüglich des ODEMA-Beschreibungsmittels die UML-Diagrammtyp-Rolle *Komponenten-Spezifikation* erhält. Die Spezifikation der Ausführungskontrolle wird durch das *Komponenten-Verhaltensdiagramm* (Zustandsdiagramm) und die der Algorithmen einer Steuerungskomponente durch die Diagrammtyp-Rolle *Algorithmen-Spezifikation* (Aktivitätsdiagramm) beschrieben. Beide UML-Diagrammtypen werden gemäß dem Konzeptionsansatz prinzipiell zur Beschreibung des *Kausalprinzips* einer Steuerung herangezogen. Ergänzend zur Komponenten-Spezifikation ist das *Datendiagramm* (Klassendiagramm) zu sehen (siehe Bild 5.8). Obwohl das Konzept der Transportobjekte bzw. –klassen erhalten bleibt, können z.B. sog. *Agententelegramme* so komplex sein, dass sie sinnvoll nur durch eine Kombination mehrerer Transportklassen beschrieben werden können (siehe Kapitel 3.1.3). Dies ist dann in einem separaten Klassendiagramm - also dem Datendiagramm – zu beschreiben, um die UML-Diagrammtyp-Rolle *Komponenten-Spezifikation* nicht durch zu viele Beschreibungselemente unübersichtlich zu machen (*Anwendbarkeit*).

Da bezüglich der Beschreibung von Produktionsagenten prinzipiell keine *White-Box*-Beschreibung vorgesehen ist, ist auch kein *Classifier Lifecycle* zu berücksichtigen (Prozesssicht/Systemschicht). Elementar zur Beschreibung funktionaler Anforderungen ist aber die Beschreibung der Interaktion zwischen Produktionsagenten. Hierfür ist das *System-Interaktionsdiagramm* vorgesehen, welches die anwendungsspezifische UML-Diagrammtyp-Rolle eines Sequenzdiagramms darstellt. Dieser UML-Diagrammtyp wird gewählt, weil weiche Echtzeitspezifikationen im Zusammenhang mit der Spezifikation von Produktionsagenten möglich sind und das Sequenzdiagramm durch die Berücksichtigung der Zeitachse hierfür am besten geeignet ist (*Temporalprinzip*).

Grundsätzlich beschreiben Sequenzdiagramme Szenarien, also beispielhafte Verläufe der nachrichtenbasierten Interaktion zwischen Instanzen (*Objekten*). Im Kontext der Beschreibung von Anwendungsfällen werden Sequenzdiagramme zur verfeinerten Darstellung der funktionalen Anforderungen verwendet. Neben der Systemstruktur spezifizieren sie im Kontext der Beschreibung von Produktionsagenten ebenso Kausal- und Temporalprinzip der Steuerung, insoweit sie ein bestimmtes kausales und temporales Verhalten fordern.

Im Zuge der Beschreibung von UML-Funktionsblöcken und Steuerungskomponenten steht die Beschreibung der System– und Software-Dekomposition von Produktionsagenten und schließlich deren Realisierung im Vordergrund. Eine Dekomposition von System-Interaktionsdiagrammen ist hierbei nicht notwendig, da durch die Möglichkeit der Beschreibung von *Steuerungen-Interaktionsdiagrammen* in der Prozesssteuerungsebene dies bereits vorgesehen ist (Integration des Ebenenmodells, siehe Kapitel 5.2). Zudem ist die Interaktion innerhalb eines Funktionsblocks der Prozessführungsebene so wenig komplex, dass eine Dekomposition in weitere Sequenzdiagramme unnö-

tig ist und die Beschreibung des *Classifier Interaction Model* in der Subsystem- und Klassenschicht der Prozesssicht nicht ausgeführt wird.

Die UML-Diagrammtyp-Rollen der Implementierungssicht beschreiben die Komponenten-Implementierungen von Steuerungskomponenten (*PF-Implementierungsdiagramm*). Die Einsatzsicht beschreibt deren Verteilung auf die Architektur der Automatisierungsgeräte (*Installationsdiagramm*). Hieraus folgt unmittelbar, dass die Verhaltensbeschreibungen der UML-Komponenten identisch sind mit denen der Steuerungskomponenten in der Prozesssicht. UML-Diagrammtyp-Rollen, die aus der Kategorie *Classifier Lifecycle* hervorgehen, werden somit bezüglich der gesamten physischen Sicht nicht berücksichtigt.

Sicht / Schicht	Anwendungsfall		Daten	Prozess		Implementierung	Einsatz	
System	**CRM** 1.Systemarchitektur 2.*Systemszenarien*	**CIM**		1.*Agenten-Dekompositionsdiagramm* 2. Agenten-Diagramm	System-Interaktionsdiagramm		Installationsdiagramm	
	C System-Anwendungsfall	**CL** Agenten Protokoll-Bibliothek		Agenten-Bibliothek			1.Maschinen-Bibliothek	
Subsystem	**Nicht ODEMA**			1. PF-Funktionsblockarchitektur 2. PF-Komponentenarchitektur		PF-Implementierungsdiagramm		
				1. PF-Funktionsblock-Bibliothek 2. PF-Komponenten-Bibliothek		PF-Implementierungen-Bibliothek		
Klasse			Daten-Diagramm	Komponenten-Spezifikation				
			Daten-Bibliothek		Komponenten-Verhaltensdiagramm			
Methode	**Nicht UML**				Algorithmen-Spezifikation			

Tabelle 5.2: UML-Diagrammtyp-Rollen der Prozessführungsebene als Ergebnis der Detail-Konzeption (Unterstrichen = Modellbibliothek, kursiv = Konsistenz-sichernder Diagrammtyp, x = Streichung durch Detail-Konzeption, C = Classifier, CIM = Classifier Interaction Model, CL = Classifier Lifecycle, CRM = Classifier Relationship Model, PF = Prozessführung)

Wie bereits im Kontext der Prozesssicht diskutiert, kann auch in der Implementierungssicht auf die Beschreibung eines Classifier Interaction Model verzichtet werden, zumal in der Implementierungssicht a priori keine Anforderungen modelliert werden und somit keine Sequenzdiagramme eingesetzt werden. Das Verhalten der Automatisierungsgeräte (*UML-Geräte-Beschreibung*) ist durch die Komponen-

tenimplementierungen der Steuerungskomponenten gegeben und nicht durch das UML-Beschreibungselement *UML-Knoten* selbst, das lediglich die örtliche Beschreibung der UML-Komponenten spezifiziert (siehe auch Kapitel 0). Somit wird die Kategorie *Classifier Interaction Model* für die gesamte physische Sicht vernachlässigt.

Bedingt durch die Integration des Ebenenmodells, das das System-Dekompositionsprinzip auch aus der physischen Sicht beschreibt, gilt für die Einsatzsicht außerdem, dass diese Sicht direkt auf die Ebenen abgebildet werden kann. Die Darstellung der Subsystemschicht ist daher nicht notwendig, da die gesamte Geräte-Architektur im Ebenenmodell beschrieben ist. Das *Installationsdiagramm*, das diese Gerätearchitektur beschreibt, ist somit Ebenen-übergreifend angelegt.

Die Eigenschaften von Automatisierungsgeräten, die das *Kausal-* und *Temporalprinzip* von Steuerungen beeinflussen, sind als *offered QoS* der UML-Geräte-Beschreibung zu verstehen und werden detailliert im Kontext der Konzeption des Installationsdiagramms beschrieben werden. Dies ist nicht spezifisch für eine Ebene und wird somit gesondert dargestellt (siehe Kapitel 5.5.6).

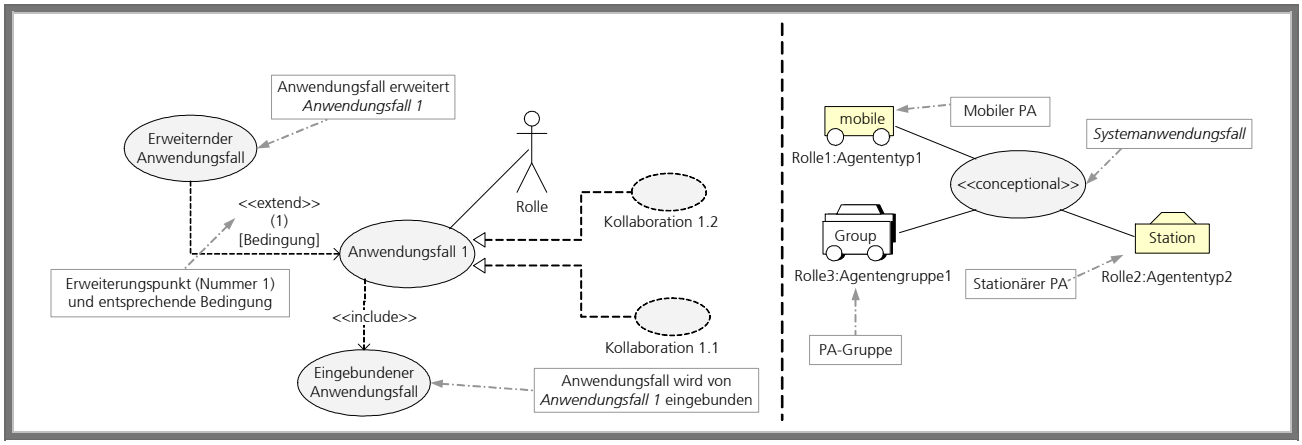

Bild 5.9: Beschreibung von Anwendungsfällen und Systemanwendungsfällen

Bedingt durch den Einsatz der Produktionsagenten zur Spezifikation der funktionalen Anforderungen gibt es Besonderheiten bezüglich der Beschreibung von Anwendungsfällen im Kontext des ODEMA-Beschreibungsmittels zu beachten. Der Produktionsagent kann zur Beschreibung des *Strukturprinzips* und gleichzeitig zur Anforderungsdefinition im Kontext der Spezifikation von Anwendungsfällen verwendet werden. Diese spezifischen Anwendungsfälle werden als *System-Anwendungsfälle* eingeführt (Anwendungsfall mit dem Stereotyp *conceptional*, siehe Tabelle 5.2 und Bild 5.9). In der Regel sind diese Anwendungsfälle - bedingt durch das Konzept des internen Akteurs - nicht unabhängig voneinander. Ein System-Anwendungsfall ist als logische Fortsetzung eines vorhergehenden anzusehen. Dies kann mit den Mitteln der UML als *Erweiterungsabhängigkeit* (Abhängigkeit mit dem Stereotyp *<<extend>>*) beschrieben werden. Ein Anwendungsfall kann durch einen oder mehrere Anwendungsfälle erweitert werden, wobei jeder einem bestimmten Erweiterungspunkt (*Extension Point*) zugeordnet ist und dieser erweiternde Anwendungsfall nur unter einer spezifizierten Bedingung (*Boolescher Ausdruck*) eintritt (siehe Bild 5.9). Bezüglich des ODEMA-Beschreibungsmittels wird diese UML-Beschreibungstechnik übernommen. Die Beschreibung der voneinander abhängigen System-Anwendungsfälle wird durch UML-Diagrammtyp-Rolle *Systemarchitektur* (Anwendungsfalldiagramm)

realisiert. Zur Darstellung der System-Anwendungsfälle müssen auch die Produktionsagenten als interne Akteure modelliert werden. Externe Akteure werden mit der UML ausschließlich durch ihrer Rolle spezifiziert, da von einem menschlichen Bediener ausgegangen wird (siehe Bild 5.9 und Kapitel 0.4). Das Begriffsmodell des Produktionsagenten sieht zusätzlich die Spezifikation eines Typs bzw. einer Klasse vor. Somit wird der Produktionsagent als interner Akteur durch die Spezifikation der *Rolle* und des *Agententyp* vom Entwickler spezifiziert. Um die Anwendbarkeit aus Sicht des Anwendungsbereichs *Produktionssysteme* zu erhöhen, werden die Stereotypen *Mobiler PA* und *Stationärer PA* eingeführt und ihnen entsprechende grafische Symbole zugeordnet (siehe Tabelle 5.3a,b und Bild 5.9).

Stereotyp	**Basis-Klasse**	**Eltern-Klasse**	**Eigenschafts-werte**	**Einschränkungen (*formal*/informal)**
UML-Funktionsblock *<<functionblock>>*	Subsystem	N/A	-	• *IsInstantiable=true* • enthält genau eine aktive Steuerungs-komponente • enthält eine beliebige Anzahl passiver Steuerungskomponenten
Daten-Klasse *<<ODdata>>*	Class	N/A	-	• *isActive=false* • enthält ausschließlich Attribute
Steuerungskomponente *<<controlComp>>*	Subsystem	N/A	ODactive	• *IsInstantiable=true* • ist nicht weiter hierarchisierbar • wird in der Prozess- und Implementierungssicht beschrieben
System-Anwendungsfall *<<conceptional>>*	Use Case	N/A	-	-
Interner Akteur *<<InternalActor>>*	Class	N/A	-	• *Self.isActive=true*
Mobiler PA *<<mobile>>*	Class Object	• Interner Akteur • Steuerungsprozess	-	• *Self.isActive=true*
Stationärer PA *<<station>>*	Class Object	• Interner Akteur • Steuerungsprozess	-	• *Self.isActive=true*
Agenten-Gruppe *<<groupOfAgents>>*	Class Object	Interner Akteur	-	• *Self.isActive=true*
Semantische Notiz *<<semantic>>*	Comment	N/A	-	• Enthält auszuwertende Spezifikationen (Eigenschaftswerte und Einschränkungen)

Tabelle 5.3a Abschnitt des UML-Profils für die UML-Diagrammtyp-Rollen der Prozessführungsebene (N/A= keine Eltern-Klasse definiert, PA = Produktionsagent)

Eigenschaftswert	**Stereotyp**	**Typ**	**Multiplizität**
ODactive	controlComp	UML::Datatypes::Boolean	1

Tabelle 5.3b: Eigenschaftswerte des UML-Profils aus Tabelle 5.3a

Wird bezüglich eines System-Anwendungsfalls ein bereits spezifiziertes *Agenten-Protokoll* (z.B. Auftragsverhandlung) angewendet, so ist dieses als Verhaltensbeschreibung (Classifier Lifecycle) des System-Anwendungsfalls zu verstehen, da ein Agenten-Protokoll basierend auf den Rollen der beteiligten Produktionsagenten beschrieben ist und somit nicht als Verhaltensbeschreibung eines einzelnen Produktionsagenten bzw. seines Typs verstanden werden kann (siehe Kapitel 2.9.3). Eine Interaktion zwischen System-Anwendungsfällen, wie sie HRUBY allgemein für Anwendungsfälle vorschlägt, ist nicht notwendig, da die Interaktion der Produktionsagenten, wie sie durch das System-

Interaktionsdiagramm beschrieben wird, die Beschreibung der Szenarien realisieren. Die Kategorie *Classifier Relationship Model* kann im Kontext der Anwendungsfallsicht vernachlässigt werden.

Der Übergang von der Anwendungsfall- in die Prozesssicht wird mit Hilfe des *Agenten-Diagramms* beschrieben. Hierbei wird der interne Akteur *Produktionsagent* (*Rolle:Type*) definitionsgemäß als Steuerungsprozess (Klasse mit Stereotyp *controlProcess*) aufgefasst werden und auf diese Weise der Prozesssicht zugewiesen (siehe Kapitel 4.1.3). Da die UML statische Abhängigkeiten zwischen Akteuren zulässt, z.B. Vererbung der Rollen zwischen Akteuren, wird dies bezüglich der Produktionsagenten im Agenten-Diagramm dargestellt.

Für die Klassenschicht ist im Zusammenhang mit der Prozesssicht keine Kategorie *Classifier* in Form von Modell-Bibliotheken beschrieben. Die Steuerungskomponenten stellen die feinste Stufe der wieder zu verwendenden UML-Beschreibungselemente dar (siehe Kapitel 5.3). Einzelne Klassen werden im Kontext der Steuerungskomponenten spezifiziert, aber isoliert von diesen nicht wieder verwendet. Die im Zusammenhang der Prozessführungsebene eingeführten UML-Beschreibungselemente bzw. Stereotypen des ODEMA-Beschreibungsmittels werden in Form des tabellarischen UML-Profils in Tabelle 5.3a beschrieben. Die Zusammenfassung der detaillierten Konzeption der UML-Diagrammtyp-Rollen in der Prozessführungsebene ist in Tabelle 5.2 dargestellt.

Grundsätzlich ist darauf hinzuweisen, dass die ODEMA-Methode keinen Unterschied zwischen UML-Funktionsblöcken und Steuerungskomponenten der Prozessführungsebene (*PF-Funktionsblock* bzw. *PF-Komponente*) und denen der Prozesssteuerungsebene (*PS-Funktionsblock* bzw. *PS-Komponente*) macht. Beide werden durch die gleichen UML-Beschreibungstechniken beschrieben. Neben der besseren Ebenen-Zuordnung ist diese unterschiedliche Bezeichnung trotzdem gewählt worden, um vorausschauend eine noch zu entwickelnde, spezialisierte Verhaltensbeschreibung von Produktionsagenten auch nachträglich in das ODEMA-Beschreibungsmittel zu integrieren. Diese Entwicklung ist aber nicht Gegenstand dieser Arbeit (siehe Kapitel 8).

5.5.3 Detail-Konzeption der Prozesssteuerungsebene

Ausgangspunkt zur Konzeption der UML-Diagrammtyp-Rollen der Prozesssteuerungsebene ist der bezüglich des ODEMA-Vorgehensmodells berücksichtigte Sonderfall, dass die zu spezifizierende Steuerung eines Produktionssystems eine einzelne Maschinensteuerung ist.

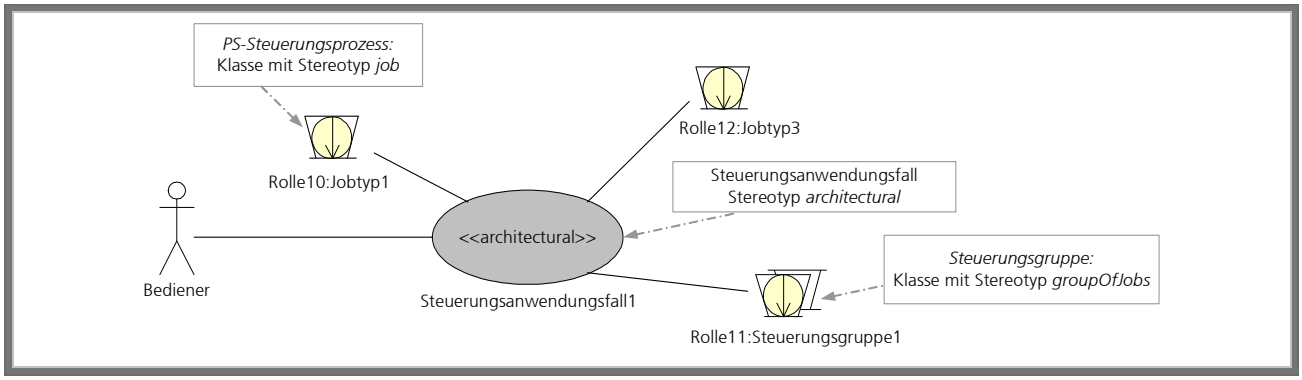

Bild 5.10: Beschreibung von Steuerungsanwendungsfällen

Der Produktionsagent als interner Akteur ist in diesem Zusammenhang nicht mehr zu berücksichtigen. Der Bediener der zu entwickelnden Maschinensteuerung ist weiterhin als ein externer Akteur in seinen unterschiedlichen Rollen zu verstehen (siehe Bild 5.10).

Die Berücksichtigung des Axioms der *Anwendbarkeit* einer Methode fordert die Verwendung durchgängig gleicher Beschreibungstechniken. Um aber direkt aus den Akteuren der Anwendungsfallsicht die Prozesse bzw. Funktionsblöcke ableiten zu können, wie es in der Dekompositionskonzeption festgelegt wurde, müssen auch für die Anwendungsfallsicht der Prozesssteuerungsebene interne Akteure beschrieben werden, die in der Prozesssicht als Steuerungsprozesse beschrieben werden können (siehe Bild 5.7). Aus diesem Grund wird der interne Akteur *PS-Steuerungsprozess* (Klasse mit dem Stereotyp *job*) für die Prozesssteuerungsebene eingeführt, der im Gegensatz zum Produktionsagenten als Steuerungsprozess verstanden harten Echtzeitanforderungen unterworfen ist. Anwendungsfälle werden in diesem Kontext als *Steuerungsanwendungsfall* (Anwendungsfall mit dem Stereotyp *architectural*) bezeichnet (siehe Tabelle 5.5 und Bild 5.10).

Sicht / Schicht	Anwendungsfall		Daten	Prozess		Implementierung	Einsatz	
System	**CRM** 1. Steuerungs-architektur 2. *Steuerungs-szenarien*	**CIM**		1. *Steuerungen-Dekomposi-tionsdiagramm* 2. Steuerungen-Diagramm	Steuerungen-Interaktions-diagramm		Installa-tions-diagramm	
	C Steuerungs-anwendungsfall	**CL**		Steuerungen-Bibliothek			PFK-PNK-Bibliothek	
Subsystem	**Nicht ODEMA**			1. PS-Funktions-blockarchitektur 2. PS-Komponenten-architektur		PS-Implemen-tierungs-diagramm		
				1. PS-Funktionsblock -Bibliothek 2. PS-Komponenten-Bibliothek		PS-Implemen-tierungen-Bibliothek		
Klasse			Daten-Diagramm	Komponenten-Spezifikation				
			Daten-Bibliothek		Kompo-nenten-Verhaltens-diagramm			
Methode	**Nicht UML**				Algorithmen-Spezifikation			

Tabelle 5.4: UML-Diagrammtyp-Rollen der Prozesssteuerungsebene (Unterstrichen = Modell-Bibliothek, kursiv = Konsistenzsichernder Diagrammtyp, x = Streichung durch Detail-Konzeption, C = Classifier, CIM = Classifier Interaction Model, CL = Classifier Lifecycle, CRM = Classifier Relationship Model, PS = Prozesssteuerung)

Die Echtzeitanforderung bringt es mit sich, dass PS-Steuerungsprozesse von außen durch Ereignisse bzw. Nachrichten *zwangsgesteuert* werden. Sie treffen keine autonomen Entscheidungen wie Produk-

tionsagenten. Über gleiche Eigenschaften verfügen sie aber im Hinblick auf ihre Funktion als Steuerungsprozesse bezüglich der Beschreibung des *System-Dekompositionsprinzips* in der Prozesssicht. PS-Steuerungsprozesse werden gemäß der Forderung nach durchgängigen Beschreibungstechniken (*Anwendbarkeit*, siehe Kapitel 2.3) ebenfalls mit der Spezifikation *Rolle:Typ* beschrieben. Es ist aber zu erwarten, dass ein PS-Steuerungsprozess i.d.R. über sehr viel weniger und weniger komplexe Rollen verfügt als ein Produktionsagent. Dies führt dazu, dass - anders als dies bei Produktionsagenten der Fall ist - kein Protokoll als Classifier Lifecycle eines Steuerungsanwendungsfalls vorgesehen werden muss (siehe Tabelle 5.4).

Die Einführung der UML-Beschreibungstechnik des internen Akteurs in den Kontext der Prozesssteuerungsebene führt nicht nur zu der geforderten Verbesserung der Anwendbarkeit der ODEMA-Methode, sondern vereinfacht die Konzeption des ODEMA-Beschreibungsmittels auch dahingehend, dass bezüglich der Prozess-, Implementierungs- und Einsatzsicht die gleichen UML-Diagrammtyp-Rollen vernachlässigt werden können wie in der Prozessführungsebene (siehe Tabelle 5.2 und 5.4).

Ausgangspunkt der bisher beschriebenen Detail-Konzeption der Prozessteuerungsebene war die Annahme, dass die zu spezifizierende Steuerung lediglich aus einem einzigen Produktionsagenten besteht. Die UML-Diagrammtyp-Rollen der Prozesssteuerungsebene haben aber noch eine zweite Funktion. Die Detail-Konzeption der UML-Diagrammtyp-Rollen der Prozessführungsebene hat ergeben, dass die Integration des *System-Dekompositionsprinzips* in das System-Interaktionsdiagramms gefordert ist. Daraus folgt, dass die Grenzen der Ebenen-Transparenz zwischen Prozesssteuerungs- und Prozessführungsebene verschoben wird und dass das Steuerungen-Interaktionsdiagramm als System-Dekomposition des System-Interaktionsdiagramms verstanden wird. Bei dieser Betrachtung ist die Prozessebene transparent.

Da die konsistente Zusammenführung der Spezifikationen der Prozessführungs- und Prozesssteuerungsebene in der Prozesssicht bzw. im Kontext des Installationsdiagramms auch in der physischen Sicht ausgeführt wird, bleibt die erste Aufgabenstellung der UML-Diagrammtyp-Rollen beider Ebenen - gemeint ist die Beschreibung der unterschiedlichen Ausgangssituationen bezüglich der Spezifikation eines PA-MAS - weiterhin erfüllt. Die Konsistenz wird hierbei durch das *Agenten-* bzw. *Steuerungen-Dekompositionsdiagramm* sichergestellt (siehe Tabelle 5.2 und 5.4, sowie Kapitel 5.5.5).

Abschließend muss aus der physischen Sicht des ODEMA-Beschreibungsmittels heraus noch die UML-Geräte-Beschreibung integriert werden. Das Begriffsmodell *Modulares Automatisierungsgerät* beschreibt den Gerätetyp PNK als denjenigen, der die PS-Steuerungsprozesse ausführt (siehe Tabelle 5.5). Somit ist der Gerätetyp PNK der Prozesssteuerungsebene zugeordnet.

In diesem Zusammenhang nimmt der Gerätetyp *Intelligentes Feldgerät* eine Sonderstellung ein, da es sensorische bzw. aktorische Funktionen und Steuerungsfunktionen in sich vereint. Diese Gerätekomponente ist somit der Prozesssteuerungs- und zugleich der Prozessebene zugeordnet. Da ein intelligentes Feldgerät seine Funktionen nach außen hin kapselt und nur eine Schnittstelle zu den Steuerungsfunktionen zur Verfügung stellt, kann die eingebettete Software dieses Automatisierungsgerätes als

eine Komponenten-Implementierung der Prozesssteuerungsebene betrachtet werden. Die Funktionalität eines programmierbaren intelligenten Feldgerätes wird durch eine vom Entwickler zu beschreibende Steuerungskomponente spezifiziert.

Stereotyp	Basis-Klasse	Eltern-Klasse	Eigenschaftswerte	Einschränkungen (*formal*/informal)
Steuerungsprozess *<<controlProcess>>*	Class	N/A	-	• *Self.isActive=true*
PS-Steuerungsprozess *<<job>>*	Class Object	• Steuerungsprozess • Interner Akteur	-	• *Self.isActive=true*
Steuerungsgruppe *<<groupOfJobs>>*	Class Object	Interner Akteur	-	• *Self.isActive=true* • Beschreibt Gruppe von PS-Steuerungsprozessen mit identischer Rolle
Steuerungsanwendungsfall *<<architectural>>*	Use Case	N/A	-	-

Tabelle 5.5: Abschnitt des UML-Profils für die Konzeption der UML-Beschreibungselemente der Prozesssteuerungsebene (N/A = keine Eltern-Klasse definiert, PS = Prozesssteuerung)

5.5.4 Detail-Konzeption der Prozessebene

Bereits die Konzeption der vereinfachten Ebenen-Sichten-Schichten-Darstellung hat gezeigt, dass für die Prozessebene weitreichende Vereinfachungen, also Reduzierungen der UML-Diagrammtyp-Rollen, möglich sind. Dies liegt im Wesentlichen darin begründet, dass einfache Feldgeräte keine Steuerungsfunktionen implementieren, sondern lediglich über sensorische bzw. aktorische Wandlungsfunktion verfügen. Das heißt, *Prozessvariablen* werden in physikalische Größen gewandelt oder aus diesen errechnet. Die Wandlungsfunktion ist gemäß dem Begriffsmodell *Modulares Automatisierungsgerät* (siehe Kapitel 3.1.5) als nicht programmierbar und somit als *eingebettete Softwarefunktionalität* zu betrachten. Zu berücksichtigen bleibt für die Prozessebene lediglich die UML-Geräte-Beschreibung und die UML-Kommunikationssystem-Beschreibung in Verbindung mit dem physischen Rahmen und dem Agenten-Rahmen (siehe Kapitel 5.2). Aus diesem Grund ist die Beschreibung der Prozesssicht und die der Implementierungssicht nicht auszuführen. Im selben Kontext kann prinzipiell auf jegliche Spezifikation der Kategorien *Classifier Lifecycle* und *Classifier Interaction Model* verzichtet werden.

Prinzipiell wird das einfache Feldgerät ausschließlich über das Kommunikationsmodul und ein entsprechendes Kommunikationssystem (z.B. Feldbus) mit der Geräte-Architektur verbunden werden. Um die Integration mit der Geräte-Architektur zu beschreiben, wird das bereits in den übrigen Ebenen berücksichtigte Installationsdiagramm in die Prozessebene integriert. Diese UML-Diagrammtyp-Rolle ist die einzige, alle Ebenen übergreifende UML-Diagrammtyp-Rolle.

Aus der logischen Sicht des Beschreibungsmittels ist nur das *Synchrone Kommunikationssystem* im Kontext der einfachen Feldgeräte zu spezifizieren (siehe Kapitel 3.1.6). Hierzu muss das ODEMA-Beschreibungsmittel eine UML-Beschreibungstechnik bereitstellen, die Sichten- und Ebenenübergreifend das *Application Layer Interface* (ALI) eines Feldbusses mit den angeschlossenen Felgerä-

ten in die Beschreibung einer Steuerungskomponente integriert. Diese Integration ist Teil der UML-Kommunikationssystem-Beschreibung, die in Kapitel 5.5.6 detailliert hergeleitet wird.

Neben dem ALI als Zugriffsschicht ist bezüglich der Datensicht die Spezifikation der Prozessvariablen notwendig, denn diese sind das Ergebnis der sensorischen und aktorischen Wandlungsfunktionen (siehe Kapitel 3.1.3). Auch hier werden gemäß der bereits dargestellten Konzeption Transportklassen verwendet. Aber Prozessvariablen enthalten neben dem eigentlichen Wert noch ergänzende Spezifikationen (z.B. Zeitstempel). Anstelle des *Daten-Diagramms* in den anderen Ebenen wird in der Prozessebene daher die UML-Diagrammtyp-Rolle *Prozessvariablen-Diagramm* eingeführt, welche ebenso als Teil der Spezifikation einer Steuerungskomponente angesehen werden kann (siehe Tabelle 5.6 und Bild 5.8).

Sicht / Schicht	Anwendungsfall		Daten	Prozess	Implementierung	Einsatz
System	CRM	CIM				Installations-diagramm
	C	CL				Feldgeräte-Bibliothek
Subsystem	Nicht ODEMA					
Klasse			Prozessvariablen-Diagramm			
			Prozessvariablen-Bibliothek			
Methode	Nicht UML					

Tabelle 5.6: Diagrammtypen der Prozessebene (Unterstrichen = Modell-Bibliothek, x = Streichung durch Detail-Konzeption, C = Classifier, CIM = Classifier Interaction Model, CL = Classifier Lifecycle, CRM = Classifier Relationship Model)

5.5.5 Detaillierte Konzeption der *Konsistenz-sichernden Diagrammtypen* und der Verfahren der Validation und Prüfung

Die Klassifizierung der *Konsistenz-sichernden Diagrammtypen* wurde in Kapitel 4.3 eingeführt und bezüglich ihrer Funktion zur Unterstützung der Prüfung auf syntaktische Konsistenz in Kapitel 5.4.2 dargestellt. Diese UML-Diagrammtyp-Rollen können syntaktische Konsistenz nicht aus sich selbst heraus herstellen, da sie vom Entwickler erstellt werden. Im Zusammenhang mit einem definierten Verfahren zur Prüfung der syntaktischen Konsistenz aber, ermöglichen sie die Formulierung von Prüfkriterien, die dann durch ein entsprechendes CASE-Tool oder durch den Entwickler auf ihre Erfüllung hin untersucht werden. Diese Verfahren werden hier als Verfahren *Prüfung der funktionalen Anforderungen* (Entwicklungsphase *Anforderungsdefinition*), *Prüfung des Pflichtenheftes auf Konsistenz* (Entwicklungsphase *Fachtechnische Lösungskonzeption*) und *Prüfung des objektorientierten Steuerungsentwurfs* (Entwicklungsphase *Systementwurf,* enthält zusätzlich das Verfahren *Test*) eingeführt.

In Kapitel 5.4.2 wurde die überwiegend gute Integration der Sichten durch die UML dargestellt. Ausnahme hierbei sind die UML-Diagrammtyp-Rollen der Kategorie *Classifier Lifecycle* und *Classifier Interaction Model*, für die bezüglich der Anwendungsfall- und Prozesssicht kein syntaktischer oder semantischer Zusammenhang definiert ist [UML_03]. In Kapitel 5.5.2 wurden bereits die Kollaborationen (*Collaborations*) eingeführt, die zwischen den Anwendungsfällen und Sequenzdiagrammen einen Zusammenhang beschreiben können. Für das ODEMA-Beschreibungsmittel werden deshalb die UML-Diagrammtyp-Rollen *System-* und *Steuerungsszenarien* eingeführt (siehe Tabelle 5.2 und 5.4 sowie Bild 5.11). Sie verknüpfen mittels einer Realisierung einen System- oder Steuerungsanwendungsfall mit einer *Anwendungsfall-Realisierung* (Kollaboration mit dem Stereotyp *ODuseCaseRealization*). Dieses erweiterte UML-Beschreibungselement ist so definiert, dass es nur ein einziges Sequenzdiagramm (System- oder Steuerungen-Interaktionsdiagramm) enthalten darf (siehe Tabelle 5.7). Bezüglich der Konsistenz der Kategorie *Classifier Lifecycle* ist in diesem Zusammenhang ausschließlich die Agenten-Protokoll-Bibliothek zu berücksichtigen. Es kann mit den Mitteln der UML aber kein Konsistenz-sicherndes Diagramm zur Verbindung eines Aktivitäts- (Agenten-Protokoll-Bibliothek) und eines Klassendiagramms (Agenten-Diagramm) definiert werden. Die syntaktische Konsistenz kann ausschließlich mittels Prüfkriterien durchgeführt werden (siehe Kapiteln 6.1.4).

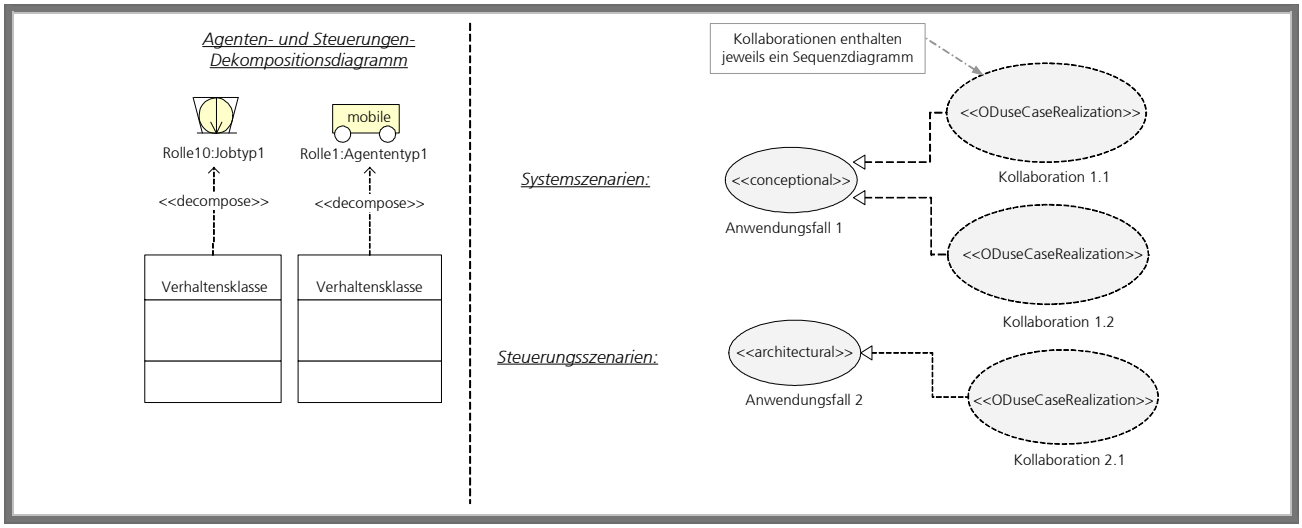

Bild 5.11: Exemplarische Darstellung der Konsistenz-sichernden UML-Diagrammtyp-Rollen

Stereotyp	Basis-Klasse	Eigenschaftswerte	Einschränkungen (*Formal*/informal)
Anwendungsfall-Realisierung *<<ODuseCaseRealization>>*	Collaboration	-	enthält nur ein UML-Sequenzdiagramm
Dekomposition <<decompose>>	Dependency	-	verbindet einen internen Akteur mit einer Verhaltensklasse

Tabelle 5.7: UML-Beschreibungselemente zur Konsistenz-Sicherung als Abschnitt des UML-Profils des ODEMA-Beschreibungsmittels (Es sind keine Eltern-Klasse definiert)

Durch die Anwendung des UML-Beschreibungselements *UML-Subsystem* kann bezüglich der Beschreibung des *Software-Dekompositionsprinzips* auf die zusammenfassende Organisation als UML-Beschreibungstechnik zurückgegriffen werden (siehe Kapitel 5.4.2). Da zwischen der Subsystemschicht und Klassenschicht - gilt nicht für die Prozessebene - ausschließlich statische Beschreibungen vorgese-

hen sind, wird hier durch diese UML-Beschreibungstechnik auf die Einführung Konsistenz-sichernder Diagrammtypen verzichtet. Der Übergang zwischen dem internen Akteur bzw. Steuerungsprozess und dem UML-Funktionsblock erfordert dagegen einen Konsistenz-sichernden Diagrammtyp (*Agenten-* und *Steuerungen-Dekompositionsdiagramm*). Zu diesem Zweck wird die *Dekomposition* (Abhängigkeit mit dem Stereotyp *decompose*) eingeführt. Die *UML-Verfeinerung* kann nicht eingesetzt werden, da sie das Dekompositionsprinzip nur allgemein beschreibt. Die Dekomposition dagegen beschreibt die System-Dekomposition eines internen Akteurs (siehe Bild 5.11 und Tabelle 5.7).

Im Kontext der Klassen- und Methodenschicht ist Konsistenz bezüglich der Verhaltensbeschreibung herzustellen. Die Konzeption des ODEMA-Beschreibungsmittels fordert zur Beschreibung des *Kausalprinzips* die Integration von Aktivitäts- und Zustandsdiagrammen (siehe Bild 5.1). Durch die eingeführte UML-Beschreibungstechnik *Steuerungskomponente* werden diese UML-Diagrammtypen auf das Komponenten-Verhaltensdiagramm (Klassenschicht) und die Algorithmen-Spezifikation (Methodenschicht) abgebildet. Da diese UML-Diagrammtyp-Rollen die feinste Dekompositionsstufe bezüglich der Verhaltensbeschreibung darstellen, sind sie Teil des Spezifikationsdokumentes *Objektorientierter Steuerungsentwurf* und müssen somit durch das Verfahren *Prüfung des objektorientierten Steuerungsentwurf* prüfbar sein (Methoden-Axiom *Prüfbarkeit*). Die Einführung eines Konsistenz-sichernden Diagrammtyps ist mit den Mitteln der UML nicht möglich (siehe Kapitel 5.4.2). Somit wird syntaktische Konsistenz dieser UML-Diagrammtyp-Rollen ausschließlich über die Komponenten-Spezifikation (Klassendiagramm) hergestellt.

Das Verfahren *Prüfung des objektorientierten Steuerungsentwurf* enthält neben der Prüfung auf syntaktische Konsistenz auch das Validationsverfahren *Test (*siehe Kapitel 4.4 und 6.3.3) . Aus der bisher durchgeführten Konzeption lässt sich ableiten, dass das Verfahren *Test* auf einem Vergleich der System- bzw. Steuerungen-Interaktionsdiagramme mit den Komponenten-Verhaltensdiagrammen beruht (siehe Tabelle 5.2 und 5.4). Die durch die Standard-UML und die bisher eingeführten Konsistenz-sichernden Diagrammtypen gewährleistete syntaktische Konsistenz sorgt dafür, dass für das Verfahren *Test* keine weiteren Konsistenz-sichernden Diagrammtypen eingeführt werden müssen.

Nach der Konzeption der syntaktischen Konsistenz bezüglich der Sichten und Schichten des ODEMA-Würfels müssen nun die Ebenen-Übergänge zusammengeführt werden. Hierbei sind die Klassen- und Methodenschicht nicht von Belang, da die Steuerungskomponenten nach außen gekapselt sind und Ebenen-übergreifend über ihre Schnittstellen-Klassen verbunden werden. Alle UML-Diagrammtyp-Rollen der Anwendungsfallsicht in den unterschiedlichen Ebenen dagegen sind ihrer Definition nach unabhängig voneinander und müssen nicht konsistent gehalten werden. Bezüglich des System- und Steuerungen-Interaktionsdiagramms sowie bezüglich der PF- und PS-Funktionsblockarchitektur gilt, dass diese in der Spezifikation zusammengeführt werden, d.h. sie können prinzipiell in einem gemeinsamen UML-Diagramm beschrieben werden. Nur aus Gründen der *Anwendbarkeit* werden sie getrennt beschrieben. Dies gilt ebenso für die Konsistenz bezüglich des PF- und des PS-

Implementierungsdiagramms. Um die syntaktische Konsistenz bezüglich der Ebenen-Übergänge zu prüfen, müssen somit keine weiteren Konsistenz-sichernden Diagrammtypen eingeführt werden.

5.5.6 Erweiterte Konzeption des *Installationsdiagramms*

Im Kontext der Konzeption des Vorgehensmodells der ODEMA-Methode wurden die UML-Beschreibungstechniken *UML-Geräte-Beschreibung* und *UML-Kommunikationssystem-Beschreibung* bereits eingeführt (siehe Kapitel 4.2.1). Durch die Ebenen-spezifische Konzeption des ODEMA-Beschreibungsmittels wurden diese UML-Beschreibungstechniken dem Installationsdiagramm (Einsatzdiagramm) zugeordnet, da allgemein Kommunikationssysteme und Geräte in der UML mittels UML-Knoten modelliert werden (siehe Kapitel 5.2 sowie 5.5.2 bis 5.5.4 und [BOOCH_99, OESTER_99b]). UML-Knoten in Einsatzdiagrammen beschreiben aber lediglich die örtliche Verteilung von UML-Komponenten und verfügen darüber hinaus über keine weiteren Eigenschaften. Die Beschreibung von einfachen und eigenschaftslosen Punkt-zu-Punkt-Verbindungen mittels UML-Kommunikationsverbindungen (*communication associations*) ist dabei zur Beschreibung von Kommunikationssystemen ebenfalls nicht ausreichend, da z.B. Mehr-Punkt-Topologien nicht beschreibbar sind (siehe Kapitel 3.1.6). Um die Begriffsmodelle *Modulares Automatisierungsgerät* und das Begriffsmodell des Kommunikationssystems auf die entsprechenden UML-Beschreibungstechniken abbilden zu können, müssen die zur Verfügung stehenden elementaren UML-Beschreibungstechniken eingeführt und bewertet werden.

5.5.6.1 Varianten der Konzeption

Zwei elementare UML-Beschreibungstechniken stehen prinzipiell für die durchzuführende Abbildung zur Verfügung. Zum einen bieten sich die UML-Erweiterungsmechanismen *Stereotyp* und *Eigenschaftswert* an, mit denen sich das Standard-Modellelement *UML-Knoten* präzisieren bzw. mit erweiterten Eigenschaften versehen lässt [UML_03].

Die zweite elementare UML-Beschreibungstechnik ist die *Beschreibende Implementierung*. Eine beschreibende Klasse (Stereotyp *descriptionClass*) wird mittels einer *Implementierungsbeziehung* (Abhängigkeit mit dem Stereotyp *reside*) mit einer *Beschreibenden UML-Komponente* (Stereotyp *deviceDescription* oder *comSysDescription*) verbunden. Diese beschreibende UML-Komponente ist nicht als Implementierung einer Steuerungsfunktionalität zu verstehen, sondern ausschließlich als eine Zusammenfassung für UML-Beschreibungselemente, mit deren Hilfe sich UML-Knoten erweitert beschreiben lassen (siehe Bild 5.12). Beschreibende Komponenten sind untrennbar mit dem jeweiligen UML-Knoten verbunden und nicht portierbar.

Tabelle 5.8 stellt Vor- und Nachteile der vorgestellten UML-Beschreibungstechniken gegenüber. Die Bewertung der Varianten ergibt, dass beide elementaren UML-Beschreibungstechniken kombiniert werden müssen, um die gestellte Aufgabe der Abbildung der Begriffsmodelle in die UML zu ermöglichen. Um die geforderte Typisierung von Geräten und Kommunikationssystemen zu ermöglichen (siehe Tabelle 3.5 und Bild 3.2) erhalten die UML-Knoten entsprechende Stereotypen. Um die bezüglich

des *Synchronen Kommunikationssystems* notwendige Zusammenführung von Prozess- und Einsatzsicht zu realisieren – als die Möglichkeit des Zugriffes einer Steuerungskomponente auf die Dienstes des ALI - , wird eine beschreibende Implementierung eingesetzt. Weitere Eigenschafen von Automatisierungsgeräten und Kommunikationssystemen werden über die Attribute der beschreibenden Klasse modelliert.

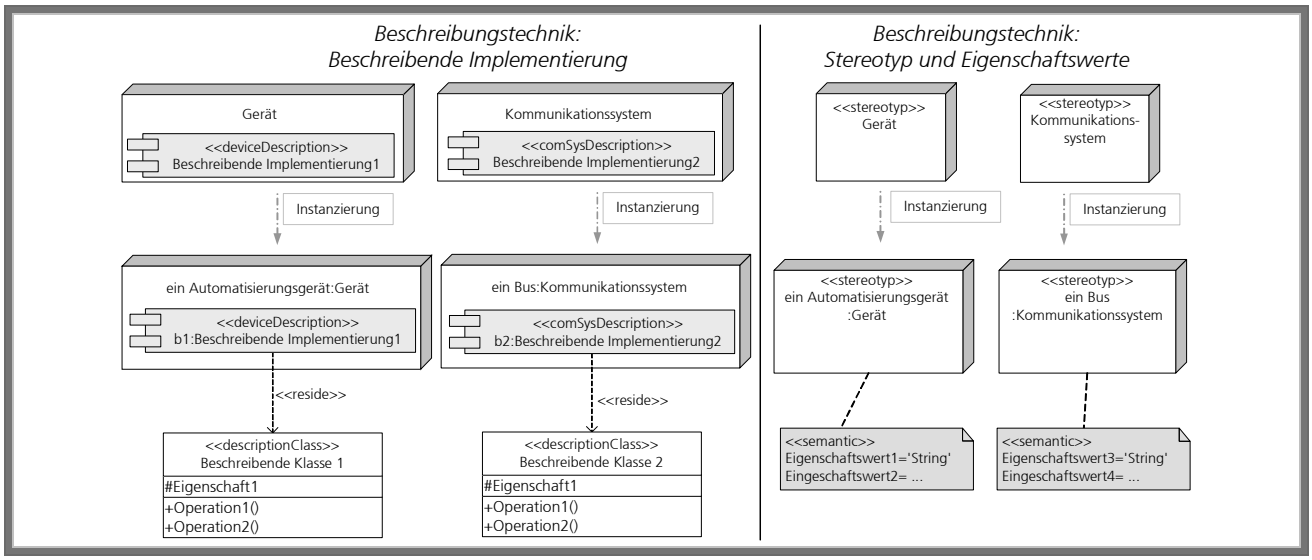

Bild 5.12: UML-Beschreibungstechniken zur Erweiterung der Semantik eines UML-Knotens

Elementare UML-Beschreibungstechnik	Vorteil	Nachteil
Stereotyp und Eigenschaftswerte	UML-Knoten lassen sich mittels Stereotypen als Gerätetypen (z.B. PNK) und Kommunikationssystemtypen (z.B. Profibus) beschreiben	Eigenschaftswerte lassen sich nicht von Steuerungskomponenten beeinflussen (Zusammenführung von Prozess- und Einsatzsicht)
	Über Stereotypen lässt sich einem UML-Knoten auch eine angepasste grafische Syntax zuweisen (*Anwendbarkeit*)	UML-Diagramme werden durch die Verwendung von Eigenschaftswerten unübersichtlich (*Anwendbarkeit*)
Beschreibende Implementierung	UML-Knoten erhalten wie UML-Klassen Operationen und Eigenschaften (*Attribute*)	Ein UML-Knoten lässt sich nicht als Gerätetyp oder Feldbustyp klassifizieren
	Mittels beschreibender Klassen lassen sich Eigenschaften der UML-Knoten von Steuerungskomponenten beeinflussen	

Tabelle 5.8: Gegenüberstellung der elementaren UML-Beschreibungstechniken

5.5.6.2 Konzeption der UML-Geräte-Beschreibung

Grundlage der Typisierung der UML-Knoten als Automatisierungsgeräte ist die Verknüpfung der Stereotypen von UML-Knoten mit den damit verbundenen Eigenschaften eines Gerätes, welche über eine beschreibende Komponente festgelegt werden. Um die unterschiedlichen Eigenschaften der im Kontext des *Modularen Automatisierungsgerätes* eingeführten Gerätetypen (PNK, PFK usw.) zu beschreiben, müssen die entsprechenden beschreibenden Klassen hergeleitet werden. Aus Sicht der UML lässt sich dies mittels eines Klassendiagramms und entsprechender UML-Vererbungsbeziehungen (*Generalization*) bewerkstelligen. Zunächst wird eine allgemeine beschreibende Klasse für alle Automatisierungsgeräte eingeführt (UML-Klasse *Device*, siehe Bild 5.13) und gemäß dem Begriffsmodell *Modulares Automatisierungsgerät* in programmierbare und nicht-programmierbare Automatisierungsgeräte

unterschieden. Eine weitere Stufe der Vererbung, die dann die Gerätetypen beschreibt, schließt den generischen Teil der Modellierung der beschreibenden Klassen ab.

Die abschließende Stufe des Klassendiagramms ist anwendungsbezogen, d.h. die in einer Spezifikation eingesetzten beschreibenden Klassen werden aus den generischen Klassen abgeleitet. Hier wird z.B. ein spezieller Sensortyp (z.B. der Laserscanner LMS-200) als Typ durch eine anwendungsbezogene, beschreibende Klasse spezifiziert (Klasse *LMS-200*, siehe Bild 5.13). Ausschließlich diese anwendungsbezogenen, beschreibenden Klassen werden von der beschreibenden Implementierung realisiert. Hierbei muss vorausgesetzt werden, dass der Stereotyp eines UML-Knotens mit den entsprechenden Eltern-Klassen der beschreibenden Klasse übereinstimmt (siehe Bild 5.13). Ein Verfahren zur Prüfung oder Validation kommt hierbei nicht zum Einsatz, da dieser Vorgang nicht Teil der Spezifikation einer Steuerung ist. UML-Geräte-Beschreibungen stehen dem Entwickler durch die UML-Diagrammtyp-Rollen *Feldgeräte-* oder *PFK-PNK-Bibliothek* zur Verfügung und werden entsprechend der Aufgabenstellung ausgewählt.

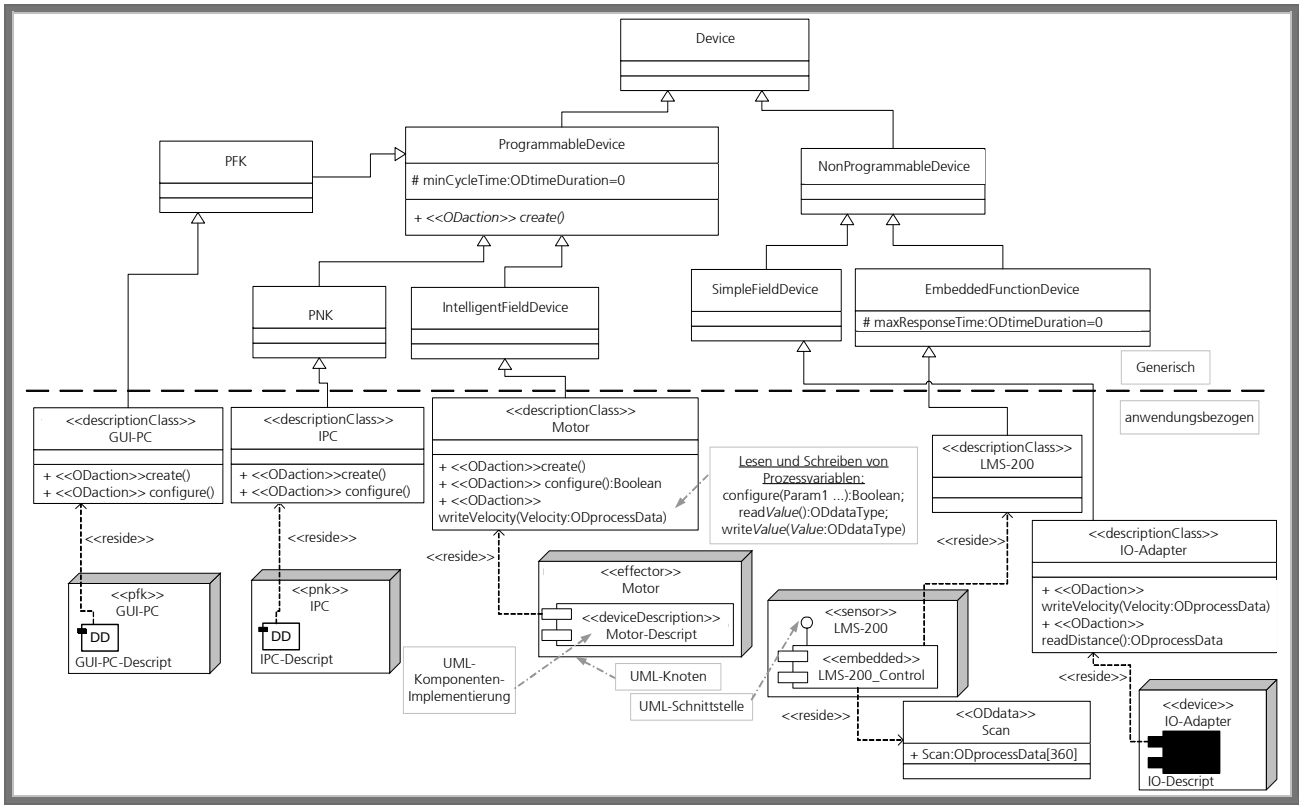

Bild 5.13: Konzeption der UML-Geräte-Beschreibung und exemplarische Beschreibung der Anwendung

Gemäß der Konzeption des Begriffsmodells *Modulares Automatisierungsgerät* weisen die programmierbaren Automatisierungsgeräte Schnittstellen (API und ALI) auf, die von den Steuerungskomponenten verwendet werden (siehe Kapitel 0). Da für jedes Kommunikationsmodul ein separates ALI zur Verfügung steht, ist diese Schnittstelle nicht der UML-Geräte-Beschreibung zuzuordnen, sondern der UML-Kommunikationssystem-Beschreibung (siehe Kapitel 5.5.6.3).

Die Schnittstelle zum Organisationsmodul, also zum Betriebssystem des Gerätes, als Teil der API aber gehört zur UML-Geräte-Beschreibung. Um allerdings die Plattform-Unabhängigkeit des Beschreibungsmittels zu erhalten, können hier nicht die typischen Dienste eines steuerungstechnischen Be-

triebssystems beschrieben werden, wie z.B. der Aufruf von Timern oder die Kommunikation über Ereignis-Warteschlangen. Es wird daher eine Plattform-unabhängige Operation zur lokalen Konfiguration des Gerätes aus Sicht einer Steuerungskomponente eingeführt (Operation *configure()*). Diese Operation ist aber anwendungsspezifisch und kann somit durch verschiedene Parameter ergänzt werden je nach zu beschreibendem Gerätetyp (siehe Bild 5.13). Durch Hinzufügung dieser Operation in den Kontext der beschreibenden Klasse erfährt diese eine Erweiterung ihrer Semantik. Die Instanz der beschreibenden Klasse ist ergänzend als *Fassade* (Entwurfsmuster *Facade* [GAMMA_97]) anzusehen, d.h. die angebotenen Operationen eines Betriebssystems werden in einer implementierungsunabhängigen Abstraktion zusammengefasst. Die Operation *create()* wird hierbei als Konstruktor der beschreibenden Klasse verstanden, welche darüber hinaus nur eine Instanz bilden darf, da die Funktionen des Automatisierungsgeräts nur einmal zur Verfügung stehen.

Bezüglich des Gerätetyps *Intelligentes Feldgerät* muss noch eine Operation zum Lesen bzw. Schreiben von Prozessvariablen eingeführt werden, um die Anforderung nach einer Schnittstelle zum Prozessgrößen-Verarbeitungsmodul zu erfüllen. Dies sind die anwendungsbezogenen Operationen *readValue()* und *writeValue()*. Beide Operationen verwenden zur Beschreibung der jeweiligen Prozessvariable den Datentyp *ODdataType*, der die Basisklasse der elementaren, anwendungsbezogenen Datentypen des ODEMA-Beschreibungsmittels ist (Die Herleitung der elementaren Datentypen ist in Kapitel 5.5.7 beschrieben).

Abschließend ist für die Konzeption der UML-Geräte-Beschreibung noch das Methoden-Axiom *Temporalprinzip* zu berücksichtigen. Die eingeführten Operationen sind bezüglich der UML als Deklaration zu verstehen, d.h. über ihre Realisierung und damit über ihr zeitliches Verhalten ist nichts ausgesagt [UML_03]. Da die hier eingeführten Operationen ausschließlich auf lokale Ressourcen zugreifen, kann von einer nahezu verzögerungsfreien Ausführung der Operationen ausgegangen werden. Hierbei wird vorausgesetzt, dass es zu keinen Konflikten zwischen Steuerungskomponenten beim Zugriff auf die identische Operation kommt.

Die verzögerungsfreie und nicht unterbrechbare Ausführung einer Operation definiert die UML als *Aktion* (*Action*). Um die eingeführten Operationen als Aktionen kenntlich zu machen, erhalten diese den Stereotyp *ODaction*. Ebenso ergibt sich das Attribut *minCycleTime* aus der Berücksichtigung des *Temporalprinzips* bezüglich des Begriffsmodells *Modulares Automatisierungsgerät*. Es beschreibt die kürzest mögliche Zykluszeit einer PNK, PFK oder eines Intelligenten Feldgerätes in Bezug auf einen zyklisch gesteuerten Steuerungsprozess als *offered QoS*.

Die nicht-programmierbaren Automatisierungsgeräte, also die einfachen Feldgeräte, verfügen über keinerlei Spezifikation ihrer Schnittstellen API und ALI. Ihre Funktionalität ist aus Sicht einer Steuerungskomponente (Prozesssicht) ausschließlich über das Kommunikationsmodul zugänglich. Der Zugriff auf diese Geräte (*Synchrones Kommunikationsmodell*) geschieht also ausschließlich über das Kommunikationssystem. Die Deklaration entsprechender Operationen wird somit im Kontext der beschreibenden Klassen eines Kommunikationssystems formuliert. Die Lese- und Schreib-Operationen,

die bezüglich der beschreibenden Klasse eines einfachen Feldgerätes formuliert werden, sind daher nicht zugänglich und durch die *Eingebettete Implementierung* (UML-Komponente mit dem Stereotyp *embedded*) gekapselt (siehe Klasse *IO-Adapter* in Bild 5.13).

Stereotyp	**Basis-Klasse**	**Eigenschaftswerte**	**Einschränkungen (*formal* / informal)**
Atomare Aktion <<ODaction>>	Stimulus	-	*concurrency = cck_guarded*
	Operation		
	Procedure		
Nicht-unterbrechbare Aktivität <<ODactivity>>	Stimulus	ODrequiredTime (siehe Tabelle 6.5b)	*concurrency = cck_guarded*
	Operation		
	Procedure		
Eingebettete Implementierung <<embedded>>	Component	-	• ausführbare Komponenten-Implementierung • unlösbar mit dem Automatisierungsgerät verbunden
	ComponentInstance		
Beschreibende Implementierung eines Gerätes <<deviceDescription>>	Component	-	-
	ComponentInstance		
Beschreibende Klasse <<descriptionClass>>	Class	-	Klasse ist ein *Singleton* [GAMMA_97]
Prozessnahe Komponente <<pnk>>	Node	-	-
	NodeInstance		
Prozessferne Komponente <<pfk>>	Node	-	-
	NodeInstance		
Aktor <<effector>>	Node	-	-
	NodeInstance		
Sensor <<sensor>>	Node	-	-
	NodeInstance		
Feldgerät <<device>>	Node	-	Beschreibt ein einfaches oder intelligentes Feldgerät
	NodeInstance		

Tabelle 5.9: UML-Geräte-Beschreibung als Abschnitt des UML-Profils des ODEMA-Beschreibungsmittels (Es werden keine Eltern-Klassen verwendet, PNK = Prozessnahe Komponente, PFK = Prozessferne Komponente)

Eine Sonderstellung nimmt die eingebettete Implementierung im Zusammenhang mit einer eingebetteten Steuerungsfunktionalität ein. Jedes Automatisierungsgerät mit Ausnahme eines einfachen Feldgerätes kann prinzipiell über eine eingebettete Steuerungsfunktionalität verfügen und ist somit nicht mehr als programmierbar anzusehen. Nach der Definition einer Softwarekomponente (siehe Kapitel 2.1) ist diese als Implementierung ausschließlich durch ihre Schnittstellen-Klassen spezifiziert. Hierbei bleibt die Realisierung zwar verborgen, aber ein zeitliches Verhalten (*Temporalprinzip*) ist trotzdem von außen beobachtbar. Das Attribut *maxResponseTime* modelliert die Eigenschaft der maximalen Antwortzeit (*offered QoS*) einer eingebetteten Komponenten-Implementierung, also die Zeit, die maximal vergeht, bis die Implementierung bzw. das Gerät auf eine eingehende Nachricht eine Antwort sendet. Darüber hinaus werden keine Operationen eingeführt, da der Zugriff auf die eingebettete Implementierung durch die explizit beschriebenen Schnittstellen-Klassen realisiert wird. Komplexe Datentypen im Sinne einer Transportklasse können ebenso implementiert werden, sofern sie Teil der Beschreibung der Schnittstellen-Klassen sind (*Transportobjekte*).

Zu beachten ist, dass bisher im Kontext der UML-Geräte-Beschreibung ausschließlich UML-Knoten für die UML-Diagrammtyp-Rolle *Geräte-Bibliothek* hergeleitet wurden. Instanzen dieser UML-Knoten werden im Installationsdiagramm eingesetzt. Hierbei bleiben die Gerätetypen bzw. UML-Knoten notwendigerweise erhalten. UML-Knoten-Instanzen und die jeweilige Instanz der beschreibenden Implementierung eines Gerätes werden erzeugt und erhalten einen eindeutigen Namen.

Tabelle 5.9 zeigt die Konzeption der UML-Geräte-Beschreibung als UML-Profil. Bild 5.14 zeigt die hier eingeführten grafischen Symbole für Stereotypen. Das Symbol für die beschreibende Implementierung eines Gerätes ist in Bild 5.13 bereits angewendet worden und zeigt deutlich die Verbesserung bezüglich der Übersichtlichkeit der UML-Beschreibung und schafft somit eine Verbesserung der Anwendbarkeit.

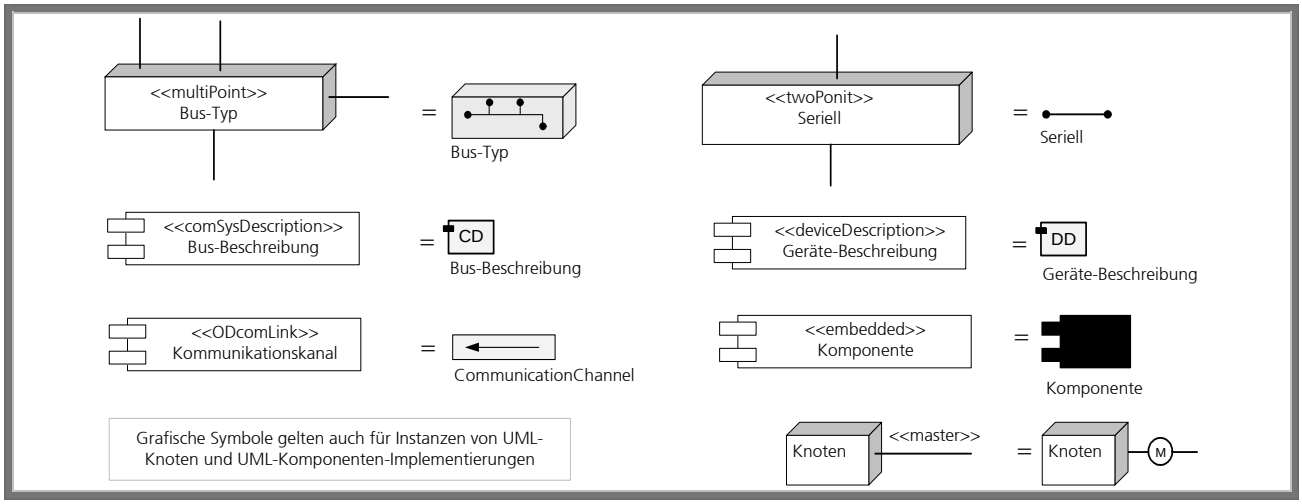

Bild 5.14: Grafische Syntax der Stereotypen für das UML-Profil der *UML-Geräte-* und *UML-Kommunikationssystem-Beschreibung* (CD = comSysDescription, DD = deviceDescirption, M = Master)

5.5.6.3 Konzeption der UML-Kommunikationssystem-Beschreibung

Gemäß der Konzeption der *Beschreibenden Implementierung* werden Geräte und Kommunikationssysteme als UML-Knoten beschrieben. Aus diesem Grund ist die Verbindung zweier Geräte mittels einer eigenschaftslosen UML-Kommunikationsverbindungen nicht mehr zulässig, da diese Modellierungstechnik zu einer semantischen Lücke des Installationsdiagramms führt. Hieraus folgt unmittelbar, dass eine UML-Kommunikationsverbindung stets die Instanzen eines Gerätes und eines Kommunikationssystems verbindet.

Als erster Schritt der Konzeption der UML-Kommunikationssystem-Beschreibung ist das ALI für den synchronen Zugriff auf das Kommunikationssystem zu integrieren (*Synchrones Kommunikationssystem*). Auf diese Weise wird die Schnittstelle zur UML-Geräte-Beschreibung hergestellt. Dies geschieht gemäß der Beschreibungstechnik *Beschreibende Implementierung* mittels der bereits eingeführten beschreibenden Klasse als implementierungs-unabhängige Zugriffsschicht. Zur Unterscheidung von einer beschreibenden Implementierung für Geräte wird eine beschreibende Implementierung für Kommunikationssysteme (Stereotyp *comSysDescription*) eingeführt. Da die gleiche Modellierungstechnik verwendet wird, wie bei der Herleitung der UML-Geräte-Beschreibung, kann auch hier mit Hilfe eines

Klassendiagramms ein generisches Klassendiagramm beschrieben werden, aus dem die anwendungsspezifischen, beschreibenden Klassen abgeleitet werden.

Im generischen Teil des Klassendiagramms werden aus der allgemeinen Klasse eines Kommunikationssystems (Klasse *CommunicationSystem*) die spezialisierten Klassen zur Beschreibung eines synchronen Kommunikationssystems (Klasse *SynchronComSystem*) und zur Spezifikation eines *Kommunikationskanals* (Klasse *AsynchronComSystem*) abgeleitet (siehe Bild 5.15).

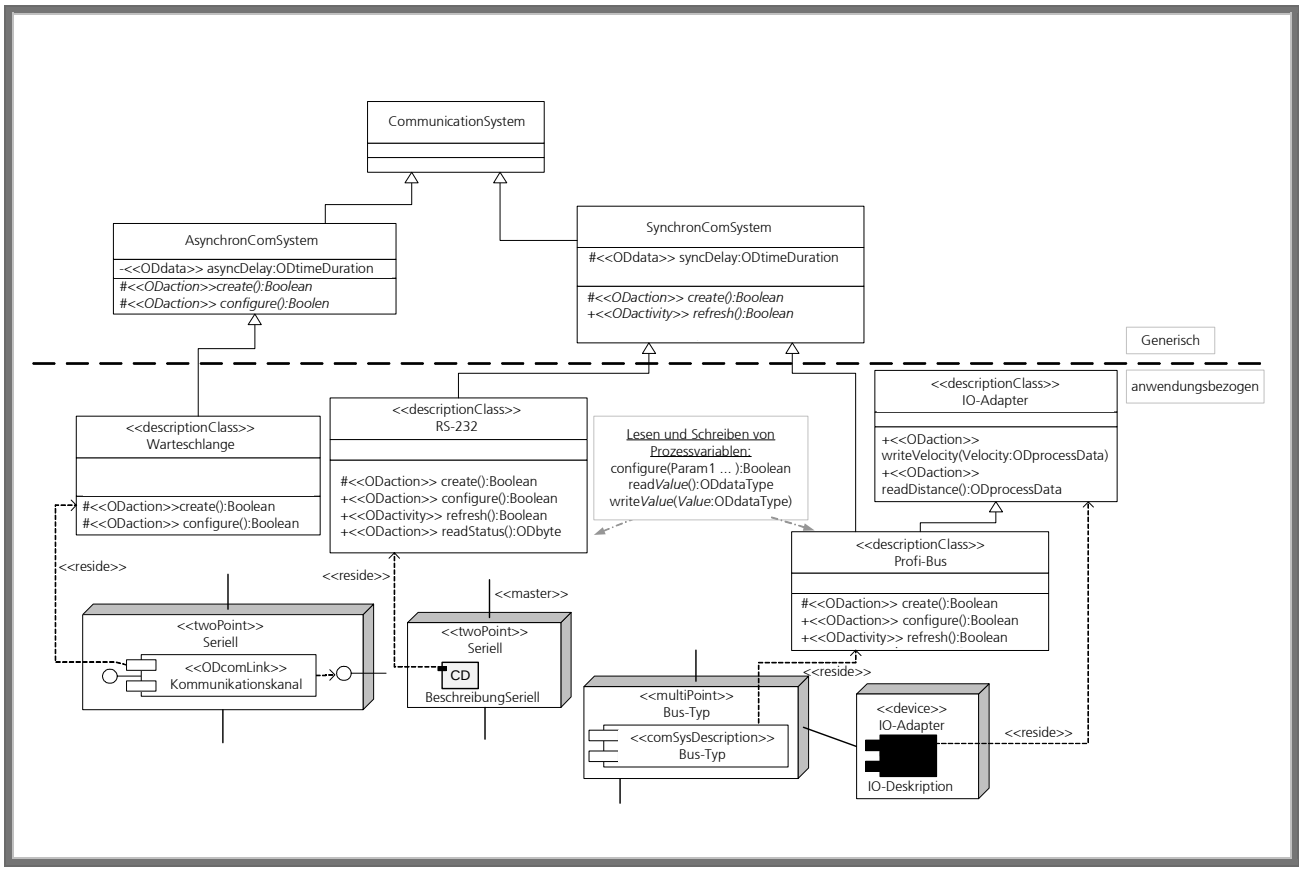

Bild 5.15: Konzeption der UML-Kommmunikationssystem-Beschreibung und exemplarische Beschreibung der Anwendung

Im Unterschied zur UML-Geräte-Beschreibung sind neben den vom Entwickler zu wählenden Lese- und Schreib-Operationen, die sich auf das Kommunikationssystem im Ganzen beziehen (siehe z.B. *readStatus()* der beschreibenden Klasse *RS-232*), die bereits definierten Operationen der einfachen Feldgeräte zu integrieren, sofern diese mit dem Kommunikationssystem verbunden sind. Dies wird mittels einer weiteren Ableitung (*Vererbung*) der beschreibenden Klasse des Kommunikationssystems aus den beschreibenden Klassen der einfachen Feldgeräte bewerkstelligt (siehe beschreibende Klassen *Profi-Bus* und *IO-Adapter* in Bild 5.15). Die als öffentlich (*public*) deklarierten Operationen der beschreibenden Klassen sind in der abgeleiteten Klasse weiterhin öffentlich und somit Teil des ALI.

Eine weitere Eigenschaft des Begriffsmodells des Kommunikationssystems ist die Unterscheidung in einfache *Punkt-zu-Punkt-Systeme* und *Bus-Systeme* (*Mehr- oder Zweipunkttopologie*, siehe Kapitel 3.1.6). Zu diesem Zweck werden für die UML-Kommunikationssystem-Beschreibung die Typisierungen *Zwei-Punkt-System* (Stereotyp *twoPoint*) und *Mehr-Punkt-System* (Stereotyp *multiPoint*) eingeführt. Das Attribut *syncDelay* der Klasse *SynchronComSystem* beschreibt für beide Systeme die Zykluszeit des synchronen Kommunikationssystems, also die Ausführung der Operation *refresh()*, die innerhalb eines

Zyklus alle Prozessvariablen von den Sensoren liest und entsprechende Prozessvariablen in die Aktoren schreibt (Prozessabbild).

Abschließend muss die Geräterolle (*Master* oder *Slave*) integriert werden (siehe Kapitel 3.1.6). Sie bezieht sich stets auf den lokalen Kommunikationsprozess, der ein Gerät mit einem Kommunikationssystem verbindet. Für die UML-Beschreibung bedeutet dies, dass die UML-Kommunikationsverbindung zwischen einem Gerät und einem Kommunikationssystem diese Eigenschaft erhält. Der Stereotyp *master* beschreibt eine UML-Kommunikationsverbindung, die ein Gerät als *Master* mit dem Kommunikationssystem verbindet. Das Modell des synchronen Kommunikationssystems sieht hierbei vor, dass für ein Kommunikationssystem jeweils nur einmal die *Master*-Rolle vergeben werden darf. Im Zusammenhang mit dem Einsatz von Kommunikationskanälen (Stereotyp *ODcomLink*) zur Beschreibung des Installationsdiagramms ist die Geräte-Rolle transparent und wird somit nicht beschrieben. Hieraus folgt unmittelbar, dass der Entwickler sich für eine Beschreibungsvariante (Synchrones Kommunikationssystem oder Kommunikationskanal) bezogen auf die Instanz eines Kommunikationssystems in einem Installationsdiagramm entscheiden muss. Eine gemeinsame Anwendung beider Beschreibungsvarianten ist in diesem Zusammenhang nicht möglich.

Die Konzeption des Kommunikationskanals unterscheidet sich von der des synchronen Kommunikationssystems insofern, als dass die beschreibende Klasse nicht als ALI verstanden wird. Kern der Beschreibung des Kommunikationskanals ist die Transparenz des Kommunikationssystems selbst (siehe Kapitel 3.1.6). Die Basis-Operationen *create()* und *configure()* werden während der Instanzierung, d.h. zum Zeitpunkt der Spezifikation aufgerufen und sind für die Steuerungskomponenten nicht zugänglich (*protected*).

Im Zusammenhang mit der Beschreibung der Verteilung der Implementierungen von Steuerungskomponenten ist das Methoden-Axiom *Temporalprinzip* zu berücksichtigen. Die Kommunikation zwischen Steuerungsprozessen, die Gerätegrenzen überschreitet, ist mit unterschiedlichen und zu spezifizierenden Zeitverzögerungen behaftet (siehe Kapitel 3.1.6),. Die generische Klasse *AsynchronComSystem* berücksichtigt dies durch das Attribut *asyncDelay*, d.h. die Zeit, die eine Nachricht zur Übertragung benötigt. Diese zeitliche Verzögerung spezifiziert die zeitliche Verzögerung zwischen dem Aufruf einer als Nachricht verstandenen Operation auf der exportierten Schnittstellen-Klasse des Kommunikationskanals und dem Aufruf der identischen Operation auf der importierten Schnittstellen-Klasse (siehe Bild 5.15).

Hieraus folgt, dass jeder Kommunikationskanal im einfachsten Fall jeweils eine Schnittstellen-Klasse exportiert und eine identische importiert. Unter der Berücksichtigung, dass auch *Broadcast*- oder *Multicast*-Kommunikation möglich sein sollen, können Kommunikationskanäle auch mehrere identische Schnittstellen-Klassen importieren und dabei eine einzige Schnittstellen-Klasse exportieren, welche ebenfalls identisch ist zu den Importierten.

Aus der Gleichheit der UML-Beschreibungstechnik für Geräte und Kommunikationssysteme geht unmittelbar hervor, dass auch UML-Kommunikationssystem-Beschreibungen als Teil der Feldgeräte- bzw.

PFK-PNK-Bibliothek anzusehen sind. Im Fall des synchronen Kommunikationssystems können diese auch im Verbund mit entsprechenden Feldgeräten in der Feldgeräte-Bibliothek abgelegt werden. Hierbei ist aber zu beachten, dass UML-Knoten nicht mittels UML-Kommunikationsverbindungen miteinander verbunden werden können, sondern ausschließlich mittels UML-Knoten-Instanzen (*NodeInstance*). Soll also z.B. die Instanz eines Mehr-Punkt-Systems zusammen mit einer Anzahl Sensoren und Aktoren als wieder zu verwendendes UML-Beschreibungselement eingesetzt werden, so wird dieses als Einsatzdiagramm mit so genannten anonymen Instanzen umgesetzt. Dies bedeutet, dass die UML-Knoten-Instanzen keine Namen haben. Diese erhalten sie erst, wenn sie in einem Installationsdiagramm zur Spezifikation einer Steuerung eingesetzt werden.
Die notwendigen Ergänzungen des UML-Profils des ODEMA-Beschreibungsmittels durch die Konzeption der UML-Kommunikationssystem-Beschreibung sind in Tabelle 5.10 beschrieben.

Stereotyp	**Basis-Klasse**	**Einschränkungen (*formal*/informal)**
Geräterolle *Master* <<master>>	Communication association	**-**
Beschreibende Implementierung eines Kommunikationssystems <<comSysDescription>>	Component	-
	ComponentInstance	
Zwei-Punkt-System <<twoPoint>>	Node	• enthält ausschließlich eine beschreibende Implementierung eines Kommunikationssystems oder eine beliebige Anzahl Kommunikationskanäle • es können nur zwei Geräte angeschlossen werden • wird eine beschreibende Implementierung eines Kommunikationssystems verwendet, dann muss ein Geräte die Rolle des Masters übernehmen
	NodeInstance	
Mehr-Punkt-System <<multiPoint>>	Node	• enthält ausschließlich eine beschreibende Implementierung eines Kommunikationssystems oder eine beliebige Anzahl Kommunikationskanäle • wird eine beschreibende Implementierung verwendet, dann muss ein Gerät die Rolle des Masters übernehmen
	NodeInstance	
Kommunikationskanal <<ODcomLink>>	Component	• zeitliche Reihenfolge der Nachrichten bleibt erhalten • jede importierte Schnittstellen-Klasse wird auch exportiert und umgekehrt
	ComponentInstance	

Tabelle 5.10: UML-Kommunikationssystem-Beschreibung als Abschnitt des UML-Profils des ODEMA-Beschreibungsmittels (Es werden keine Eltern-Klassen und keine Eigenschaftswerte verwendet.)

5.5.7 Konzeption der elementaren Datentypen

Aus der Perspektive der Datensicht ist jede Transportklasse als komplexer Datentyp anzusehen. Zur Beschreibung solcher Datentypen werden atomare, so genannte elementare Datentypen benötigt (Methoden-Axiom *Software-Dekomposition*). Diese haben die gleiche Bedeutung wie Datentypen in Programmiersprachen (z.B. IEC-61131-Sprachen, siehe Kapitel 3.1.3). Im Unterschied hierzu definiert die UML elementare Datentypen ohne Berücksichtigung ihres Speicherbedarfs. Der elementare Datentyp *Integer* beschreibt also die Menge der ganzen Zahlen, ohne irgend eine Angabe über ihre Begrenzung zu machen [UML_03]. Zudem versteht die UML elementare Datentypen (Stereotyp *datatype*) prinzipiell

als Klassen mit Attributen und Operationen, die aber als Instanz keine Identität haben. Somit werden diese Datentypen ausschließlich im Kontext der Beschreibung von Attributen angewandt.

Die in Bild 5.16 gezeigten elementaren Datentypen der UML (*String*, *Integer* und *Boolean*) sind gemäß ihrer Definition als implementierungs-unabhängige Datentypen zu verstehen. Die Standard-Datentypen der UML sind aus Sicht der Spezifikation von Steuerungen aber nicht vollständig und ergänzende elementare Datentypen sind somit einzuführen (siehe Kapitel 3.1.3).

Zunächst fehlt die Gleitkommazahl (Klasse *ODreal*). Dieser Datentyp wird zur Darstellung der Prozessvariablen benötigt. Zusammen mit der Beschreibung einer physikalischen Einheit (Datentyp *ODunit*), dem Zeitstempel (Datentyp *ODanyDate*) und einem unteren und oberen Grenzwert (Attribute *minValue* und *maxValue*) beschreibt dieser Datentyp den zusammengesetzten und elementaren Datentyp *ODprocessData* (Prozessvariable).

Eine Zeitdauer in Millisekunden spezifiziert der Datentyp *ODtimeDuration*. Er wurde bereits zur Spezifikation des *offered QoS* von Geräten und Kommunikationssystemen verwendet (siehe Bilder 5.13 und 5.15). Der Datentyp *ODbyte* beschreibt einen Byte-Wert (Wertebereich 0 bis 255), da dieser i.d.R. zur Beschreibung von Status- bzw. Konfigurationswerten für Geräte und Kommunikationssysteme verwendet wird.

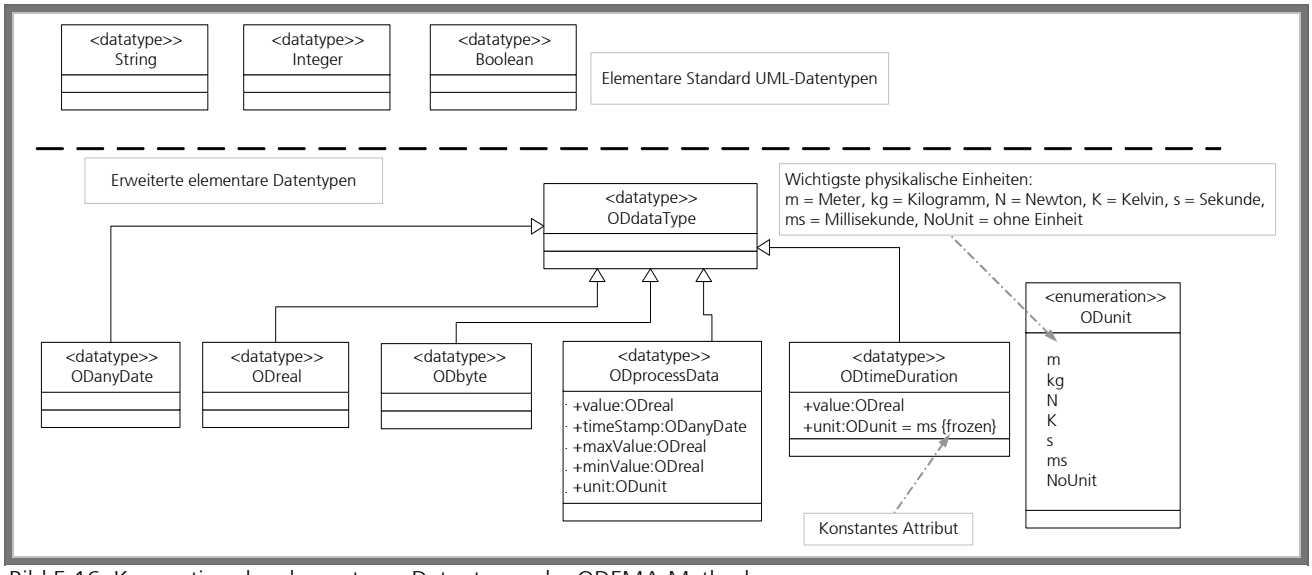

Bild 5.16: Konzeption der elementaren Datentypen der ODEMA-Methode

5.6 Zusammenführung des Beschreibungsmittels mit den Spezifikationsdokumenten des Vorgehensmodells der Methode

Nach der Herleitung der UML-Diagrammtyp-Rollen für das Beschreibungsmittel der ODEMA-Methode muss abschließend die Abbildung auf die bereits eingeführten Spezifikationsdokumente des Vorgehensmodells vorgenommen werden (siehe Kapitel 4.2.1). Hierdurch werden die Konzeption des ODEMA-Vorgehensmodells und die des ODEMA-Beschreibungsmittels zusammengeführt und abgeschlossen.

Zunächst sind die in diesem Kapitel hergeleiteten UML-Diagrammtyp-Rollen in eine geschlossene und flache Darstellung, also ohne sichtbare Ebenen-Grenzen zu bringen. Die UML bietet für das so ge-

nannte *Model-Management* von UML-Diagrammtypen die bereits eingeführten UML-Beschreibungselemente *UML-Subsystem*, *UML-Paket* und *UML-Modell-Pakete*. Während das UML-Subsystem ein physisches System, z.B. die Steuerung, in dekomponierten Teilen mit einem eigenen Verhalten beschreibt, enthält ein UML-Modell-Paket UML-Beschreibungselemente aus einer bestimmten Sicht auf die Steuerung. Ein UML-Paket ist die allgemeinste Form der anwendungsspezifischen Gliederung von UML-Beschreibungselementen und –Diagrammtypen.

Das UML-Subsystem *Steuerung* enthält das gesamte zu beschreibende System (siehe Bild 5.2). Aus dem Ansatz der Sichten-Schichten-Darstellung ergibt sich, dass keine unmittelbar anschließende Zerlegung in weitere UML-Subsysteme erfolgen kann, sondern sich zunächst eine Zerlegung in Sichten ergibt, die mittels UML-Modell-Pakete dargestellt wird (siehe Bild 5.17).

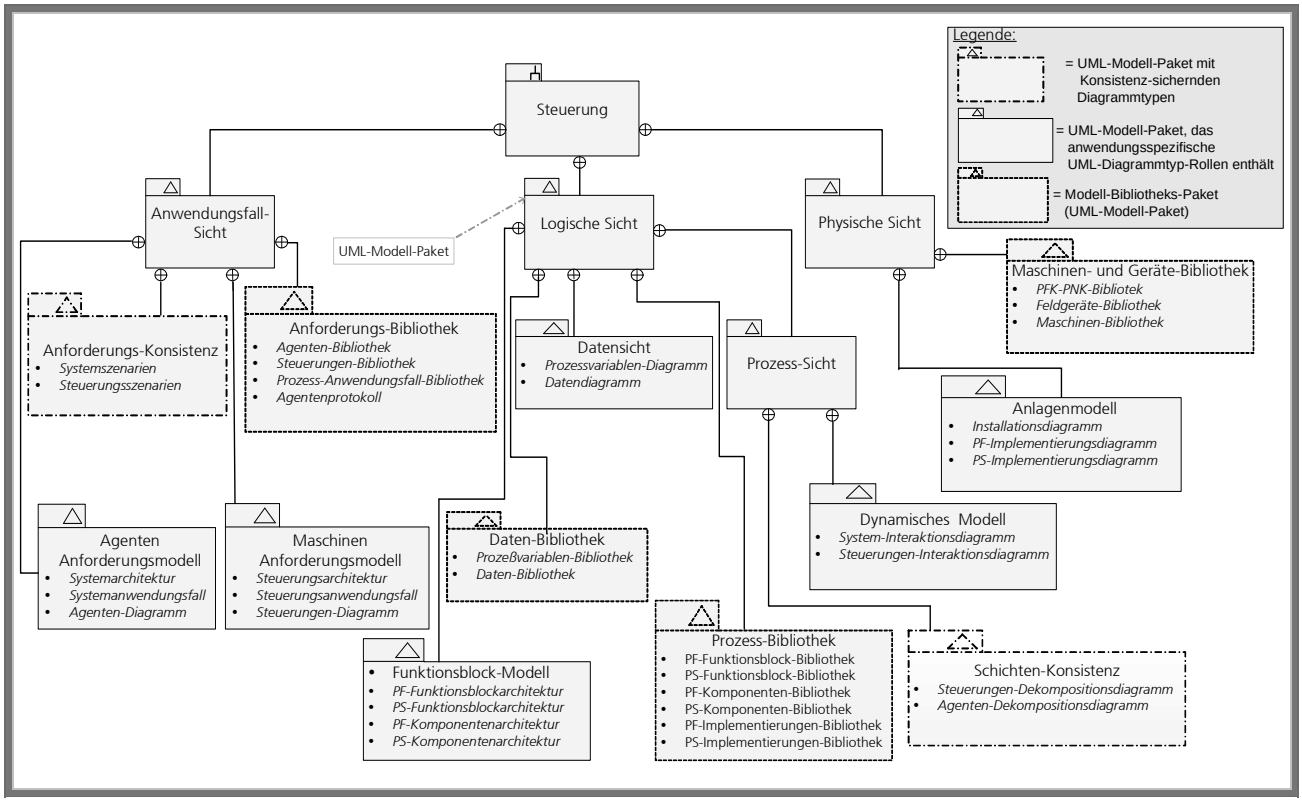

Bild 5.17: Sichten-orientierte und flache Darstellung der UML-Diagrammtyp-Rollen des ODEMA-Beschreibungsmittels (PF = Prozessführung, PS = Prozesssteuerung, PNK = Prozessnahe Komponente, PFK = Prozessferne Komponente)

Aus der abgeleiteten flachen Darstellung der UML-Diagrammtyp-Rollen werden für die Spezifikationsdokumente *Lasten-* und *Pflichtenheft* sowie *Objektorientierter Steuerungsentwurf* ausschließlich anwendungsspezifische UML-Diagrammtyp-Rollen verwendet.

Die UML-Diagrammtyp-Rollen der Anwendungsfallsicht sind definitionsgemäß der Anforderungsdefinition und somit dem Dokument Lastenheft zuzuordnen (siehe Kapitel 4.2.1). Ergänzt werden diese durch das UML-Modell-Paket *Dynamisches Modell* zur Spezifikation der funktionalen Anforderungen des geplanten und des zukünftig zu planenden Produktionssystems bzw. seiner Steuerung (Abschnitt 3 des Lastenheftes). Berücksichtigt sind ebenfalls die unterschiedlichen Ausgangssituationen der PA-Methode durch das *Agenten-* und *Maschinen-Anforderungsmodell (*siehe Bild 5.17 und 5.19*)*.

Die Konzeption des UML-Funktionsblocks, der ausschließlich Schnittstellen spezifiziert und das Verhalten mittels Steuerungskomponenten beschreibt, macht es möglich, die UML-Diagrammtyp-Rolle *PF-*

bzw. *PS-Funktionsblockarchitektur* zur Beschreibung der Schnittstellen bezogen auf das Lastenheft (Abschnitt 4) heranzuziehen. Zur Beschreibung des bestehenden Produktionssystem (Abschnitt 2) bzw. seiner Steuerung wird, da diese als Implementierung vorliegt, das Installationsdiagramm und das PF- bzw. das PS-Implementierungsdiagramm verwendet.

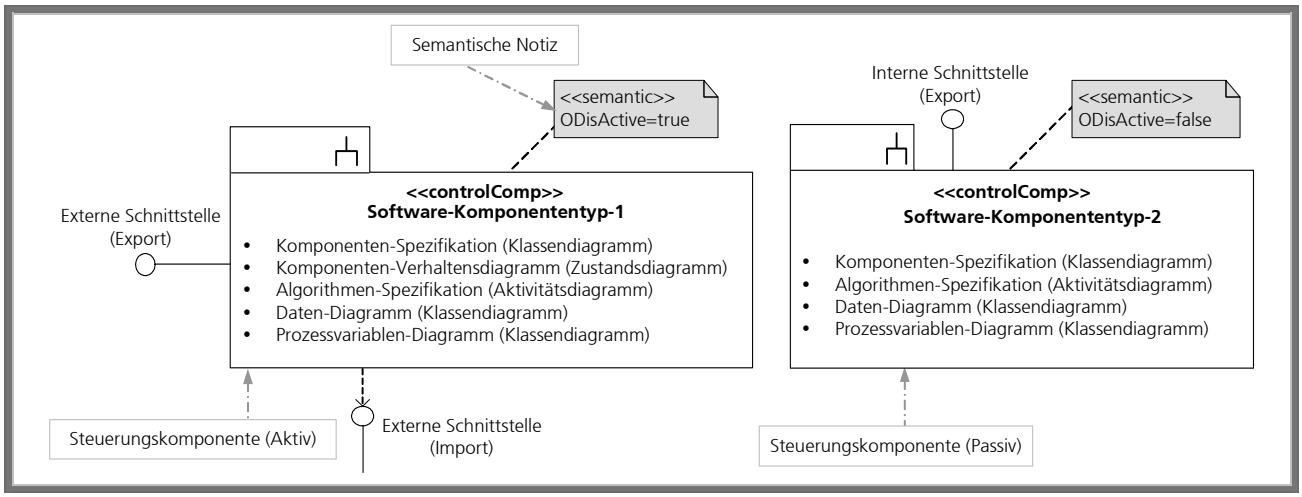

Bild 5.18: Vollständige Darstellung der UML-Diagrammtyp-Rollen zur Spezifikation einer Steuerungskomponente

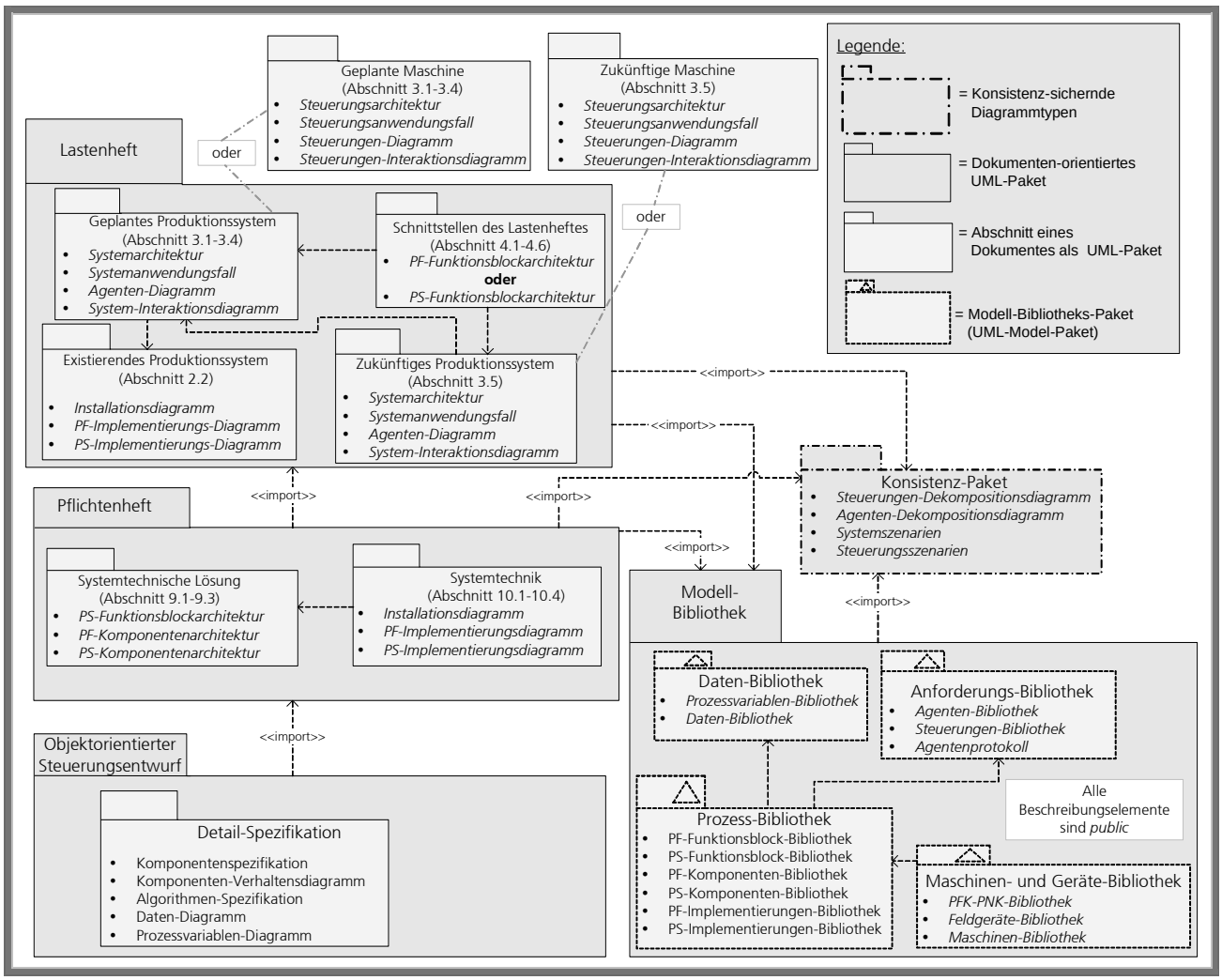

Bild 5.19: Dokumenten-orientierte Darstellung des ODEMA-Beschreibungsmittels (Abschnitte beziehen sich auf VDI/VDE-Richtlinie 3694)

Gemäß der VDI-Richtlinie 3694 schreibt das Pflichtenheft das Lastenheft fort. Das heißt je nach Ausgangssituation der PA-Methode ist die Funktionsblockarchitektur abzuschließen und sind die UML-

Diagrammtyp-Rollen *PF-* und *PS-Komponentenarchitektur* zu spezifizieren. Das Installationsdiagramm und das PF- bzw. das PS-Implementierungsdiagramm sind dann entsprechend weiter zu entwickeln bzw. zu vervollständigen. System- und Software-Dekomposition werden hierdurch vollständig beschrieben, so dass auf dieser Basis entschieden werden kann, welche Software- bzw. Gerätekomponenten zugekauft, wieder zu verwenden oder zu entwickeln sind. Dies ist die Grundlage der Angebotserstellung. Das Pflichtenheft ist somit aus Sicht der Steuerungsspezifikation vollständig beschrieben (siehe Kapitel 2.11 und Bild 5.18 und 5.19).

Die Sichten-orientierte Darstellung der hergeleiteten UML-Diagrammtyp-Rollen stellt weder die Klassen- noch die Methodenschicht dar. Diese dienen ausschließlich der verfeinernden Beschreibung der Steuerungskomponenten und werden zusammen mit denen der Datensicht in Bild 5.18 dargestellt. Sie beschreiben die Umsetzung und Realisierung der funktionalen Anforderungen (*Kausal-* und *Temporalprinzip*). Diese Spezifikationen sind als spezifisches Wissen (*Intellectual Property*) des Entwicklers als Auftragnehmer anzusehen und sind somit Teil des nicht-öffentlichen Dokumentes *Objektorientierter Steuerungsentwurf*.

6 Entwicklung der Methode zur Spezifikation von Steuerungen

In diesem Kapitel wird die Methode ODEMA auf Basis der bereits abgeleiteten UML-Beschreibungstechniken und UML-Diagrammtyp-Rollen zur Spezifikation von Steuerungen entwickelt. Ziel ist es den vollständigen Beschreibungsumfang der verschiedenen UML-Diagrammtyp-Rollen des ODEMA-Beschreibungsmittels und deren Anwendung darzustellen. Zu diesem Zweck gliedert sich diese Entwicklung gemäß der Produkten des Vorgehensmodells *Lastenheft*, *Pflichtenheft*, *Objektorientierter Steuerungsentwurf*. In diesem Kontext werden auch die UML-Diagrammtyp-Rollen *Modell-Bibliothek* und *Konsistenz-sichernder Diagrammtyp* entwickelt, sofern sie nicht trivial sind und in Kapitel 5.4 bereits beschrieben wurden.

6.1 Entwicklung der Methode zur Spezifikation des Lastenheftes

6.1.1 Beschreibung des existierenden Produktionssystems

Bezüglich der Spezifikation von Steuerungen gibt es bedingt durch die Integration der PA-Methode prinzipiell zwei unterschiedliche Ausgangssituationen zu berücksichtigen (siehe Kapitel 4.2.2).
Die Ausgangssituation 2 lässt es zu, dass die Steuerung des bestehenden Produktionssystems in ihrem Zustand bestehen bleibt und einen *Stellvertreter-PA* implementiert, der wiederum über eine beliebige Anzahl von Rollen verfügt. Eine Beschreibung des innere Aufbau des Stellvertreter-PA, z.B. mittels eines Installationsdiagramms, ist nicht notwendig. Der Stellvertreter-PA kann als externer Akteur angesehen werden.
Die Neu-Entwicklung eines Produktionssystem als PA-MAS (Ausgangssituation 1) variiert ausschließlich bezüglich des Umfangs des Produktionssystems. Die Agentifizierung eines bestehenden Produktionssystems im Kontext einer Neuentwicklung ist hierbei eine Variante von Ausgangssituation 2. Nach Abschluss einer Agentifizierung stehen dem Entwickler Beschreibungen bezüglich des physischen Aufbaus der Produktionsagenten und seiner Rollen sowie Produktionsfunktionen zur Verfügung. Bezüglich der Beschreibung des existierenden Produktionssystem wird ausschließlich der physische Aufbau durch die UML-Diagrammtyp-Rolle *Installationsdiagramm* berücksichtigt. Die übrigen Informationen werden in den Kontext der Spezifikation des geplanten Produktionssystems beschrieben (siehe Kapitel 6.1.2).

6.1.1.1 Spezifikation der Ausgangsituation eines agentifizierten Produktionssystems

Gemäß der Konzeption des Installationsdiagramms enthält diese UML-Diagrammtyp-Rolle UML-Geräte- und UML-Kommunikationssystem-Beschreibungen und zudem Beschreibungen der Komponenten-Implementierungen, die als Implementierungen von Steuerungskomponenten verstanden werden. Für UML-Komponenten sind bereits durch die UML entsprechende Stereotypen definiert, die auch in der ODEMA-Methode ihre Anwendung finden. Unter einem *ausführbaren Programm* (Stereotyp *executable*) versteht man eine im Kontext des Laufzeitsystems ausführbaren Software, die zudem als eine identifizierbare und beliebig kopierbare Datei zu handhaben ist. Im Gegensatz hierzu wird die

Programm-Bibliothek (Stereotyp *library*) als ein Stück passiver, implementierter Programmcode verstanden, der aber ebenfalls die Eigenschaften einer Datei besitzt. Als Beispiel hierfür können DLLs und SLLs gelten. Prinzipiell lassen sich UML-Komponenten hierbei als Typ bzw. Klasse und als Instanz beschreiben (siehe Bild 0.4).

Die mechanischen Grenzen innerhalb eines Installationsdiagramms, z.B. durch eine Werkzeugmaschine als Automatisierungs-Subsystem, und die Grenzen eines Steuerungsverbundes von Automatisierungsgeräten und –Subsystemen, wie es Produktionsagenten in der Physischen Sicht darstellen, wurden bisher nicht berücksichtigt noch stellt die UML hierfür Beschreibungstechniken (z.B. Hierarchisierung von UML-Knoten) zur Verfügung [UML_03].

Zur Beschreibung eines agentifizierten Produktionssystems als Ausgangssituation gehört aber die Information der Zugehörigkeit bestimmter Automatisierungsgeräte und Produktionssubsysteme zu einem Produktionsagenten. In Kapitel 4.2.1 wurden die UML-Beschreibungstechniken *Physischer Rahmen* und *Agenten-Rahmen* bereits eingeführt. Zur Spezifikation eines Produktionsagenten bzw. eines Produktionssubsystems im Installationsdiagramm sind zu diesem Zweck zwei Stereotypen des UML-Modell-Paketes zu wählen, da es sich hier um eine Sichten-orientierte Gliederung handelt (Stereotypen *agentFrame* und *physicalFrame*, siehe Tabelle 6.1a). Bild 6.1 zeigt die entsprechende Erweiterung der UML-Pakete *Systemtechnik* und *Existierendes Produktionssystem (*siehe auch Bild 5.19*)*. Das UML-Modell-Paket *Maschinen- und Geräte-Bibliothek* ist ebenfalls zu erweitern, da Produktionsagenten und Produktionssubsysteme wieder zu verwendende Beschreibungselemente darstellen.

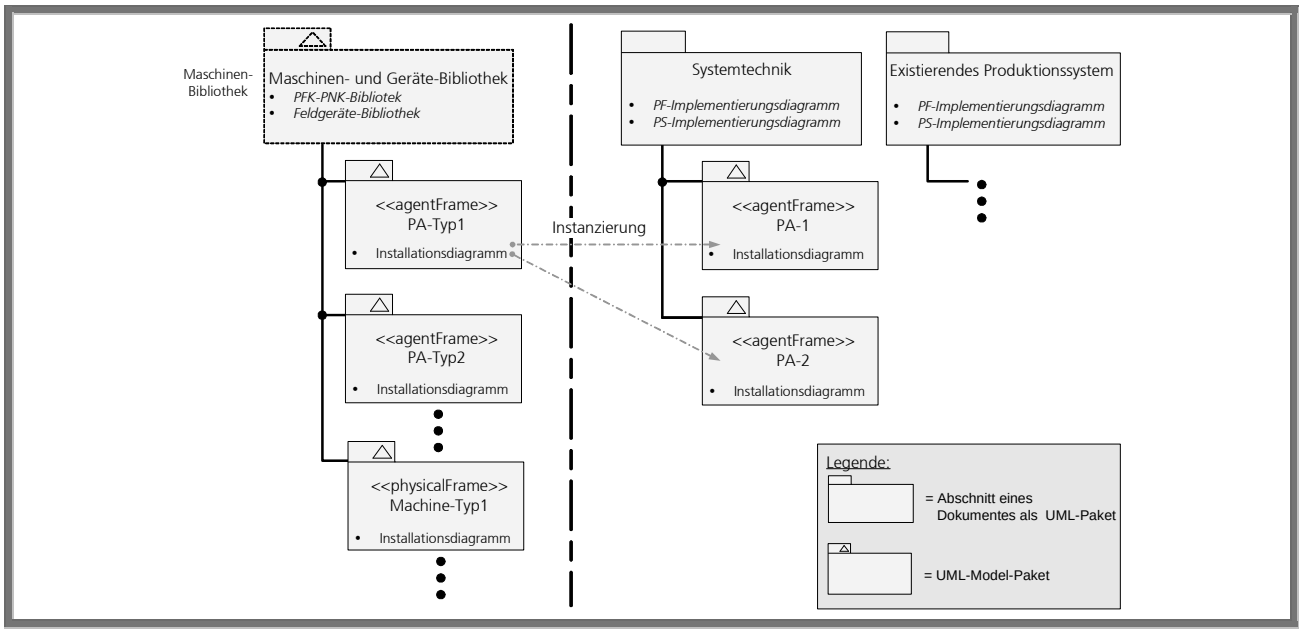

Bild 6.1: Konzept der Wiedervervendung von Produktionsagenten und Produktions-Subsystemen als Rahmen

Im Kontext der Wiederverwendung von UML-Kommunikationssystem-Beschreibungen inklusive der angeschlossenen Feldgeräte wurde bereits die Modellierungstechnik von UML-Knoten-Instanzen als anonyme Instanzen vorgestellt. Um die Wiederverwendung von Agentenrahmen zu ermöglichen wird diese Beschreibungstechnik auch im Kontext der Wiederverwendung von Produktionsagenten angewendet. Hieraus folgt, dass im Fall der Instanzierung eines Produktionsagenten bzw. eines Agenten-

Rahmens ein Installationsdiagramm mit anonymen UML-Knoten-Instanzen spezifische Bezeichner erhält und zugleich, da UML-Modell-Pakete nicht instanzierbar sind, erhält auch der Agenten-Rahmen im Kontext der UML-Pakete *Systemtechnik* oder *Existierendes Produktionssystem* einen neuen Bezeichner. Der physische Rahmen basiert auf der gleichen UML-Beschreibungstechnik wie der Agenten-Rahmen (siehe Bilder 6.1 und 6.2).

Instanzen von UML-Komponenten lassen sich im Kontext der Wiederverwendung von physischen Rahmen und Agenten-Rahmen ebenfalls als anonyme Instanzen beschreiben. Wird ein Produktionssubsystem oder ein Produktionsagent als programmierbar verstanden, sind die Installationsdiagramme der Maschinen-Bibliothek ohne entsprechende UML-Komponenten zu beschreiben. Sie sind dann das Ergebnis der Projekt-bezogenen Spezifikation also der Instanzierung (siehe Bild 6.2).

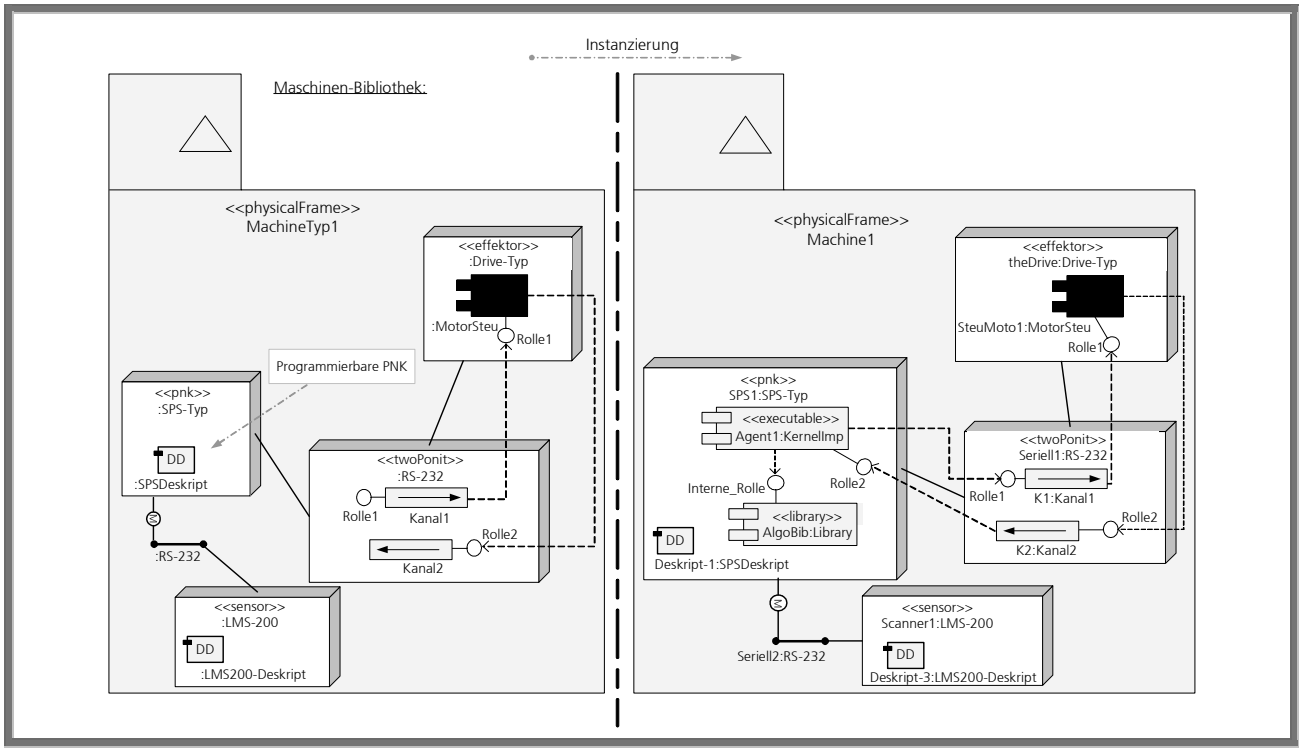

Bild 6.2: Exemplarisches Installationsdiagramm im Kontext eines physischen Rahmens

Stereotyp	Basis-Klasse	Eigenschaftswerte	Einschränkungen (*formal*/informal)
Physischer Rahmen <<physicalFrame>>	Model	-	enthält nur ein Installationsdiagramm
Agenten-Rahmen <<agentFrame>>	Model	ODagentType	enthält nur ein Installationsdiagramm
Ausführbares Programm <<executable>>	Component	-	Implementierung einer aktiven Steuerungskomponente
	ComponentInstance		
Prgramm-Bibliothek <<library>>	Component	-	Implementierung einer passiven Steuerungskomponente
	ComponentInstance		

Tabelle 6.1a: Stereotypen im Kontext der physischen Beschreibung als Abschnitt des UML-Profil der ODEMA-Methode (Es werden keine Elternklasse verwendet)

Eigenschaftswert	Stereotyp	Typ	Multiplizität
ODagentType	Agenten-Rahmen	UML::Datatypes::String	1

Tabelle 6.1b: Eigenschaftswerte im Kontext der Tabelle 6.1a

6.1.2 Beschreibung des geplanten Produktionssystems

Das zu planende Produktionssystem und somit auch dieser Aspekt der zu spezifizierenden Steuerung sind der Kern des Lastenheftes. Ausgehend von der Konzeption der ODEMA-Methode kann vom allgemeinen Fall der Planung und Spezifikation eines PA-MAS der triviale Fall der Spezifikation einer Maschinensteuerung abgeleitet werden (siehe Kapitel 6.1.1).
Das UML-Paket *Geplantes Produktionssystem* (siehe Bild 5.19) berücksichtigt somit zwei Varinaten, deren Spezifikation in den Kapiteln 6.1.2.1 und 6.1.2.2 dargestellt werden. Um den Übergang zur PF- bzw. PS-Funktionsblockarchitektur als Beschreibung der Schnittstellen der Steuerung darzustellen, ist zusätzlich die Beschreibung der Konsistenz-sichernden Diagrammtypen notwendig.

6.1.2.1 Spezifikation der Steuerung eines geplanten PA-MAS

Der System-Anwendungsfall beschreibt einen Anwendungsfall als Interaktion aller an ihm beteiligten externen und internen Akteure der Prozessführungsebene. Bereits bestehende Produktionsagenten werden nicht als externe Akteure beschrieben, sondern - genauso wie die noch zu entwickelnden Produktionsagenten - mittels der eingeführten Stereotypen (*mobile* und *station*) spezifiziert. Dies ist notwendig, da der externe Akteur der UML lediglich die Rolle eines Akteurs ohne seinen Typ spezifiziert und Produktionsagenten aber allgemein mittels Rolle und Typ beschrieben werden (siehe Kapitel 2.9.3).
Die UML-Diagrammtyp-Rolle *Systemarchitektur* wurde zur Beschreibung der Abhängigkeit von System-Anwendungsfällen untereinander eingeführt (siehe Kapitel 5.5.2). Hierbei sind ausschließlich Erweiterungsabhängigkeiten zu berücksichtigen (siehe Bild 6.3). Mit dieser Spezifikation verbunden ist die Angabe der Nummer eines Erweiterungspunktes (*Extension Point*) und die Bedingung, unter welcher der erweiternde Anwendungsfall eintritt. Die Änderung dieser Bedingung ist ein Ereignis des Systems, also des PA-MAS. Hieraus wiederum folgt das jede dieser Bedingungen einem Produktionsagenten zu zu ordnen ist. Im Kontext der System-Anwendungsfälle heißt dies, dass ein bestimmter Produktionsagententyp in einer bestimmten Rolle diese Bedingung spezifiziert und somit nur dieser auf das entsprechende Ereignis reagieren kann. Da die UML prinzipiell einen Anwendungsfall mittels der Rollen beschreibt und unterscheidet, die ein externer Akteur einnimmt, muss die Erweiterungsabhängigkeit bezüglich der UML-Diagrammtyp-Rolle *Systemarchitektur* als die Spezifikation eines Rollenwechsels eines internen Akteurs aufgefasst werden. Die Spezifikation der Bedingung einer Erweiterungsabhängigkeit wird in diesem Kontext durch den Ausdruck Rolle:*Agententyp.Bedingung* erweitert (siehe Bild 6.3).

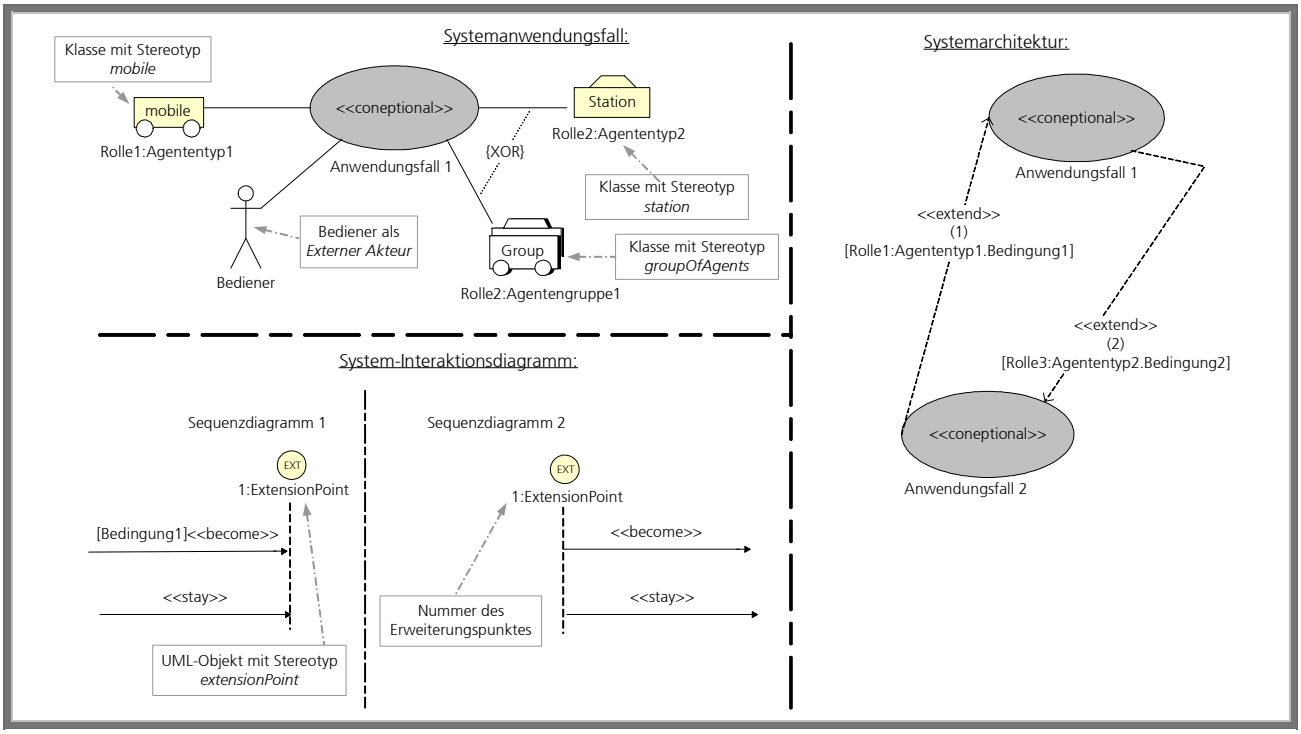

Bild 6.3: Exemplarische Darstellung der Systemanwendungsfälle und der Systemarchitektur

Stereotyp	Basis-Klasse	Eigenschaftswerte	Einschränkungen (*formal*/informal)
Rollen-Verbleib <<stay>>	Stimulus	-	verbindet Instanzen gleichen Typs und gleicher Rolle
Rollenwechsel <<become>>	Stimulus	-	verbindet Instanzen gleichen Typs und unterschiedlicher Rolle
Bedingung <<ODcondition>>	Attribute	-	Bedingung ist vom Datentyp *Boolean*
Vertikale Nachricht <<ODlocal>>	Operation	-	*self.concurrency=#cck_concurrent*
	Message	-	-
	SignalEvent		
Horizontale Nachricht <<com>>	Operation	-	*self.concurrency=#cck_concurrent*
	Message	-	-
	SignalEvent		
	CallState		
Weiche Echtzeitspezifikation <<softTime>>	Comment	-	-
Erweiterungspunkt <<extensionPoint>>	ExtensionPoint	-	-
	Object		
Einbindungspunkt <<includePoint>>	Class	-	-
	Object		
Rollen-Vererbung <<ODrole>>	Generalization	-	Es werden ausschließlich horizontale Nachrichten vererbt.
Typen-Veerbung <<ODtype>>	Generalization	-	Es werden ausschließlich vertikale Nachrichten atomare Aktionen und nicht-unterbrechbare Aktivitäten vererbt.

Tabelle 6.2: Abschnitt des UML-Profils des ODEMA-Beschreibungsmittels bezüglich des geplanten PA-MAS (Es werden keine Elternklasse verwendet)

Die Unterscheidung bezüglich der Erfüllung einer Bedingung im Verlauf eines System-Anwendungsfalls wird durch die Beschreibung der Instanzen eines System-Anwendungsfalls spezifi-

ziert, welche gemäß der Konzeption des ODEMA-Beschreibungsmittels durch die System- bzw. Steuerungen-Interaktionsdiagramme als Szenarien spezifiziert werden. Der Erweiterungspunkt muss somit explizit im System- und Steuerungen-Interaktionsdiagramm beschrieben werden. Die stereotypisierte Klasse *Erweiterungspunkt* (Stereotyp *extensionPoint*) wird zu diesem Zweck eingeführt. Im Zusammenhang mit Sequenzdiagrammen wird ein Zustands- oder Rollenwechsel mittels der Nachricht oder des Stimulus *Rollenwechsel* (Stereotyp *become*) beschrieben [BOOCH_99]. Die Instanz eines internen oder externen Akteurs sendet somit einen bedingten Stimulus zu der Instanz eines Erweiterungspunktes. In einem oder in mehreren System-Interaktionsdiagrammen des erweiternden System-Anwendungsfalls ist die identische Instanz des Erweiterungspunktes beschrieben und leitet den Rollenwechsel auf die entsprechende Instanz des externen oder internen Akteurs (mit veränderter Rolle) weiter. In Ausnahmefällen kann es aus Sicht des Entwicklers wünschenswert sein, eine identische Instanz – also in der identischen Rolle - in zwei System-Interaktionsdiagrammen zu beschreiben. Hierfür wird der *Rollen-Verbleib* (Stereotyp *stay*) eingeführt, der nur im System- bzw. Steuerungen-Interaktionsdiagramm angewendet wird und über keine Entsprechung bezüglich der UML-Diagrammtyp-Rollen System- sowie Steuerungsarchitektur verfügt. Zusammenfassend werden die eingeführten Stereotypen in Tabelle 6.2 dargestellt.

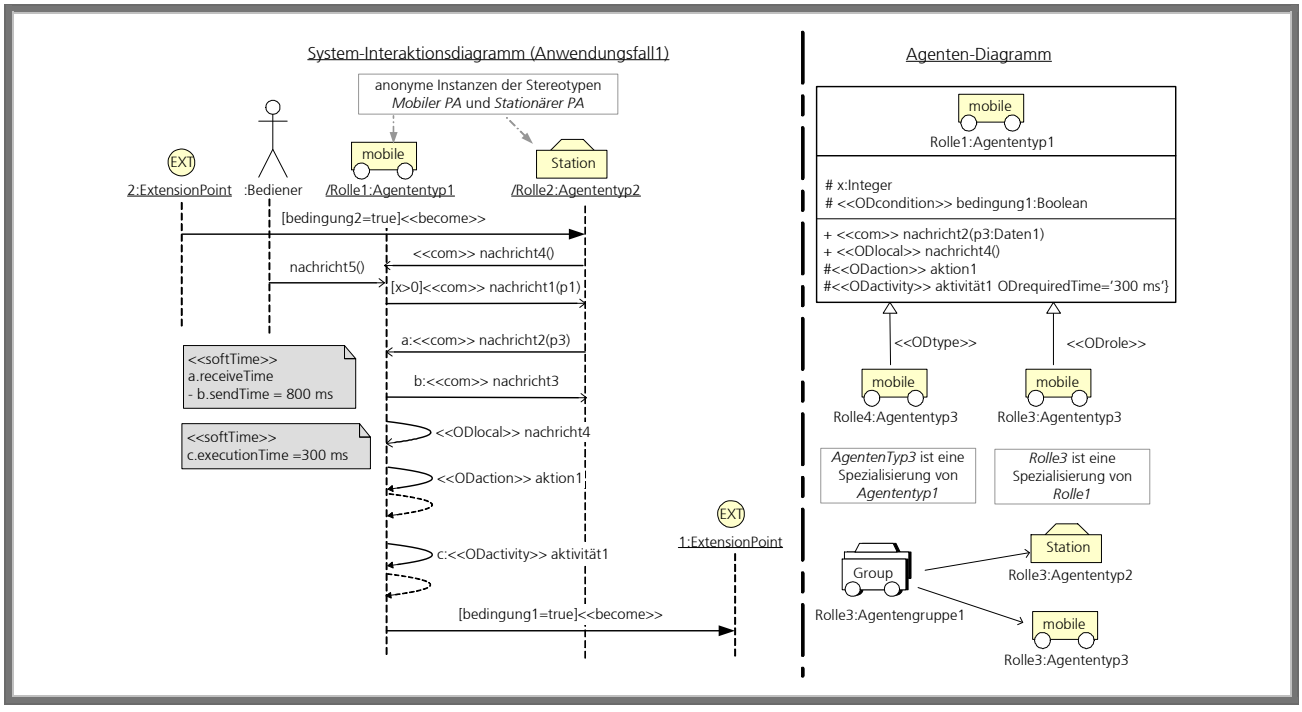

Bild 6.4: Exemplarische Darstellung des System-Interaktionsdiagramms und des Agenten-Diagramms

Die exemplarische Darstellung des System-Interaktionsdiagramms (siehe Bild 6.4) zeigte eine Instanz des System-Anwendungsfall *Anwenungsfall1* (siehe Bild 6.3). Zwei anonyme Instanzen der internen Akteure interagieren mittels Nachrichten. Ebenso sind Prozeduraufrufen beschrieben, die durch die bereits eingeführten Stereotypen *Atomare Aktion* (Stereotyp *ODaction*) und *Nicht-unterbrechbare Aktivität* (Stereotyp *ODactivity*) unterschieden werden (siehe Tabelle 5.13).

Nachrichten werden unterteilt in h*orizontale* und *vertikale Nachrichten*. Horizontale Nachrichten (Stereotyp *com*) werden innerhalb einer Ebene und vertikale Nachrichten werden zwischen Steuerungs-

prozessen verschiedener Ebenen gesendet und empfangen. Die Einführung dieser Unterscheidung ist notwendig, da nur die horizontalen Nachrichten zur Spezifikation der Rollen innerhalb einer Ebene beitragen. Die vertikalen Nachrichten (Stereotyp *ODlocal*) werden nach der Verschiebung der Transparenz-Grenze auf die internen Akteure in der nächst tieferen Ebene abgebildet. Atomare Aktionen und Nicht-unterbrechbare Aktivitäten beziehen sich stets auf das Innere des Steuerungsprozesses *Mobiler PA* oder *Stationärer PA*. Sie sind somit als Methoden bzw. Operationsaufrufe zu verstehen, die innerhalb des aus dem PA-Kopf abgeleiteten UML-Funktionsblock beschrieben werden.

Der interne Akteur *Agenten-Gruppe* kann als Instanz im Kontext eines System-Interaktionsdiagramms ebenso über die dargestellte UML-Beschreibungstechnik verfügen. Die Semantik unterscheidet sich aber von der der internen Akteure *Stationärer* bzw. *Mobiler PA*. Das Senden bzw. Empfangen einer Nachricht bezieht sich auf jeweils nur eine Instanz eines mobilen bzw. stationären PA, der zu der spezifizierten Agenten-Gruppe gehört. Das Gleiche gilt für den Aufruf atomaren Aktion oder einer nicht-unterbrechbaren Aktivität. Soll beschrieben werden, das aus einer Agenten-Gruppe heraus die identische Nachricht von mehreren Instanzen gesendet wird, so kann die UML-Beschreibungstechnik *Iteration* verwendet werden *(*[Anzahl Iterationen] nachricht()*, siehe [UML_03]).

Wesentliches Kriterium für die Auswahl des Sequenzdiagramms zur Beschreibung der UML-Diagrammtyp-Rolle *System-* bzw. *Steuerungen-Interaktionsdiagramm* war die Berücksichtigung der Zeitachse. Im Kontext der Beschreibung eines PA-MAS sind weiche Echtzeitanforderungen möglich. Sie können gemäß dem *Echtzeitmodell* (siehe Kapitel 3.1.7) als Maximalwert oder im Fall weicher Echtzeitanforderungen auch als Mittelwert formuliert werden. Zur Spezifikation von Echtzeitanforderungen (*required QoS*) stellt die UML die Beschreibungstechnik *Marken* zur Verfügung. Marken (z.B. a:, b: und c: in Bild 6.4) bezeichnen den Zeitpunkt einer Nachricht oder eines Stimulus im Sequenzdiagramm. Mit Hilfe der *Zeitfunktionen sendTime* (Zeitpunkt des Sendens), *receiveTime* (Zeitpunkt des Empfangens) und *executionTime* (Ausführungszeit eines Operationsaufrufs) lassen sich *zeitbezogene Ausdrücke* (*TimeExpression*) beschreiben, die durch die hier einzuführende weiche Echtzeitspezifikation (Stereotyp *softTime*) im Sequenzdiagramm bekannt gemacht werden können. Im Fall des System-Interaktionsdiagramm werden ausschließlich Mittelwerte formuliert (z.B. *x*.receiveTime - *y*.sendTime = *z* ms oder s).

Das *Agenten-Diagramm* beschreibt sie statischen Beziehungen der stationären PA, der mobilen PA und der Agenten-Gruppen, wobei mittels einer *gerichteten Assoziation* (*Association*) die Teilnehmer einer Agentengruppe spezifiziert werden. Voraussetzung hierfür ist die identische Rolle bezüglich der Agenten-Gruppe. Die Vererbung von Rollen bzw. Agententypen wird durch eine Vererbungsbeziehung beschrieben (siehe Bild 6.4). Hierbei bedeutet die *Rollen-Vererbung* (Stereotyp *ODrole*), dass die Rolle des abgeleiteten, internen Akteurs ausschließlich die horizontalen Nachrichten erbt, d.h. nur die Nachrichten, die zu Spezifikation der Rolle in seiner Ebene beitragen. Die *Typen-Vererbung* (Stereotyp *ODtype*) ist als komplementär anzusehen. Durch sie werden alle geschützten (*protected*) Attribute und die vertikalen Nachrichten, atomaren Aktionen und nicht-unterbrechbaren Aktivitäten vererbt.

6.1.2.2 Spezifikation der geplanten Maschinensteuerung

Der triviale Fall der Beschreibung eines geplanten Produktionssystems ist die Spezifikation einer Maschinensteuerung. Die Beschreibung funktionalen Anforderungen wird in diesem Fall mittels Steuerungsanwendungsfällen und den dazugehörigen PS-Steuerungsprozessen bzw. Steuerungsgruppen durchgeführt (siehe Bild 6.5).

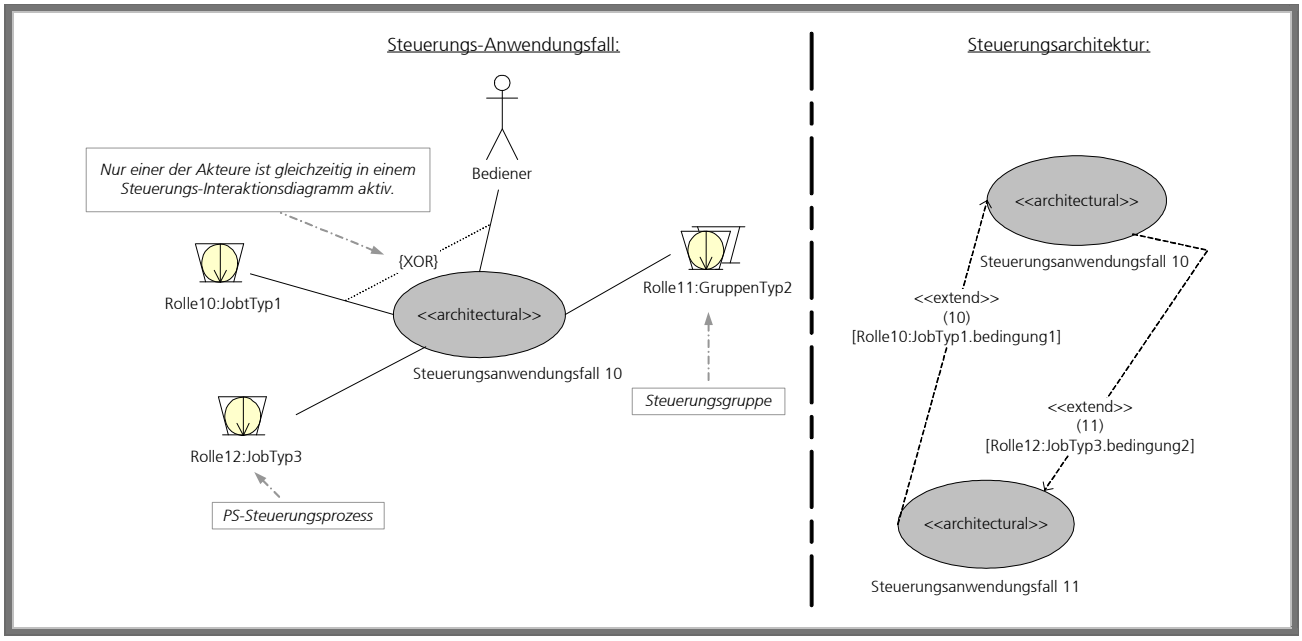

Bild 6.5: Exemplarische Darstellung des Steuerungsanwendungsfalls und der Steuerungsarchitektur

Die Durchgängigkeit der bezüglich der ODEMA-Methode verwendeten Beschreibungstechniken sorgt dafür, dass auch das Steuerungen-Interaktionsdiagramm und das Steuerungen-Diagramm in der gleichen Weise angewendet werden können, wie die entsprechenden UML-Diagrammtyp-Rollen der Prozessführungsebene (*Anwendbarkeit*). Lediglich eine Entsprechung zum Agenten-Protokoll steht nicht zu Verfügung.

Ergänzend ist für das Steuerungen-Interaktionsdiagramm zu berücksichtigen, dass im Unterschied zum System-Interaktionsdiagramm harte Echtzeitspezifikationen und zeit-gesteuerte Steuerungsprozesse beschreibbar sein müssen (siehe Kapitel 3.1.2). Zur Spezifikation harter Echtzeitspezifikationen wird eine entsprechende stereotypisierte Notiz (Stereotyp *hardTime*) eingeführt, die prinzipiell die gleichen zeitbezogenen Ausdrücke enthalten, wie bereits für das System-Interaktionsdiagramm dargestellt wurde (siehe Kapitel 6.1.2.1). Erweiternd wird die Zeitfunktion *loopTime* eingeführt, die ausschließlich im Kontext des Schleifenaufrufs (Stereotyp *ODloop*) eingesetzt wird. Der Schleifenaufruf wird ausschließlich in den erweiterten Sequenzdiagrammen verwendet und markiert auf der Lebenslinie einer Instanz (*Object Lifeline*) einen Prozeduraufruf, der als Beginn und Ende einer Iteration zu verstehen ist. Alle Operationsaufrufe und gesendeten Nachrichten innerhalb zweier Schleifenaufrufe werden wiederholt ausgeführt. Gesteuert wird diese Iteration durch die mittels einer Marke zugewiesene Zeitfunktion *loopTime*, die die Zykluszeit des in diesem Bereich zeitgesteuerten Steuerungsprozess spezifiziert.

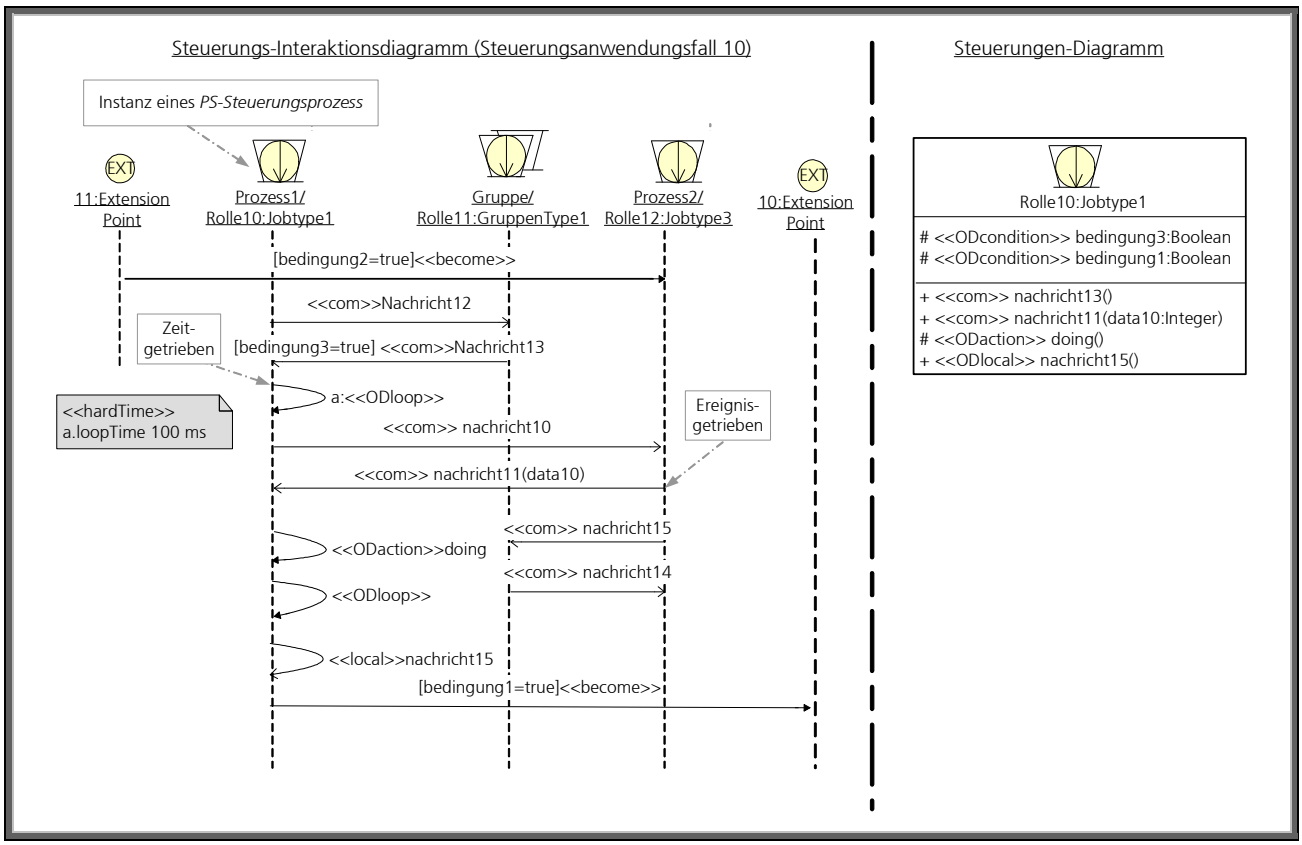

Bild 6.6: Exemplarische Darstellung des Steuerungen-Interaktionsdiagramms und des Steuerungen-Diagramms

Stereotyp	**Basis-Klasse**	**Eigenschaftswerte**	**Einschränkungen (*formal*/informal)**
Harte Echtzeitspezifikation <<hardTime>>	Comment	-	-
Schleifenaufruf <<ODloop>>	Procedure	_	Beginn oder Ende einer Iteration

Tabelle 6.3: Abschnitt des UML-Profils der ODEMA-Methode bezüglich der Steuerungen-Interaktionsdiagarmme (Es werden keine Elternklasse verwendet)

6.1.2.3 Entwicklung der Agenten-Protokoll-Bibliothek

Während die im Kontext der Beschreibung des geplanten Systems zu verwendenden Modell-Bibliotheken *Agenten-* und *Steuerungen-Bibliothek* als trivial anzusehen sind, da sie lediglich unverknüpfte Klassen beschreiben, ist das *Agenten-Protokoll* eine eigenständige UML-Diagrammtyp-Rolle. Kennzeichnend für diese Rolle des Aktivitätsdiagramms ist neben der Interpretation der *Swimlanes* als Rollen die ausschließliche Verwendung von horizontalen Nachrichten als Aufrufe (*Call state*) und Bedingungen (*guard condition*) zur Beschreibung der logischen Verzweigung. Gemäß der Definition der Begriffe *Rolle* und *Typ* im Kontext der ODEMA-Methode werden bei Anwendung eines Agenten-Protokolls die horizontalen Nachrichten Teil der Spezifikation der Rolle und die Bedingungen Tei des Agententyps. Die System-Interaktionsdiagramme sind somit als Beschreibung der durch die Bedingungen hervorgerufenen Varianten des Agenten-Protokolls zu verstehen.

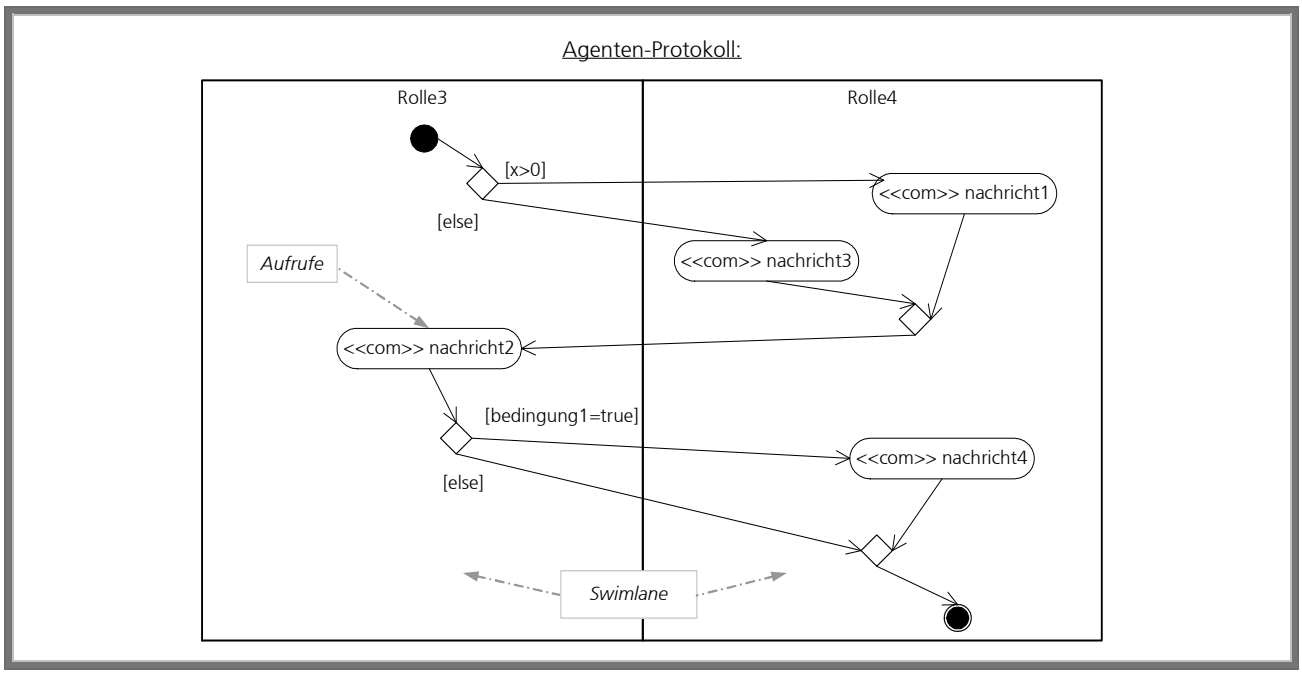

Bild 6.7: Exemplarische Darstellung eines Aktivitätsdiagramm als UML-Diagrammtyp-Rolle *Agenten-Protokoll*

6.1.3 Beschreibung der Schnittstellen des Produktionssystems

Die Konzeption des ODEMA-Beschreibungsmittel sieht je nach Ausgangssituation die UML-Diagrammtyp-Rolle *PF-* oder *PS-Funktionsblockarchitektur* zur Beschreibung der Schnittstellen im Lastenheft vor (Kapitel 5.6). Da diese UML-Diagrammtyp-Rollen in der Subsystemschicht definiert sind, muss zunächst die Dekomposition der internen Akteure in UML-Funktionsblöcke in Form des Konsistenz-sichernden Diagrammtyp *Agenten-* bzw. *Steuerungen-Dekompositionsdiagramm* beschriebnen werden. Zusätzlich ist zu berücksichtigen, das auch eine Dekomposition des System- auf das Steuerungen-Interaktionsdiagramm möglich ist.

6.1.3.1 Entwicklung des Agenten- und des Steuerungen-Dekompositionsdiagramms

Interne Akteure, dies gilt für mobile und stationäre PA sowie für PS-Steuerungsprozesse, werden mittels Rolle und Typen spezifiziert. Diese werden bezüglich der System-Dekomposition auf UML-Funktionsblöcke abgebildet, wobei jeder Typ auf genau einen UML-Funktionsblock abgebildet wird (siehe Kapitel 5.5.1). Rolle und Typ sind von daher auch getrennt voneinander zu dekomponieren. Da sich gemäß Konzeption die PF- und PS-Funktionsblöcke lediglich bezüglich ihres Spezifikationsumfanges unterscheiden, genügt es an dieser Stelle, ausschließlich das Agenten-Dekompositionsdiagramm vorzustellen. Die Übertragung der dargestellten Beschreibungstechniken auf das Steuerungen-Dekompositionsdiagramm ist als trivial anzusehen.

Da die UML die System-Dekomposition von Rollen bzw. Typen nicht unterstützt, wird die *Dekomposition* (Abhängigkeit mit dem Stereotyp *decompose*) in das ODEMA-Beschreibungsmittel eingeführt (siehe Tabelle 6.4a). Sie beschreibt die Dekomposition eines internen Akteurs in Schnittstellen-Klassen (Stereotyp *interface*) und eine Verhaltensklasse (Stereotyp *ODbehavior*), wobei die Schnittstellen-Klasse kein neu einzuführender Stereotyp ist, sondern zum UML-Standard gehört (siehe Bild 6.8).

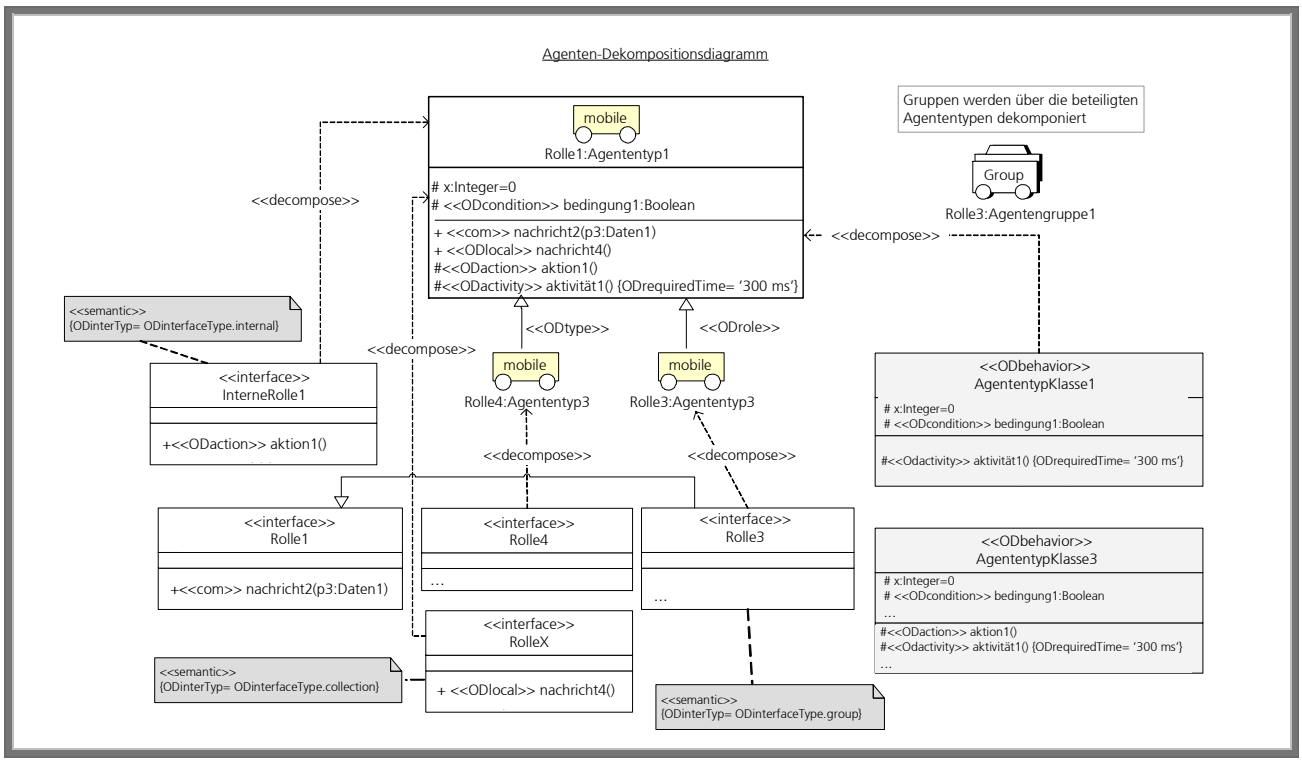

Bild 6.8: Exemplarische Darstellung des Agenten-Dekompositionsdiagramms

Gemäß der eingeführten Semantik der Rollen-Vererbung werden zunächst ausschließlich die horizontale Nachrichten einer Rolle zugeordnet und somit auch bezüglich der entsprechenden Schnittstellen-Klasse, *Rollen-Schnittstelle* genannt, dekomponiert. Rollen-Schnittstellen sind dadurch gekennzeichnet, dass sie als Schnittstellen-Klassen ohne Spezifikation des Eigenschaftswertes *ODinterType* dargestellt werden, der zur Unterscheidung der verschiedenen Schnittstellen-Klassen eingeführt wird (siehe Tabelle 6.4b und Bild 6.8).

Äquivalent gilt für die Dekomposition des Typs, dass ausschließlich Attribute, atomare Aktionen und nicht-unterbrechbare Aktivitäten berücksichtigt werden. Sie werden auf die in diesem Kontext einzuführende *Verhaltensklasse* dekomponiert (siehe Kapitel 6.4a). Da die Realisierung von UML-Funktionsblöcken durch Steuerungskomponenten und deren gekapselte UML-Diagrammtyp-Rollen beschrieben ist, ist die Verhaltensklasse Teil der UML-Diagrammtyp-Rolle *Komponenten-Spezifikation* (Klassendiagramm).

Vertikale Nachrichten beschreiben die Kommunikation mit einem UML-Funktionsblock aus einer tiefer gelegenen Ebene. Aus z.B. einem System-Interaktionsdiagramm geht aber nicht hervor, ob eine Nachricht diesen Funktionsblock bzw. Steuerungsprozess gesendet oder von diesem empfangen wird. Dies muss vom Entwickler festgelegt werden. Beschrieben wird dies im Agenten-Dekompositionsdiagramm mit Unterstützung des Eigenschaftswertes *ODinterType*. Ausnahme ist hierbei, wenn bereits ein Steuerungen-Interaktionsdiagramm beschrieben ist, aus dem die Richtung der vertikalen Nachrichten hervorgeht. Der *Wert collection* des Eigenschaftswertes *ODinterType* zeigt an, dass eine vertikale Nachricht gesendet wird, aber die Rolle des Empfängers noch nicht bekannt ist. Die Schnittstellen-Klasse wird in diesem Fall als *Sammel-Schnittstelle* bezeichnet und muss im Verlauf der Spezifikation des Pflichtenheftes weiter ausgearbeitet werden. Eine Sammelschnittstelle wird also noch durch eine wei-

tere Stufe dekomponiert. Wird eine vertikale Nachricht als empfangene Nachricht identifiziert, wird sie Teil einer Rollen-Schnittstellen, die von dem empfangenen UML-Funktionsblock exportiert wird. Bezüglich des System- und des Steuerungen-Interaktionsdiagramms lässt sich diese Dekomposition auch mittels der Funktionen *sendPoint* und *receivePoint* beschreiben (siehe Kapitel 6.1.3.3).

Eine weitere Ergänzung ist bezüglich des Agenten- bzw. Steuerungen-Dekompositionsdiagramm im Kontext der Beschreibung des Abschnitts *Systemtechnische Lösung* des Pflichtenheftes zu machen. Atomare Aktionen und nicht-unterbrechbare Aktivitäten können von passiven Steuerungskomponenten exportiert werden. Daraus folgt, dass diese stereotypisierten Operationen nicht nur mittels einer Verhaltensklasse, sondern auch durch *Interne Schnittstellen* (*ODinterType = internal*) dekomponiert werden können (siehe Bild 6.8).

Rollen, die sich auf Gruppen beziehen, werden bereits durch das Agenten- bzw. Steuerungen-Diagramm auf die internen Akteure *Mobiler* und *Stationärer PA* bzw. *PS-Steuerungsprozess* abgebildet. Somit werden sie ebenso auf Schnittstellen-Klassen dekomponiert. Trotzdem werden diese Schnittstellen-Klassen durch den Eigenschaftswert *ODinterType* gekennzeichnet. Der Wert *group* wird aber erst im Kontext der Verteilung der Steuerungskomponenten berücksichtigt und kennzeichnet die *Gruppen-Schnittstelle*. Hierbei ist zu beachten, dass die Eigenschaft *Gruppen-Schnittstelle* sich ausschließlich auf die horizontalen Nachrichten einer Schnittstellen-Klasse bezieht, da die Agenten- bzw. Steuerungen-Gruppen im Kontext der horizontalen Interaktion von Steuerungsprozessen spezifiziert werden.

Stereotyp	**Basis-Klasse**	**Eigenschaftswerte**	**Einschränkungen (*formal*/informal)**
Verhaltensklasse <<ODbehavior>>	Class	-	• *self.isActive = true* • *self.isLeaf = true* • Klasse ist ein *Singelton* • Verfügt über ein Zustandsdiagramm (*Komponenten-Verhaltensdiagramm*)
Dekomposition <<decompose>>	Dependency	-	-

Tabelle 6.4a: Abschnitt des UML-Profils bezüglich des Agenten- und Steuerungen-Dekompositionsdiagramms (Es werden keine Elternklasse verwendet)

Eigenschaftswert	**Stereotyp**	**Typ**	**Multiplizität**
ODinterType	interface*	<<enumeration>> ODinterface group collection internal	1
ODrequiredTime *Maximale Ausführungszeit*	ODactvity (siehe Tabelle 5.13)	UML::Datatypes::String	1

Tabelle 6.4b: Eigenschaftswerte bezüglich des Lastenheftes (*= ist durch den UML-Standard bereits definiert)

6.1.3.2 Spezifikation der Funktionsblockarchitektur

Die VDI/VDE-Richtlinie 3694 gliedert die Beschreibung der Schnittstellen einer Steuerung mittels der Aspekte *Mensch*, *Rechner (Automatisierungsgerät)*, *Steuerungsprozess* und *Technischer Prozess*.

Durch die Definition der *Bedienersteuerung* als Steuerungsfunktion muss die Schnittstelle zwischen Mensch und Automatisierungsgerät nicht berücksichtigt werden. Diese wird als Schnittstelle zwischen Mensch und Steuerungsprozess beschrieben. Die Beschreibung der Schnittstelle zwischen Automatisierungsgeräten wird durch die UML-Diagrammtyp-Rolle *Installationsdiagramm* dargestellt und gehört zum Abschnitt *Systemtechnik* des Pflichtenheftes. Die UML-Geräte-Beschreibung, ebenso Teil des Installationsdiagramms, enthält durch die Spezifikation der Sensoren und Aktoren bereits die Schnittstelle zum technischen Prozess.

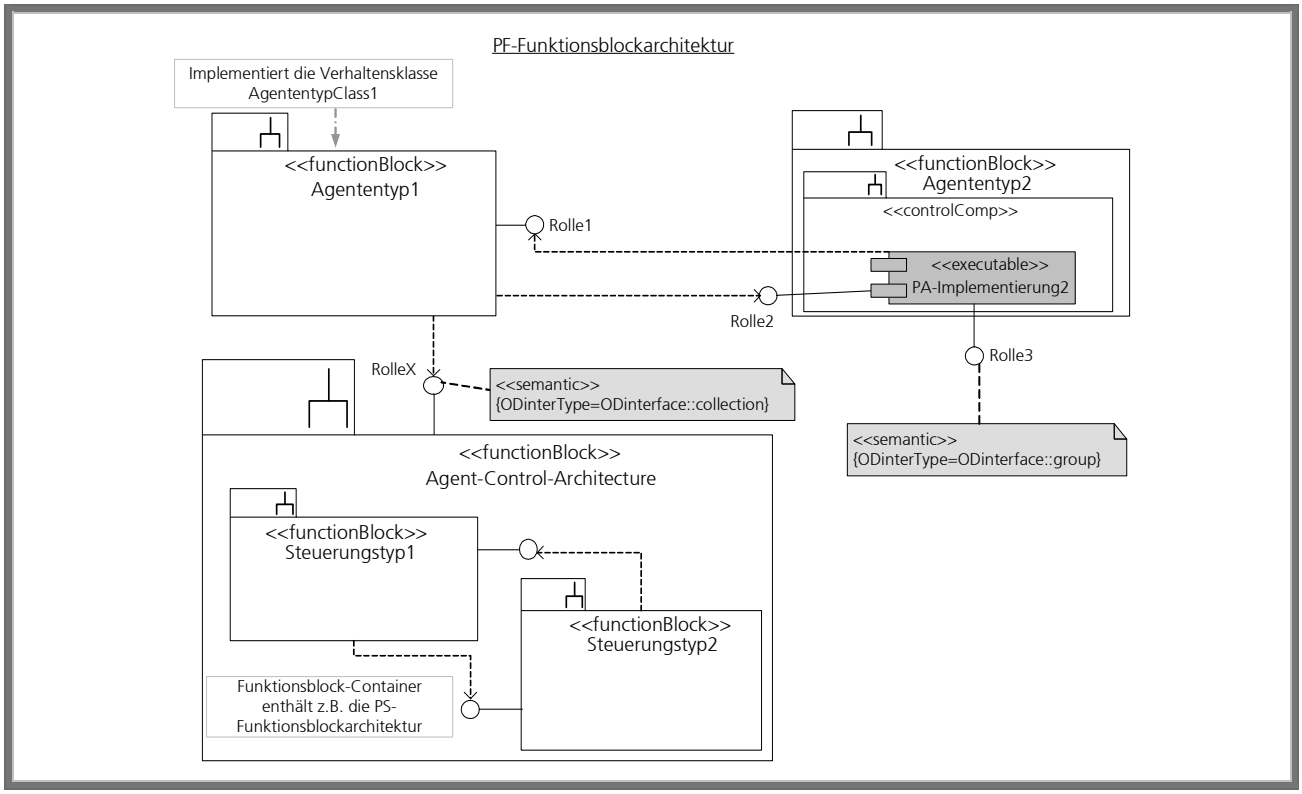

Bild 6.9: Exemplarische Darstellung der PF-Funktionsblockarchitektur

Da die Schnittstelle zwischen Steuerungsprozessen und Automatisierungsgeräten erst im Kontext der Beschreibung der UML-Diagrammtyp-Rolle *Komponenten-Spezifikation* dargestellt werden kann, ist in Ergänzung zur Schnittstelle zwischen Mensch und Steuerungsprozess im Zusammenhang mit der Erstellung des Lastenheftes ausschließlich die Schnittstelle zwischen Steuerungsprozessen zu berücksichtigen.

Die aus dem Agenten- bzw. Steuerungen-Dekompositionsdiagramm abgeleiteten Rollen-Schnittstellen werden von den UML-Funktionsblöcke exportiert. Dies gilt auch für die, die als Gruppen-Rolle spezifiziert sind. UML-Funktionsblöcke, die Sammelschnittstelle exportieren, werden als *Funktionsblock-Container* bezeichnet. Ihre Spezifikation ist als vorläufig zu verstehen und wird während der Beschreibung des Pflichtenheftes aufgelöst (*Projekt-Dekompositionsprinzip*). Sie können weitere UML-Funktionsblöcke bzw. die Funktionsblockarchitektur einer anderen Ebene enthalten (siehe Bild 6.9).

Ist zum Zeitpunkt der Beschreibung des Lastenheftes bereits die Implementierung eines UML-Funktionsblocks gegeben (z.B. durch eine UML-Komponenten, siehe Kapitel 6.1.1), kann diese in Ver-

bindung mit einer Steuerungskomponente in die PF- bzw. PS-Funktionsblockarchitektur integriert werden (siehe Bild 6.9).

6.1.3.3 System-Dekomposition durch das Steuerungen-Interaktionsdiagramm

Die Konzeption der ODEMA-Methode sieht vor, System- und Steuerungen-Interaktionsdiagramme nicht nur bezüglich der Anforderungsdefinition in getrennten Ausgangssituationen einzusetzen. Es soll ebenso möglich sein, das System-Interaktionsdiagramm unter dem Aspekt der System-Dekomposition mit dem Steuerungen-Interaktionsdiagramm zu verbinden (siehe Kapitel 5.5.2). Hierfür wurde der Einbindungspunkt eingeführt (siehe Tabelle 6.2). Im Unterschied zum Erweiterungspunkt, wird dieser bezüglich der ODEMA-Methode nicht zur Beschreibung von Anwendungsfällen eingesetzt, da dies zu sehr komplexen und unübersichtlichen Anwendungsfalldiagrammen führt (*Anwendbarkeit*, siehe hierzu auch [PANG_01]).

Bezüglich der Interaktionsdiagramme des ODEMA-Beschreibungsmittels ist der Erweiterungspunkt in solchen Fällen einzusetzen, wenn UML-Funktionsblöcke in der Prozessführungs- und gleichzeitig in der Prozesssteuerungsebene bezüglich ihrer Spezifikation berücksichtigt werden müssen. Wie schon im Kontext der Entwicklung des Agenten- bzw. Steuerungen-Dekompositionsdiagramms dargelegt, obliegt es dem Entwickler Sender und Empfänger von vertikalen Nachrichten festzulegen. Die gleiche Entwurfsentscheidung ist im Zusammenhang mit den Interaktionsdiagrammen zu treffen. Ziel ist es dabei die funktionale Anforderungen zu dekomponieren und aus ihnen auch UML-Funktionsblöcke der Prozesssteuerungsebene abzuleiten. Zu diesem Zweck werden die Funktionen *sendPoint* und *receivePoint* eingeführt, die in Verbindung mit Marken *kausale* Punkte in Interaktionsdiagrammen markieren und mit einer Nummer versehen sind (siehe Bild 6.10). Das heißt, dass der Zeitpunkt des Sendens bzw. des Empfangens einer markierten vertikalen Nachricht ausschließlich zur kausalen Verknüpfung mit einem Einbindungspunkt gleicher Nummer in einem anderen Interaktionsdiagramm verwendet wird.

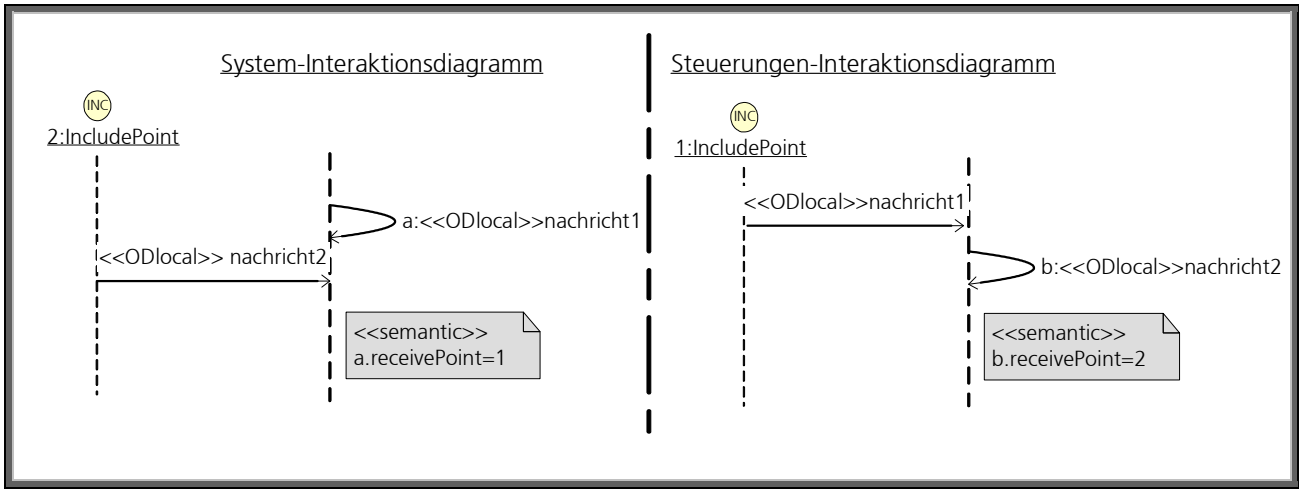

Bild 6.10: Darstellung der Sytem-Dekomposition im Kontext des System- und des Steuerungen-Interaktionsdiagramms

6.1.4 Verfahren zur Prüfung des Lastenheftes

Aus der Konzeption des Vorgehensmodells der ODEMA-Methode geht hervor, dass im Kontext des Lastenheft ausschließlich das Verfahren *Prüfung der funktionalen Anforderungen* zum Einsatz kommt (siehe Kapitel 5.5.5). Zu diesem Zweck sind die Prüfkriterien abzuleiten, die sich aus der Definition der Stereotypen ergeben (siehe Tabelle 6.1 bis 6.4a,b, sowie Tabelle 5.3a,b). Zudem können Konsistenz-sichernde Diagrammtypen zum Einsatz kommen. Dieses Verfahren ist in Tabelle 6.6 dargestellt.

Verfahren		**Konsistenz-sichernder Diagrammtyp**	**Prüfkriterien**
Prüfung der funktionalen Anforderungen	Syntaktische Konsistenz	-	• Die Typen der Parameter im Kontext der Deklaration von Operationen müssen durch einen elementaren Datentyp der ODEMA-Methode oder durch eine Daten-Klasse aus der Daten- oder Prozessvariablen-Bibliothek spezifiziert sein.
		• Agenten-Dekompositionsdiagramm • Steuerungen-Dekompositionsdiagramm	• Alle internen Akteure müssen in UML-Funktionsblöcke dekomponiert werden. • Alle Rollen der internen Akteure müssen in Rollen-, Gruppen- oder Sammel-Schnittstelle dekomponiert werden. • Alle Dekarationen von horizontalen und vertikalen Nachrichten der internen Akteure müssen auf entsprechende Schnittstellen-Klassen dekomponiert werden. • Alle atomaren Aktionen und nicht-unterbrechbaren Aktivitäten der internen Akteure müssen auf internen Schnittstellen oder Verhaltensklassen dekomponiert werden. • Die Einschränkungen von Typen- und Rollen-Vererbung müssen korrekt sein (siehe Tabelle 6.2)
		• Systemszenarien • Steuerungsszenarien	• Alle Anwendungsfälle verfügen über mindestens eine Anwendungsfall-Realisierung. • Eine Anwendungsfall-Realisierung verfügt über genau ein Sequenzdiagramm
			• Die Spezifikation der Erweiterungspunkte in den UML-Diagrammtyp-Rollen *System-* oder *Steuerungen-Interaktionsdiagramm* muss eine Untermenge der entsprechenden Spezifikationen bezüglich der UML-Diagrammtyp-Tollen System- oder Steuerungsarchitektur sein. • Die Spezifikation der Instanzen, die im Kontext eines System- oder Steuerungen-Interaktionsdiagramms beschrieben wird, muss eine Untermenge der internen Akteure aus dem Kontext des zugehörigen System- oder Steuerungsanwendungsfalls sein. • Horizontale Nachrichten in System-Interaktionsdiagrammen müssen eine Untermenge der Aufrufe in Agenten-Protokollen sein, die als *Classifier Lifecycle* dem entsprechenden System-Anwendungsfall zugeordnet sind.

Tabelle 6.5: Definition der Prüfung der funktionalen Anforderungen

6.2 Entwicklung der Methode zur Spezifikation des Pflichtenheftes

Das Pflichtenheft ist als Spezifikationsdokument Teil und Fortschreibung des Lastenheftes. Funktion des Pflichtenheftes als wesentliches Produkt der Entwicklungsphase *Fachtechnische Lösungskonzeption* ist der Nachweis der Machbarkeit der gestellten Anforderungen und die Beschreibung der Grund-

lagen zur Kalkulation einer Entwicklung. Letzteres erfordert, dass die im Pflichtenheft enthaltenen Spezifikationen die vollständige System- und Software-Dekomposition umfassen. Auf diese Weise ist spezifiziert, welche Komponenten-Implementierungen zu entwickeln und welche wieder zu verwenden sind. Die gilt auch für die Auswahl der notwendigen Automatisierungsgeräte.

6.2.1 Beschreibung der *Systemtechnischen Lösung*

Aufgabe des Abschnitts *Systemtechnische Lösung* ist die Spezifikation der Realisierung der geforderten funktionalen Anforderungen des geplanten Produktionssystems (Abschnitt 3 des Lastenheftes, siehe Kapitel 6.1.2). Je nach Ausgangssituation der Agentifizierung müssen PF- und PS-Funktionsblockarchitektur vollstänig spezifiziert sein, d.h. alle Funktionsblock-Container müssen aufgelöst werden (siehe Kapitel 6.1.3.2). Hierzu gehört auch die Spezifikation der zu einem UML-Funktionsblock gehörenden Steuerungskomponenten, d.h. die Beschreibung der UML-Diagrammtyp-Rollen *PF-* und *PS-Komponentenarchitektur*.

6.2.1.1 Vervollständigung der Funktionsblockarchitektur

Die vollständigen Dekomposition einer Sammel-Schnittstelle wird im Agenten- bzw. Steuerungen-Dekompositionsdiagramm beschrieben. Die dekomponierten Schnittstellen-Klassen sind Rollen-Schnittstellen und somit nicht weiter dekomponierbar. Sie können weitere Operationen enthalten, die nicht von der Sammel-Schnittstelle stammen, sondern aus der Dekomposition eines internen Akteurs stammen (siehe Bild 6.11). Zur Vervollständigung der Funktionsblockarchitektur gehört abschließend die Auflösung der Funktionsblock-Container, so dass die Beschreibung der PF- und der PS-Funktionsblockarchitektur ausschließlich aus UML-Funktionsblöcken und Rollen- bzw. Gruppen-Schnittstellen besteht.

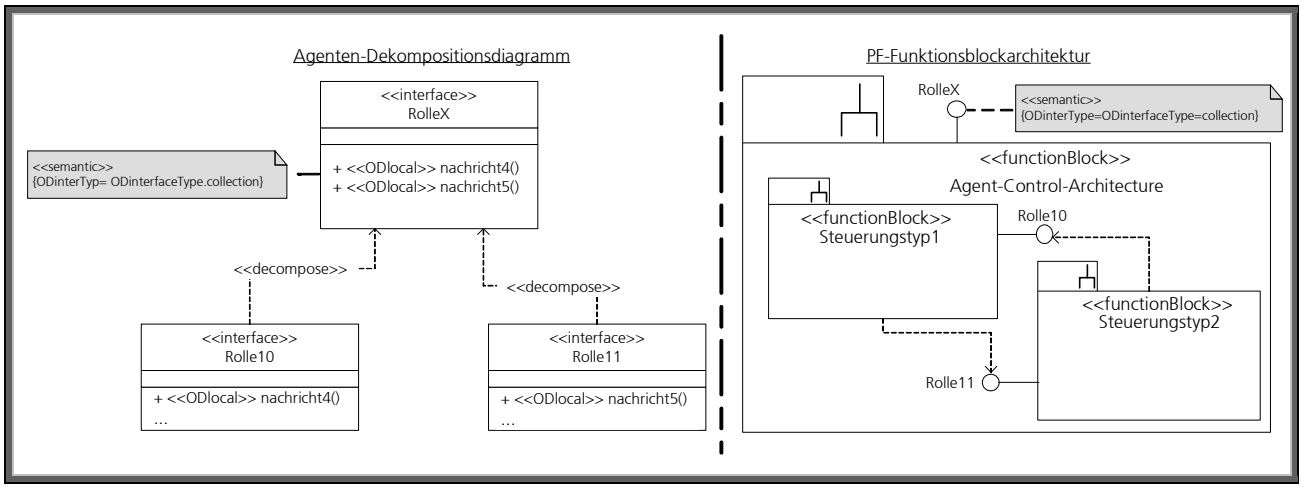

Bild 6.11: Exemplarische Darstellung des Agneten-Dekompositionsdiagramm im Kontext der Beschreibung der *Systemtechnischen Lösung*

6.2.1.2 Spezifikation der Software-Dekomposition durch Steuerungskomponenten

Gemäß der Konzeption des ODEMA-Beschreibungsmittels besteht jeder UML-Funktionsblock aus mindestens einer aktiven Steuerungskomponente. Im trivialen Fall besteht also eine PF- oder PS-

Komponentenarchitektur aus lediglich einer Steuerungskomponente. Passive Steuerungskomponenten können mittels interner Schnittstellen (Schnittstellen-Klasse mit Eigenschaftswert *ODinterTyp* = internal) mit den aktiven Steuerungskomponenten verbunden werden (siehe Bild 6.12). Außerdem ist es durch die Beschreibung der Software-Dekomposition durch eine weitere Stufe von UML-Subsystemen möglich, Komponenten-Implementierungen bzw. Automatisierungsgeräte als wieder zu verwendende Komponenten zu spezifizieren.

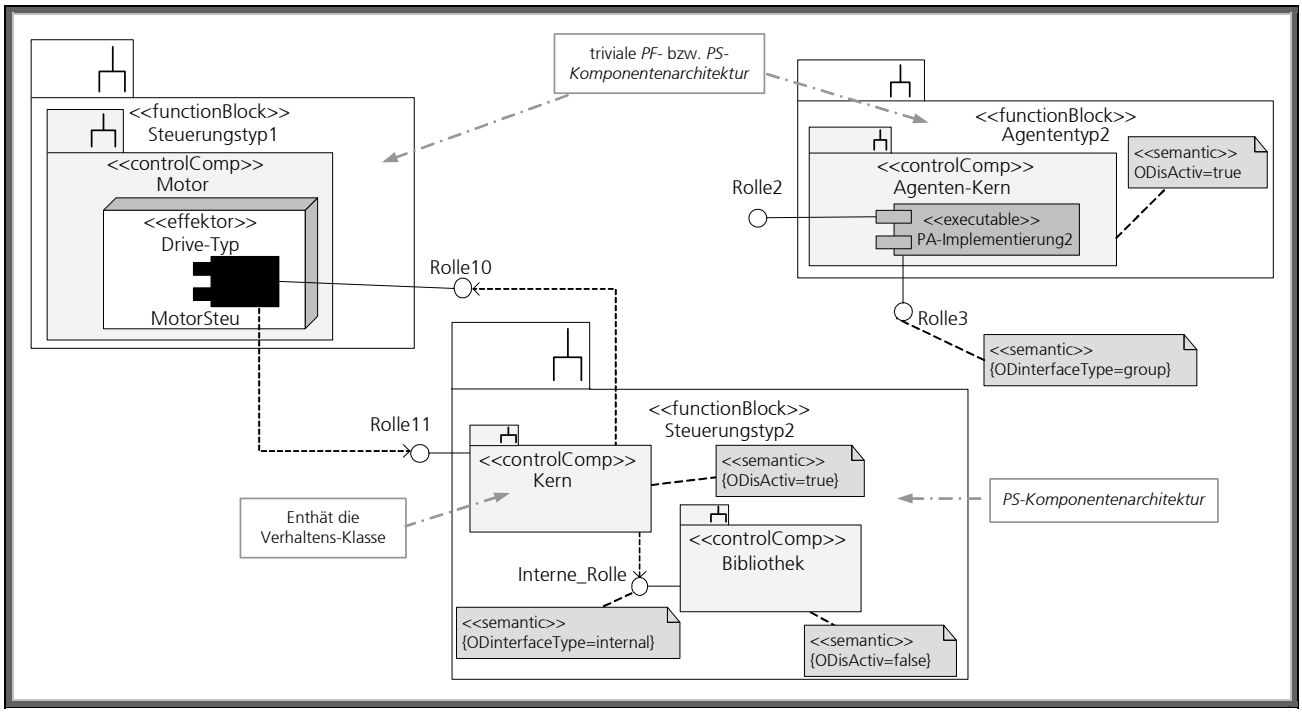

Bild 6.12: Exemplarische Darstellung der hierarchische Software-Dekomposition von UML-Funktionsblöcke mittels Steuerungskomponenten

6.2.2 Beschreibung der *Systemtechnik*

6.2.2.1 Entwicklung des PF- und PS-Implementierungsdiagramms

Durch die Beschreibung der PF- und PS-Komponentenarchitektur ist die Möglichkeit gegeben, aus diesen UML-Diagrammtyp-Rollen direkt das PF- und das PS-Implementierungsdiagramms abzuleiten. Steuerungskomponenten können entweder direkt durch eine UML-Komponente spezifiziert sein oder lassen sich basierend auf ihrer Beschreibung bereits zum Zeitpunkt der Spezifikation mit einer noch nicht umgesetzten UML-Komponente verknüpfen (z.B. ausführbares Programm, siehe Tabelle 6.1a).

Grundlage der UML-Diagrammtyp-Rollen *PF-* bzw. *PS-Implementierungsdiagramm* ist der UML-Diagrammtyp *Implementierungsdiagramm*. Dieser Diagrammtyp stellt ausschließlich die Typen (Kategorie *Classifier*) der UML-Komponenten dar (siehe Bild 0.4). Instanzen werden im *Einsatzdiagramm* beschrieben. Neben den im Kontext des Lastenheftes eingeführten ausführbaren Programmen (Stereotyp *executable*) und Programm-Bibliotheken (Stereotyp *library*) können in den Implementierungsdiagrammen auch beschreibende und eingebettete Implementierungen verwendet werden (siehe Tabelle 5.9 und 5.10).

Kommunikationskanäle werden eingesetzt, wenn ausführbare oder eingebettete Implementierungen über Rollen- bzw. Gruppen-Schnittstellen miteinander verknüpft sind. Im Gegensatz zu den anderen Stereotypen wird der Typ eines Kommunikationskanals ausschließlich durch die Typen der importierten bzw. exportierten Schnittstellen-Klassen spezifiziert. Auf die exportierte Schnittstelle eines Kommunikationskanals kann durch beliebig viele UML-Komponenten zugegriffen werden. Das Senden bzw. Empfangen einer Nachricht ist als eine atomare Aktion bzw. Methode anzusehen und ein gleichzeitiger Zugriff auf die exportierte Schnittstelle eines Kommunikationskanals führt zu keinerlei Kollision bzw. signifikanten Zeitverzögerung bezüglich der Übertragung einer horizontalen bzw. vertikalen Nachricht (siehe Bild 6.13).

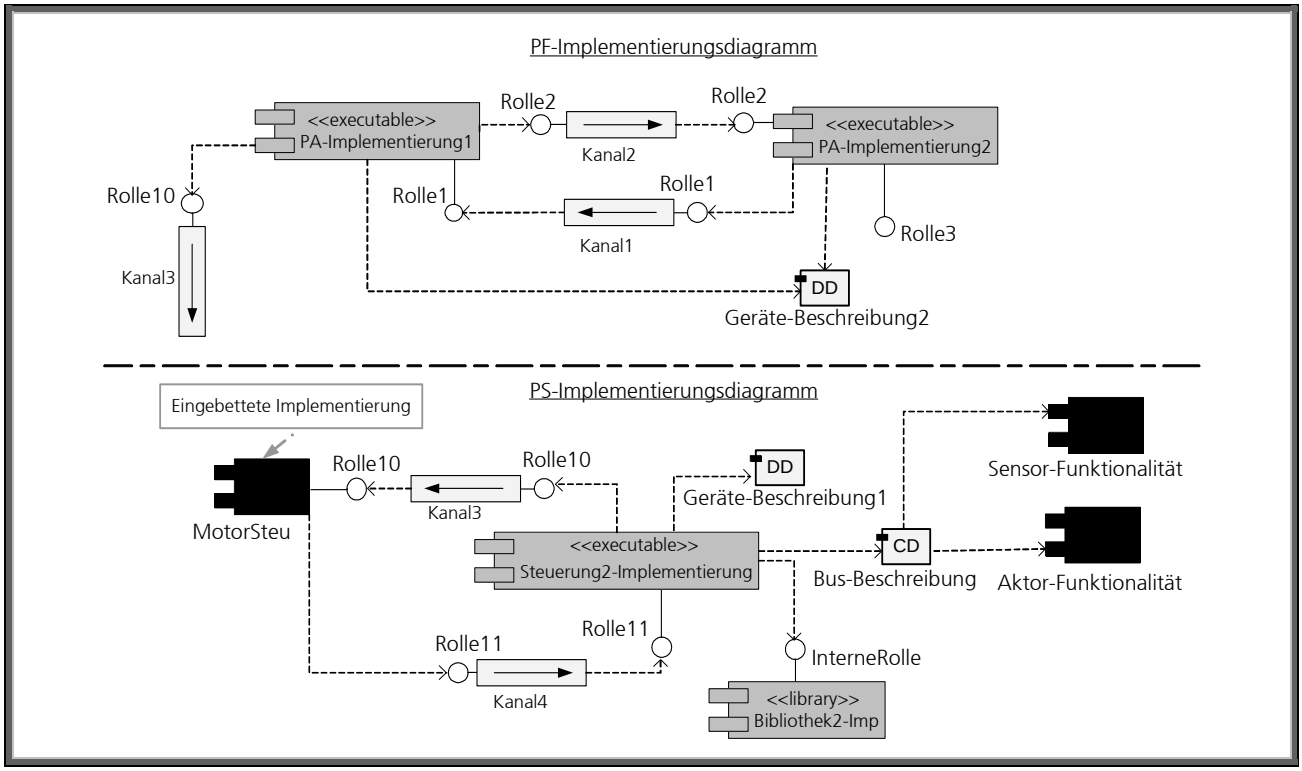

Bild 6.13: Exemplarische Darstellung des PF- und des PS-Implementirungsdiagramms

Der Zugriff auf eine importierte Schnittstellen-Klasse eines Kommunikationskanals ist dahingehend zu unterscheiden, ob es sich dabei um eine Rollen-Schnittstelle oder um eine Gruppen-Schnittstelle handelt. Bei Anwendung einer Gruppen-Schnittstelle als importierte Schnittstelle eines Kommunikationskanals, ist es durch die Definition dieses Schnittstellen-Typs möglich, dass diese von mehr als einer UML-Komponente exportiert wird. Hieraus folgt, dass das Senden einer Nachricht, also der Aufruf einer der auf der Schnittstellenklasse deklarierten Operationen, das gleichzeitige Empfangen der identischen Nachricht bei allen verbundenen UML-Komponenten zur Folge hat. Dies gilt für Instanzen von UML-Komponenten gleichen und unterschiedlichen Typs. Genauer gesagt sind in der physischen Sicht die Sender und Empfänger von Nachrichten die Instanzen einer UML-Komponente und somit gleichzeitig die Instanzen der Steuerungskomponenten (*Softwarekomponentenmodell*).

Bei Anwendung einer Rollen-Schnittstelle ist dies zwar auch möglich, aber unterscheidet sich hierbei die Semantik des Kommunikationskanals im Detail. Wird also eine gleiche Rollen-Schnittstelle, importiert von einem Kommunikationskanal, von mehr als einer Instanz einer UML-Komponente exportiert,

ohne dass hierfür eine Agenten- oder Steuerungsgruppe spezifiziert wurde, so kann dieser Konflikt durch unterschiedliche Instanzen des gleichen Typs von Kommunikationskanal im Installationsdiagramm aufgelöst werden.

Eine andere Lösung ist die Annahme, dass die auf der Rollenschnittstelle deklarierten Nachrichten über einen Parameter verfügen, der durch die Beschreibung der Identität des Empfängers sicherstellt (z.B. durch eine eindeutige Nummer zur Identifizierung), dass die empfangenen UML-Komponenten-Instanzen feststellen können, ob eine Nachricht an sie gerichtet wurde oder an eine andere UML-Komponenten-Instanz, die die identische Rollen-Schnittstelle exportiert (siehe Bild 6.14).

Beschreibende Implementierungen exportieren bzw. importieren keine Schnittstellen-Klassen. Ausführbare Programme greifen auf die Operationen zu, die durch die entsprechenden beschreibenden Klassen spezifiziert werden. Dies wird mittels einer Abhängigkeit dargestellt (siehe Bild 6.13). Da die beschreibende Implementierung eines Gerätes ausschließlich atomare Aktionen beschreibt, kann hier mehr als eine Instanz eines ausführbares Programm auf die Instanz einer solchen UML-Komponente zugreifen. Beschreibende Implementierungen der Kommunikationssysteme sind explizit so definiert, dass nur eine einzige Instanz eines ausführbaren Programms oder einer eingebetteten Implementierung auf die spezifizierten Operationen zugreifen kann, da sonst der spezifizierte *offered QoS* (Attribut *syncDelay*, siehe Kapitel 5.5.6.3) nicht garantiert werden kann.

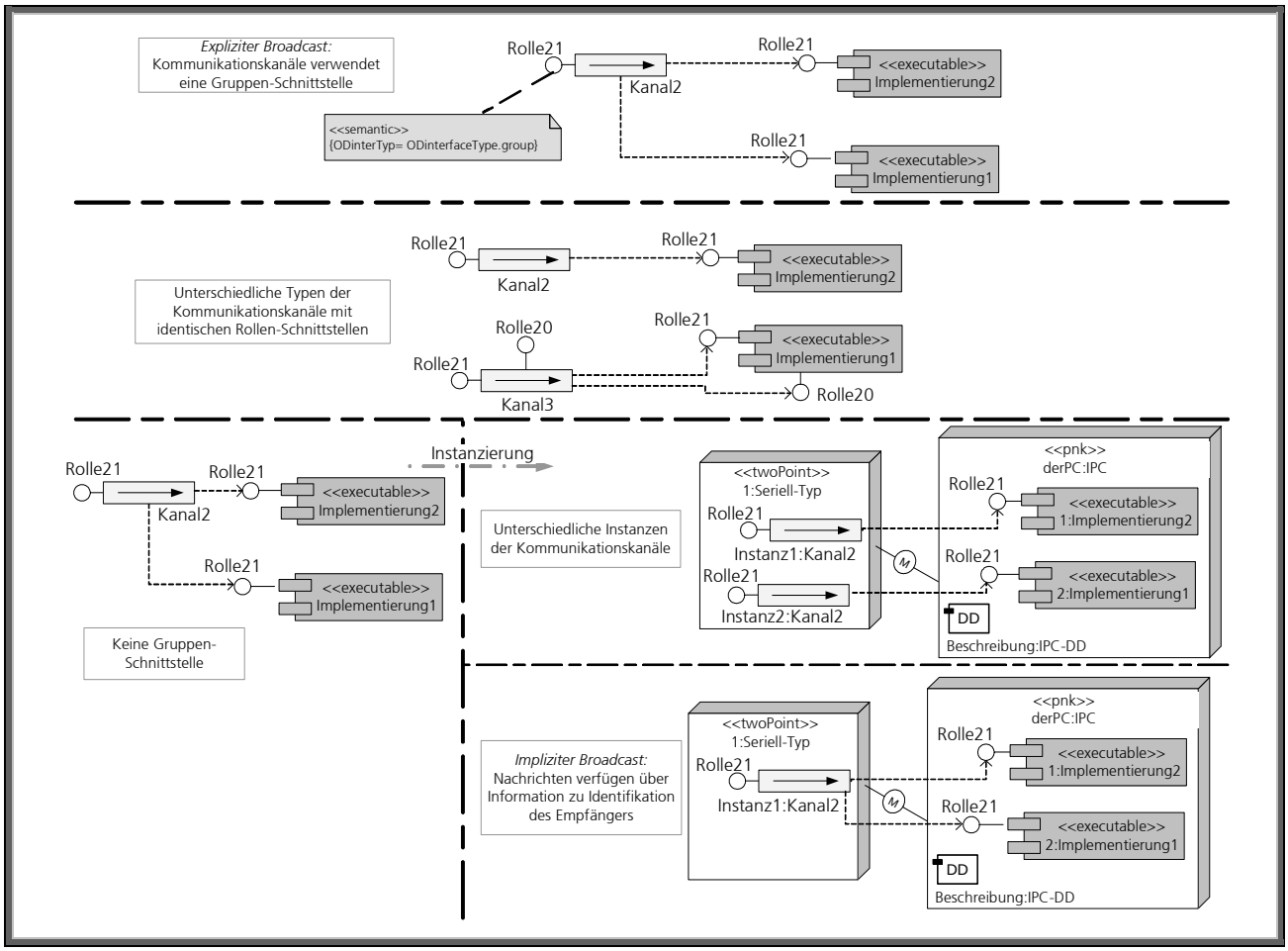

Bild 6.14: Exemplarische Darstellung der Beschreibung vom Kommunikationskanalen im Kontext von PF- und PS-Implementierungsdiagrammen

6.2.2.2 Vervollständigung des Installationsdiagramms

Das Installationsdiagramm ist bereits im Rahmen des Abschnitts der Spezifikation des bestehenden Produktionssystem bzw. seiner Steuerung eingeführt worden (siehe Kapitel 6.1.1). Das dem Installationsdiagramm zu Grunde liegende Einsatz-Diagramm beschreibt Instanzen von UML-Knoten und UML–Komponenten (siehe Bild 0.3).

Im Kontext der Erstellung des Pflichtenheftes erhält das Installationsdiagramm eine erweiterte Aufgabenstellung. Neben der Beschreibung der Gerätearchitektur auf Basis der UML-Geräte- und UML-Kommunikationssystem-Beschreibung wird in diesem Zusammenhang spezifiziert, welches temporale Verhalten die Instanzen von Steuerungskomponenten zur Laufzeit aufzeigen. Hierzu gehört auch die Prüfung auf semantische Widerspruchsfreiheit (siehe Kapitel 6.2.3).

Zunächst beschreibt die Instanzierung die Verteilung einer UML-Komponenten-Inszanzen auf Instanzen von UML-Knoten. Hierbei ist zu beachten, dass eine beschreibende Implementierung nur einmal auf einer UML-Knoten-Instanz vorkommen darf. Die beschreibende Implementierung eines Kommunikationssystems darf als Instanz maximal mit einer weiteren Instanz eines ausführbaren Programms oder einer eingebetteten Implementierung verbunden werden (*Synchrones Kommunikationssystem*). UML-Knoten-Instanzen, die über eine eingebettete Implementierung verfügen, nehmen keine weitere Instanz einer UML-Komponente auf.

Der größte Freiheitsgrad bezüglich der Beschreibung des Installationsdiagramms ist in der Gestaltung der Kommunikationskanäle gegeben (siehe Bild 6.14). Die Spezifikation der Typen und Instanzen von Kommunikationskanälen ermöglicht es dem Entwickler, die Anforderungen an das temporale Verhalten der Steuerung zu erfüllen, indem durch die Verteilung der Schnittstellen-Klassen auf Kommunikationskanäle bzw. durch die Bildung entsprechender Instanzen Nachrichten bestimmten Kommunikationssystemen zugewiesen werden. Das Attribut *asyncDelay* der entsprechenden beschreibenden Klasse beschreibt hierbei den *offered QoS* eines Kommunikationskanals und bestimmt somit den aus temporaler Sicht günstigsten Kommunikationskanal (siehe Kapitel 5.5.6.3).

Mit der Verteilung der UML-Komponenten als Instanzen ist noch keinerlei Festlegung getroffen, wie sich diese zur Laufzeit verhalten. Grundsätzlich kann im Kontext der Einsatzdiagramme jede Instanz eines ausführbaren Programms bzw. einer eingebetteten Implementierung zur Laufzeit beliebig viele Instanzen von Steuerungskomponenten erzeugen (Komponentenmodell). Dieser Teil der Semantik der UML ist durch das UML-Profil der ODEMA-Methode einzuschränken. Die eingeführten Stereotypen bezüglich der UML-Komponenten sind mit Ressourcen wie Kommunikationskanälen und beschreibenden Implementierungen für Kommunikationssysteme verknüpft. Eine beliebige Anzahl von Instanzen zur Laufzeit führt in diesem Kontext zu mehrdeutigen Beschreibungen. Aus diesem Grund wird nur eine Instanz einer Steuerungskomponente bezogen auf eine Instanz eines ausführbaren Programms oder eingebetteten Implementierung zugelassen. Auf diese Weise wird die Beschreibung der Kommunikation mittels Kommunikationskanälen zur Spezifikation der Kommunikation zwischen Instanzen von Steuerungskomponenten (siehe Tabelle 6.6).

Stereotyp	Basis-Klasse	Eigenschaftswerte	Einschränkungen (*formal*/informal)
Ausführbares Programm <<executable>>	(Siehe Tabelle 6.1	-	• Enthält zur Laufzeit nur eine Instanz einer Steuerungskomponente
Eingebettete Implementierung <<embedded>>	(Siehe Tabelle 5.13)	-	• Enthält zur Laufzeit nur eine Instanz einer Steuerungskomponente

Tabelle 6.6: Ergänzung zu den Tabellen 5.13 und 6.1

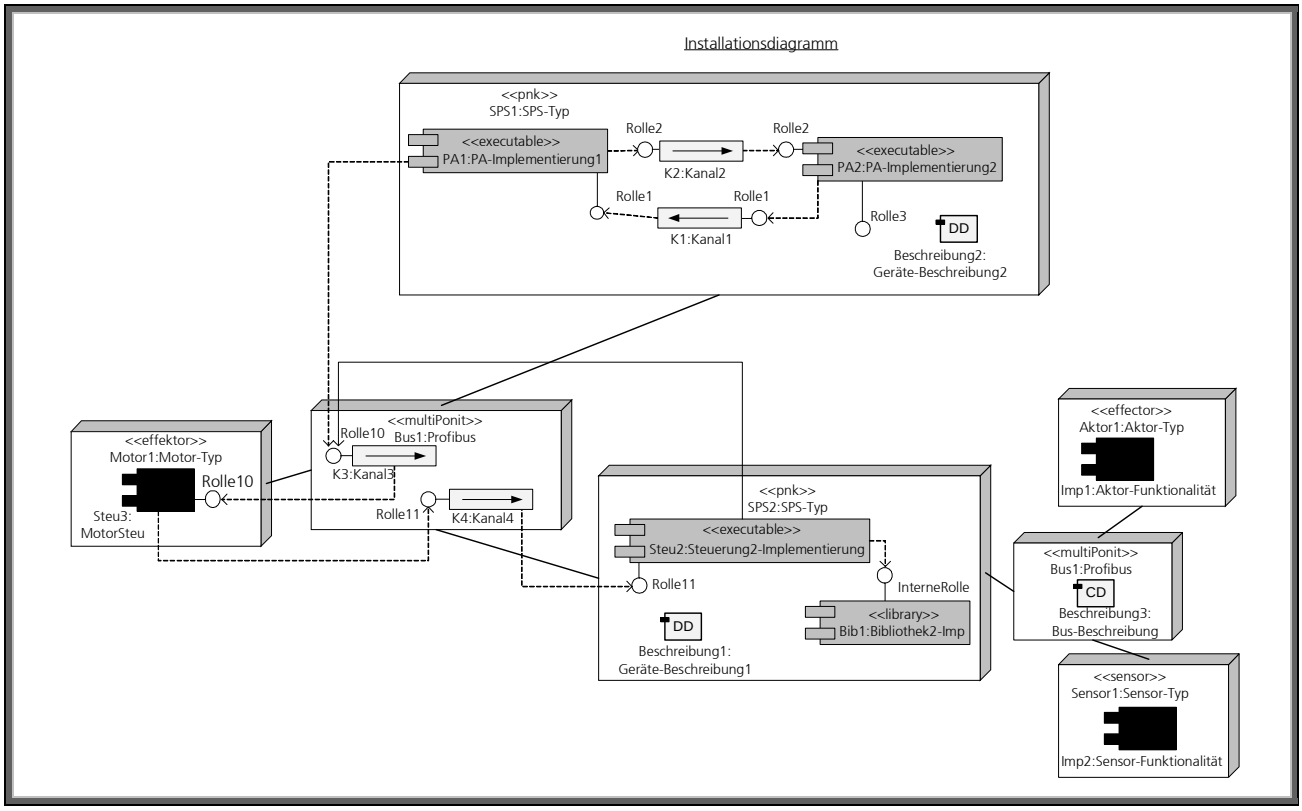

Bild 6.15: Exemplarische Darstellung des Installationsdiagramms (Darstellung ohne Agenten-Rahmen)

Darüber hinaus müssen im Installationsdiagramm nicht alle Abhängigkeiten dargestellt werden, die sich aus dem PF- und PS-Implementierungsdiagramm ergeben. Aus Gründen der Übersichtlichkeit - d.h. der *Anwendbarkeit* - kann auf die Darstellung der Abhängigkeiten bezogen auf beschreibende Implementierungen verzichtet werden, da sie bereist aus den entsprechenden Komponentendiagrammen hervorgehen. Grundsätzlich ist jedes Installationsdiagramm im Kontext des Pflichtenheftes einem Agentenrahmen bzw. physischen Rahmen zugeordnet.

6.2.3 Verfahren zur Prüfung des Pflichtenheftes

Das Pflichtenheft beschreibt die vollständige Dekomposition der statischen Beschreibung der Steuerung (Kategorie *Classifier Relationship Model*). Da keine UML-Diagrammtyp-Rollen zur Verhaltensbeschreibung verwendet werden, verfügt das Verfahren *Prüfung des Pflichtenheftes auf Konsistenz* über keinerlei Definitionen bezüglich einer Validation. Tabelle 6.8 beschreibt das Verfahren *Prüfung des Pflichtenheftes auf Konsistenz* anhand der eingeführten Prüfkriterien im Detail. Konsistenz-sichernde Diagrammtypen werden nicht angewandt, mit Ausnahme bezüglich der Dekomposition der verbliebenen Sammel-Schnittstellen, da das Pflichtenheft ausschließlich Beschreibungstechniken zur Dekomposition verwendet, die durch die UML bereits definiert sind.

Verfahren		Konsistenz-sichernder Diagrammtyp	Prüfkriterien
Prüfung des Pflichtenheftes auf Konsistenz	Syntaktische Konsitenz	-	Der durch den Eigenschaftswert *ODagentType* spezifizierte Agententyp eines Agenten-Rahmens muss auch im Kontext der System-Anwendungsfälle als interner Akteur (Agententyp) verwendet werden.
		Agenten- und Steuerungen-Dekompositionsdiagramm	Vollständige Dekomposition der Sammel-Schnittstellen
		-	Vollständige Dekomposition aller UML-Funktionsblöcke
		-	Vollständige Beschreibung der Verteilung aller UML-Komponenten als Instanzen bezüglich der UML-Knoten im Installationsdiagramm.
		-	Jeder UML-Funktionsblock enthält genau eine aktive Steuerungskomponente
		-	Jede von einem Kommunikationskanal importierte Schnittstellen-Klasse wird von dem selben Kommunikationskanal auch exportiert.
		-	Ein Kommunikationskanal kann nur ausführbare Programme (Typen und Instanzen) verbinden.
		-	Jede exportierte Schnittstellen-Klasse eines UML-Funktionsblocks ist ebenso die exportierte Schnittstellen-Klasse der dazugehörigen aktiven Steuerungskomponente und deren Komponenten-Implementierung.
			Die temporalen Spezifikationen im System- und Steuerungen-Interaktionsdiagramm müssen in sich konsistent sein (siehe Tabelle 6.8).
	Semantische Widerspruchs-freiheit	-	Die Ungleichungen Gl.2 bis Gl.4 aus dem Echtzeitmodell müssen erfüllt sein (siehe Kapitel 3.1.7 und Tabelle 6.9)

Tabelle 6.7: Definition des Verfahrens Prüfung des Pflichtenheftes auf Konsistenz (- = Es wird kein konsitenzsichernder Diagrammtyp benötigt)

Wesentlicher Unterschied zum *Verfahren Prüfung der funktionalen Anforderungen* ist die Berücksichtigung der Spezifikationen des *Temporalprinzips* der Steuerung. Semantische Widerspruchsfreiheit bezüglich dieser Spezifikationen bedeutet zum einen syntaktische Konsistenz der temporalen Spezifikation in sich selbst, da die UML in diesem Zusammenhang ausschließlich informale Beschreibungstechniken zur Verfügung stellt (siehe Tabelle 6.8). Zu Anderen sind die Forderungen des Echtzeitmodells (siehe Kapitel 3.1.7) zu integrieren. Dies formuliert die Tabelle 6.9 im Detail. Bezüglich der temporalen Spezifikationen wird nicht zwischen weichen und harten temporalen Spezifikationen unterschieden. Bei Verletzung, d.h. bei semantischen Widerspruch einer harten, temporalen Spezifikation ist die Beschreibung des Pflichtenheftes ungültig. Bei weichen, temporalen Spezifikationen ist es die Entscheidung des Entwicklers, ob den Grad der Verletzung einer temporalen Spezifikation toleriert oder die Beschreibung für ungültig erklärt.

Temporale Spezifikation	QoS		UML-Diagrammtyp-Rolle	Definition der syntaktischen Konsistenz
	off.	requ.		
Echtzeitschranke: Zeitfunktionen *sendTime* (Zeitpunkt t_{SEND}) und *receiveTime* (Zeitpunkt t_{REC}) spezifizieren eine maximale Zeitschranke Δt. Hierbei gilt: $\Delta t = t_{REC} - t_{SEND}$		X	System- und Steuerungen-Interaktionsdiagramm	• Angenommen, dass eine zeitgesteuerte Schleife (Zeitfunktion *loopTime*, Zeitspanne Δt_{LOOP}) sich innerhalb einer spezifizierten Zeitschranke befindet, dann ist gefordert: $\mathbf{\Delta t > \Delta t_{LOOP}}$ • Angenommen, dass eine nicht-unterbrechbare Aktivität (Zeitfunktion *executionTime,* Δt_{EXE}) sich innerhalb einer spezifizierten Zeitschranke befindet, dann ist gefordert: $\mathbf{\Delta t > \Delta t_{EXE}}$
Ausführungszeit: Eine nicht unterbrechbare Aktivität wird durch die maximalen Ausführungszeit $\Delta t_{EXE.MAX}$ (Eigenschaftswert *ODrequiredTime*) spezifiziert.		X	• System- und Steuerungen-Interaktionsdiagramm • Agenten-Dekompositionsdiagramm • Steuerungen-Dekompositionsdiagramm	Wenn *T1* die Menge aller Spezifikationen von Δt_{EXE} einer nicht-unterbrechbaren Aktivität in verschiedenen Sequenzdiagrammen ist, dann ist gefordert: $\mathbf{\Delta t_{EXE} \leq \Delta t_{EXE,MAX} \quad \forall \Delta t_{EXE} \in T1}$
Zeitgesteuerte Schleife: Eine zeitgesteuerte Schleife (Zeitfunktion *loopTime*, Zeitspanne Δt_{LOOP}) ist spezifiziert.		X	System- und Steuerungen-Interaktionsdiagramm	• Angenommen, dass eine Zeitschranke Δt sich innerhalb der spezifizierten zeitgesteuerten Schleife befindet, dann ist gefordert: $\mathbf{\Delta t_{LOOP} > \Delta t}$ • Angenommen, das eine nicht-unterbrechbare Aktivität (Zeitfunktion *executionTime,* $\mathbf{\Delta t_{EXE}}$) sich innerhalb einer spezifizierten zeitgesteuerten Schleife befindet, dann ist gefordert: $\mathbf{\Delta t_{LOOP} > \Delta t_{EXE}}$

Tabelle 6.8: Varianten temporaler Spezifikation und die entsprechenden Definitionen der syntaktischen Konsistenz (off.=offered, requ.=required, siehe Tabelle 6.7)

Echtzeitmodell (Kapitel 3.1.7)	QoS		UML-Diagrammtyp-Rolle	Definition der semantischen Widerspruchsfreiheit
	off.	requ.		
Gleichung 2: $\mathbf{t_{Prozess} \geq 2*t_{Zvkl,SP} + 3*t_{Zvkl,Kom}}$	X	X	• System- und Steuerungen-Interaktionsdiagramm • Installationsdiagramm • Agenten- und Steuerungen-Dekompositionsdiagramm • Komponenten-Spezifikation	Wenn $\mathbf{t_{Prozess}}$ als Zeitschranke $\mathbf{\Delta t}$, die Zykluszeit eines Steuerungsprozesses $\mathbf{t_{Zvkl.SP}}$ als Zeit-gesteuerter Schleifenaufruf $\mathbf{\Delta t_{LOOP}}$ spezifiziert und die Zykluszeit eines Kommunikationssystems $\mathbf{t_{Zvkl,Kom}}$ durch das Attribut *syncDelay* der beschreibenden Klasse bzw. beschreibenden Implementierung eines Kommunikationssystems spezifiziert ist, dann gilt: $\mathbf{\Delta t \geq 2*\Delta t_{LOOP} + 3*\ t_{syncDelay}}$
Gleichung 3: $\mathbf{t_{Prozess} \geq t_{V,Kanal} + t_{V,SP}}$	X	X	• System- und Steuerungen-Interaktionsdiagramm • Installationsdiagramm	Wenn $\mathbf{t_{Prozess}}$ als Zeitschranke $\mathbf{\Delta t}$, die zeitliche Verzögerung eines Kommunikationskanals $\mathbf{t_{V.Kanal}}$ als Attribut *asyncDelay* (Beschreibende Implementierung eines Kommunikationssystems) und die Antwortzeit eines Steuerungsprozesses $\mathbf{t_{V,SP}}$ als Attribut *maxResponseTime* (Beschreibende Implementierung eines Gerätes) spezifiziert ist, dann gilt: $\mathbf{\Delta t \geq t_{asyncDelay} + t_{maxResponseTime}}$
Gleichung 4: $\mathbf{t_{Prozess} \geq \sum_{x=1}^{x=m_{K+SP}} t_{V,Kanal,x} + t_{V,SP,x}}$	X	X	• System- und Steuerungen-Interaktionsdiagramm • Installationsdiagramm	Definition lässt sich direkt aus Gl. 3 ableiten.

Tabelle 6.9: Definition nach sematischen Widerspruchsfreiheit durch das Echtzeitmodell (off. = offered, requ. = required , siehe Tabelle 6.8)

6.3 Entwicklung der Methode zur Spezifikation des objektorientierten Steuerungsentwurfs

Der *Objektorientierte Steuerungsentwurf* ist ein internes Spezifikationsdokument des Entwicklers und somit keiner Norm oder Richtlinie unterworfen. Aufgabe dieses Dokumentes ist es, die bereits spezifizierten Steuerungskomponenten, soweit sie nicht in bereits implementierter Form vorliegen, als *White-Box* zu beschreiben. Im Kontext der Konzeption der ODEMA-Methode wird dieser Abschnitt *Detail-Spezifikation* genannt (siehe Bild 5.19). Die hierfür angewendeten UML-Diagrammtyp-Rollen sind die *Komponenten-Spezifikation*, das *Komponenten-Verhaltensdiagramm*, die *Algorithmen-Spezifikation* und das *Daten-* bzw. *Prozessvariablen-Diagram*, die alle zum Spezifikationsabschnitt einer Steuerungskomponente gezählt werden (siehe Bild 5.18).

6.3.1 Entwicklung der Komponentenspezifikation

Die UML-Diagrammtyp-Rolle *Komponenten-Spezifikation* beschreibt den Strukturellen Aufbau einer Steuerungskomponente. Neben den bereits eingeführten Varianten der Schnittstellen-Klassen ist auch die bereits eingeführte Verhaltensklasse Teil der Komponentenspezifikation. Die Verhaltensklasse wurde im Kontext des *Agenten-* bzw. *Steuerungen-Dekompositionsdiagramms* vorgestellt und spezifiziert den Typ eines internen Akteurs mittels der Klassen-Attribute, atomaren Aktionen und nicht-unterbrechbaren Aktivitäten. Zudem ist der Verhaltensklasse ein Zustandsdiagramm zugeordnet, das ebenfalls als UML-Diagrammtyp-Rolle *Komponenten-Verhaltensdiagramm* bereits eingeführt wurde (siehe Kapitel 6.1.3.1).

Zunächst erfordert es die Semantik der UML, dass die exportierten Schnittstellen-Klassen durch eine Klasse innerhalb der Steuerungskomponente implementiert werden. Da die Deklarationen der Operationen im Kontext der ODEMA-Methode als Nachrichten verstanden werden, müssen die als Gruppen- und Rollen-Schnittstellen spezifizierten Schnittstellen-Klassen von der Verhaltensklasse implementiert werden. Diese enthält die Beschreibung des Komponenten-Verhaltensdiagramm, welches die Kausalität der Steuerungskomponente auf Basis der eingehenden Nachrichten beschreibt. Interne Schnittstellen müssen spezifiziert werden, wenn atomare Aktion und nicht-unterbrechbare Aktivitäten unter dem Aspekt der Wiederverwendung in eine passive Steuerungskomponente ausgelagert werden. Im Kontext des Agenten- bzw. Steuerungen-Dekompositionsdiagramms ist die interne Rolle als Außensicht einer passiven Steuerungskomponente, also als eine exportierte Schnittstellen-Klasse bereits eingeführt worden (siehe Kapitel 6.1.3.1). Aus Sicht der Komponenten-Spezifikation gehören die spezifizierten Operationen einer internen Rolle zum Namensraum der passiven Steuerungskomponente. Somit wird die Klasse, die die interne Rolle implementiert, in den Namensraum der aktiven Steuerungskomponente importiert und mittels einer Abhängigkeit mit der entsprechenden Verhaltensklasse verbunden (*Fassaden-Klasse*, siehe Bild 6.16).

Fassaden-Klassen (UML-Klasse mit dem Stereotyp *ODfacade*) sind bezüglich der Spezifikation von aktiven Steuerungskomponenten eine optionale Beschreibung. Sie sind aus dem Entwurfsmuster *Fassade*

abgeleitet und werden im Kontext der ODEMA-Methode als Zusammenfassung von atomaren Aktionen und nicht-unterbrechbaren Aktivitäten mit den dazu gehörigen Daten-Klassen verstanden. Bezüglich einer aktiven Steuerungskomponente stellt der Einsatz der Fassadenklasse eine Entwurfsentscheidung dar, falls die Spezifikation der Verhaltensklasse durch eine große Anzahl von entsprechenden Operationsdeklarationen zu unübersichtlich wird (Methodenaxiom *Anwendbarkeit*). Eine weitere Entscheidung des Entwickler legt fest, ob die Fassaden-Klasse in eine passive Steuerungskomponente ausgelagert wird oder in dem Namensraum der aktiven Steuerungskomponente verbleibt (siehe Bild 6.16 und 6.17). Unabhängig von der Zugehörigkeit zu einem bestimmten Stereotyp einer Klasse oder einem bestimmten Typ Steuerungskomponente wird jede atomare Aktion und nicht-unterbrechbare Aktivität bezüglich ihres Verhaltens durch eine *Algorithmen-Spezifikation* (Aktivitätsdiagramm) beschrieben.

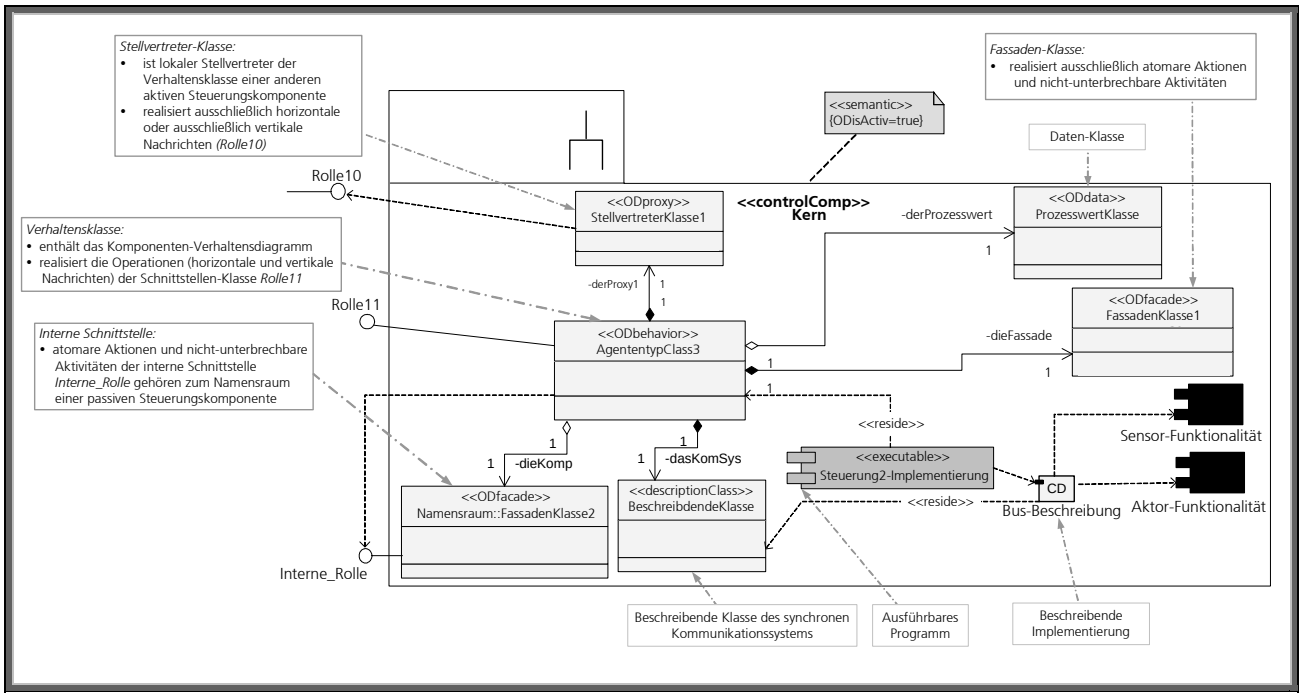

Bild 6.16: Exemplarische Darstellung einer Komponenten-Spezifikation für eine aktiven Steuerungskomponente

Um die zu importierenden Schnittstellen-Klassen (Gruppen- und Rollen-Schnittstellen) in die Komponenten-Spezifikation zu integrieren, wird die Stellvertreter-Klasse (Klasse mit dem Stereotyp *ODproxy*) eingeführt, die aus dem gleichnamigen Entwurfsmuster abgeleitet wird [GAMMA_97]. Die Stellvertreter-Klasse wird hierbei als lokaler Stellvertreter der Verhaltensklasse einer anderen aktiven Steuerungskomponente verstanden, wobei ausschließlich die Operationsdeklarationen, also Spezifikationen von Nachrichten, der importierten Rollen- bzw. Gruppen-Schnittstellen einer bestimmten Steuerungskomponente im Namensraum dieser Stellvertreter-Klasse spezifiziert werden.

Da die Daten-Klassen gemäß ihrer Konzeption als Tei der Beschreibung von Steuerungskomponenten anzusehen sind, müssen sie bezüglich der bisher beschrieben stereotypisierten Klassen eingeordnet werden. Aus der nachrichtenbasierten Kommunikation folgt, dass Datenobjekte als Wert und als Spezifikation eines Parameters einer Nachricht an eine Steuerungskomponente übergeben werden. Um zugleich eine konsistenten Zustand der Datenobjekte zu gewährleisten, ist es innerhalb der Steue-

rungskomponente somit ausschließlich der Verhaltensklasse erlaubt, Datenobjekte zu verändern. Ein Zugriff auf Datenobjekte von außerhalb ist prinzipiell durch die Konzeption des UML-Funktionsblocks nicht zulässig (siehe Kapitel 5.5.1).

Für die Beschreibung der entsprechenden Daten-Klassen im Kontext der Komponenten-Spezifikation folgt hieraus, dass sie mittels einer Aggregation (*aggregation*) mit der Verhaltensklasse verknüpft werden und als Spezifikation eines Kommunikationsobjektes zu verstehen sind. Ist eine Daten-Klasse Teil der Spezifikation einer importierten Schnittstellen-Klasse, so wird dies durch eine Abhängigkeit zwischen der betreffenden Stellvertreter-Klasse und der Daten-Klasse ausgedrückt. Prozessobjekte sind nur innerhalb einer Steuerungskomponente bekannt. Dies wird mittels einer Komposition (*composition*) mit der Verhaltensklasse beschrieben.

Abschließend wir die Integration der beschreibenden Klasse in den Kontext der Komponenten-Spezifikation dargestellt. Sie ist optional und wird zur Spezifikation einer aktiven Steuerungskomponente nur dann verwendet, wenn in der physischen Sicht die Komponenten-Implementierung eines solchen Softwarekomponentenmodells mit einer UML-Geräte-Beschreibung oder einer UML- Kommunikationssystem-Beschreibung (*Synchrones Kommunikationssystem*) verbunden ist (siehe Kapitel 6.2.2). Da beschreibende Klassen konzeptionell der Fassaden-Klasse sehr ähnlich sind, werden sie auf die gleiche Weise mit der entsprechenden Verhaltensklasse verbunden (siehe Bild 6.16). Bezüglich eines synchronen Kommunikationssystems gilt aber, dass die beschreibende Klasse per Komposition mit der Verhaltensklasse verbunden wird, da nur eine Steuerungskomponente auf die Operationen dieser Klasse zugreifen darf (siehe Kapitel 5.5.6.3). Prinzipiell können Komponenten-Implementierungen im Kontext der Komponenten-Spezifikation beschrieben werden (*Komponentenmodell* und – *Implementierung*, siehe Kapitel 5.5.1). Kommunikationskanäle werden gemäß ihrer Konzeption in der Komponenten-Spezifikation nicht berücksichtigt, da sie transparent sind.

Stereotyp	**Basis-Klasse**	**Eigenschaftswerte**	**Einschränkungen (*formal*/informal)**
Fassaden-Klasse <<ODfacade>>	Class	-	*IsActive=false*
Stellvertreter-Klasse <<ODproxy>>	Class	-	*IsActive=false*

Tabelle 6.10: Abschnitt des UML-Profils der ODEMA-Methode bezüglich der Komponenten-Spezifikation

Prinzipiell unterscheidet sich die Komponenten-Spezifikation von aktiven und passiven Steuerungskomponenten nicht von einander. Lediglich Verhaltensklassen, Stellvertreter-Klassen und beschreibende Klassen werden zur Spezifikation von passiven Steuerungskomponenten nicht verwendet (siehe Bild 6.17). Hieraus folgt, dass die zentrale Rolle, die die Verhaltensklasse im Zusammenhang mit der Komponenten-Spezifikation einer aktiven Steuerungskomponente einnimmt, in einer passiven Steuerungskomponente von einer Fassaden-Klasse übernommen wird. Grundsätzlich kann hierbei aber mehr als eine Fassaden-Klasse spezifiziert sein und auch exportiert werden.

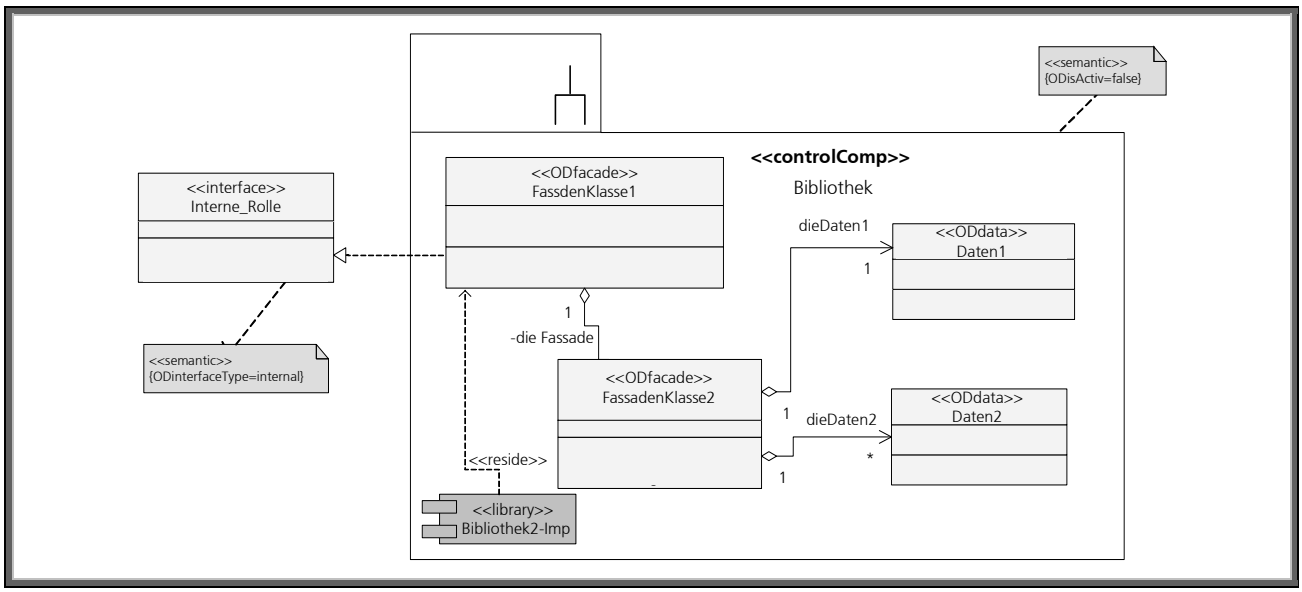

Bild 6.17: Exemplarische Darstellung einer Komponenten-Spezifikation für eine passiven Steuerungskomponente

6.3.2 Entwicklung der Komponenten-Verhaltensdiagramms und der Algorithmen-Spezifikation

Die UML-Diagrammtyp-Rollen *Komponenten-Verhaltensdiagramm* und *Algorithmen-Spezifikation* beschreiben das Verhalten von Steuerungskomponenten, wobei das Komponenten-Verhaltensdiagramm - also die spezifische Diagrammtyp-Rolle des Zustandsdiagramms - das Ereignis-basierte Verhalten beschreibt und ausschließlich der aktiven Steuerungskomponente zugeordnet ist. Die Algorithmen-Spezifikation ist die spezifische Rolle eines Aktivitätsdiagramms und ist in aktiven sowie passiven Steuerungskomponenten zu finden (siehe Kapitel 6.3.1 und 5.6).

6.3.2.1 Entwicklung des Komponenten-Verhaltensdiagramms als Erweiterung des Zustandsdiagramms

Die Entwicklung des Komponenten-Verhaltensdiagramms erfordert zum einen die Integration der UML-Beschreibungstechniken aus dem System- bzw. dem Steuerungen-Interaktionsdiagramms und zum anderen die geeignete Einschränkung der Syntax des Zustandsdiagramms, um die Anforderungen bezüglich des *Temporal*- und *Kausalprinzips* von Steuerungen zu berücksichtigen.

Entsprechend der Definition des *Rollen*-Begriffs umfasst dieser nicht nur die Spezifikation einer Schnittstelle nach außen, sondern auch die Beschreibung eines spezifischen Verhaltens (siehe Kapitel 2.9.3). Zudem findet ein Wechsel der Rolle autonom statt, d.h. der Steuerungsprozess entscheidet ohne eine externe Zwangssteuerung selbst über diesen Wechsel. Um basierend auf dem Zustandsdiagramm ein Verhalten zu kapseln und einer Rolle zuzuweisen, stellt die UML den zusammengesetzten Zustand (*Composite State*) zur Verfügung. Dieser beschreibt die hierarchische Verfeinerung bzw. Dekomposition eines Zustandsdiagramms innerhalb eines Zustands.

Zur Spezifikation des spezifischen Verhaltens einer Rolle wird hier der *Rollen-Zustand* (Zusammengesetzter Zustand mit dem Stereotyp *ODrole*) eingeführt (siehe Tabelle 6.11 und Bild 6.18). Zwischen der Spezifikation einer Rolle (z.B. im Steuerungen-Interaktionsdiagramm) und dem dazugehörigen Rollen-

Zustand besteht ausschließlich über den identischen Namen ein Zusammenhang. Hierbei ist zu berücksichtigen, dass gemäß dem Rollen-Begriff keine von außen mittels Ereignissen gesteuerten Zustandsübergänge (*transitions*) in einen Rollen-Zustand hinein bzw. aus ihm heraus führen dürfen. Die UML bietet die Möglichkeit die Änderung einer boolesche Bedingung als *Änderungs-Ereignis* (*change event*) zu spezifizieren. Dies wird durch das in der UML definierte Ereignis *when(boolescher Ausdruck)* beschrieben. Hierbei wird der Zustandsübergang, der den Wechsel einer Rolle spezifiziert, durch einen booleschen Ausdruck formuliert, der wiederum auf einem Klassen-Attribut von Datentyp *Boolean* basiert. Dies Konzept wurde bereits im Kontext der Spezifikation des Rollen-Wechsels in Sequenzdiagrammen eingeführt (*Bedingung*, siehe Tabelle 6.3). Änderungs-Ereignisse werden innerhalb eines Steuerungsprozesses erzeugt und konsumiert und können somit die Anforderung, die aus dem Rollen-Begriff hervorgeht, erfüllen.

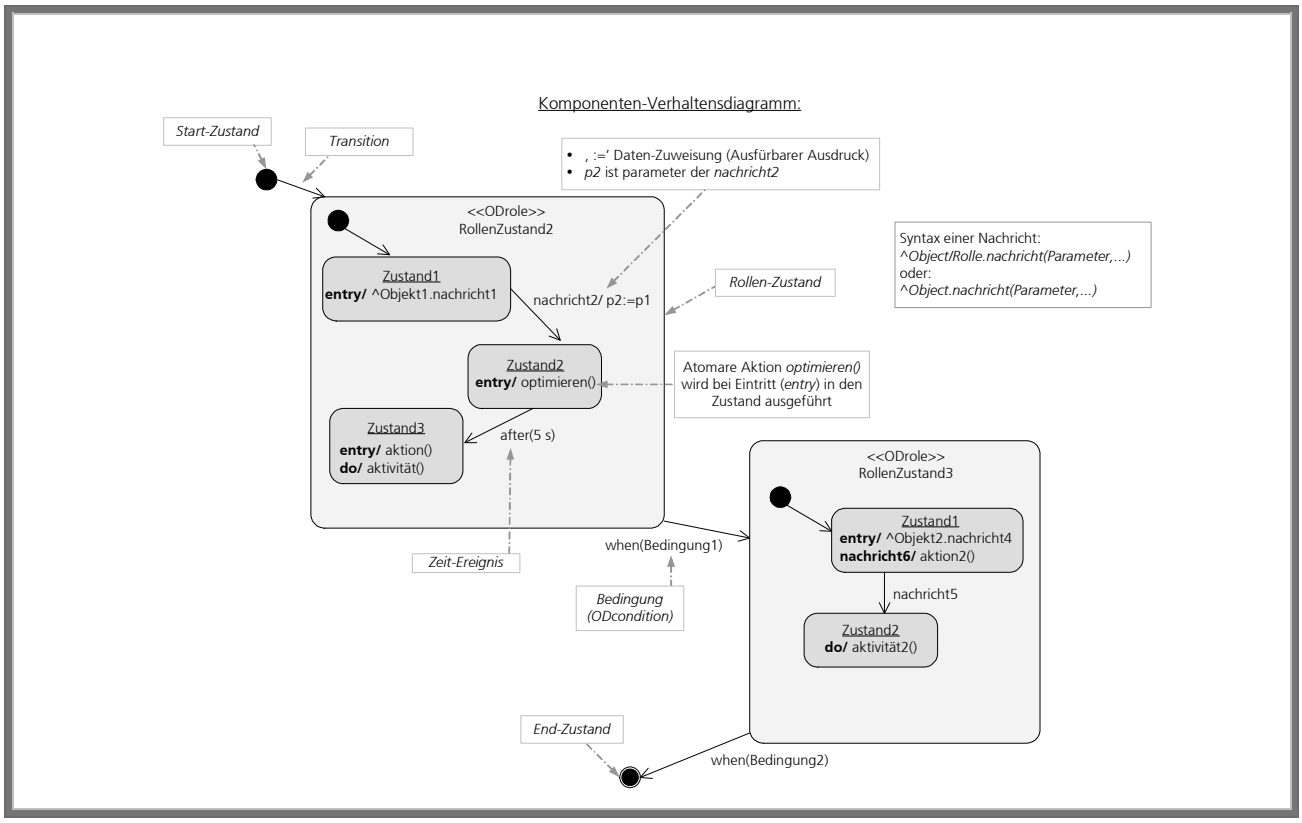

Bild 6.18: Exemplarische Darstellung der UML-Diagrammtyp-Rolle *Komponenten-Verhaltensdiagramms*

Stereotyp	**Basis-Klasse**	**Eltern-Klasse**	**Eigenschaftswerte**	**Einschränkungen (*formal*/informal)**
Rollen-Zustand <<ODrole>>	CompositState	N/A	-	• *Self.isConcurrent=false* • ausschließlich Änderungs-Ereignisse führen in und aus einem Rollen-Zustand • Ausnahme Kind-Zustand: horizontale/vertikale Nachrichten können von einem Kind- zu einem Eltern-Zustand führen (beide sind Rollen-Zustände)
Schleifen-Zustand <<ODloop>>	State	N/A	-	-

Tabelle 6.11: Abschnitt des UML-Profils der ODEMA-Methode bezüglich des Komponenenten-Verhaltensdiagramms

Der im Zusammenhang mit der Entwicklung des Steuerungs- bzw. System-Interaktionsdiagramms eingeführte Stereotyp *Schleifen-Aufruf* ist ebenfalls in das Komponenten-Verhaltensdiagramm zu integrieren. Prinzipiell lässt sich dies auf zwei Weisen realisieren. Zum einen bietet ein Aktivitätsdiagramm mittels der Beschreibungselemente *Zusammenführung* und *Verzweigung* die Möglichkeit logische Verzweigungen im Ablauf darzustellen und somit auch bedingte und unbedingte Schleifen (siehe Bild 6.20). Als zweite Beschreibungstechnik ist der so genannte Schleifen-Zustand einzuführen (Zustand mit dem Stereotyp *ODloop*, siehe Tabelle 6.12). Verbunden mit einer Selbst-Transition und einem *Zeit-Ereignis* (*TimeEvent*) kann das zyklisch Ausführen von atomaren Aktionen und nicht-unterbrechbaren Aktivitäten spezifiziert werden ([BRAATZ_03a], siehe Bild 6.19). Im Zusammenhang mit der Zeitfunktion *loopTime*, die im System- und Steuerungen-Interaktionsdiagramm zum Einsatz kommt, wurde bezüglich der Beschreibung der zyklischen Aktivierung des Schleifen-Zustandes das Zeitereignis *every(Zeit-Ausdruck)* bereits eingeführt (siehe Tabelle 6.12).

Ereignis	UML-Semantik	Beschreibung
Every(TimeExpression)	TimeEvent	Zyklische Aktivierung nach einer durch den Zeit-Ausdruck (TimeExpression) spezifizierten Zeitspanne (Physikalische Einheit *ms* oder *s*)
After(TimeExpression)	TimeEvent	Siehe [UML_03]

Tabelle 6.12: Zeit-Ereignisse in Komponenten-Verhaltensdiagrammen

Da die Schleifen-Aufrufe Teil der Beschreibung der nachrichtenbasierten Kommunikation der Steuerungsprozesse nach außen sind und allgemein das Senden und Empfangen von Ereignissen bzw. Nachrichten mittels Zustandsdiagrammen spezifiziert wird, wird der Schleifen-Zustand zur Integration der Schleifen-Aufrufe in das Komponenten-Verhaltensdiagramm gewählt.

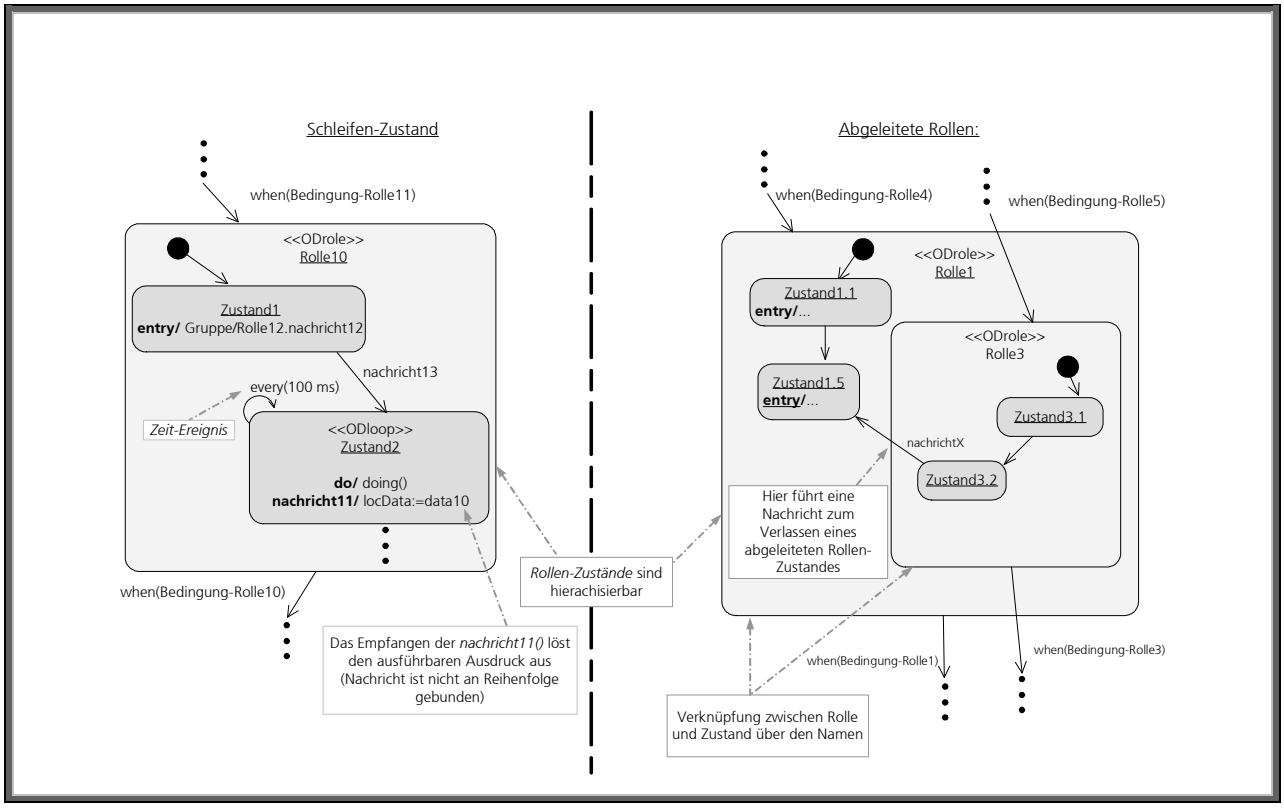

Bild 6.19: Exemplarische Darstellung der Beschreibungstechniken des *Schleifen-Zustands* und der Ableitung einer Rolle durch hierarchisierte Zustände

Allgemein beschreibt ein Zustandsdiagramm das Verhalten einer Klasse mittels Zustandsübergängen zwischen Zuständen und Aktionsbezeichnern (*action Label*) innerhalb eines Zustandes. Aktionen können während eines Zustandsübergangs, beim Eintritt (Aktionsbezeichner *entry*) und beim Verlassen (Aktionsbezeichner *exit*) eines Zustandes sowie im Zuge einer so genannten internen Transitionen - d.h. bei Nachrichten bzw. Ereignisse, die keine Zustandsübergänge auslösen - ausgeführt werden. Die Verwendung aller dieser UML-Beschreibungstechniken in Kombination macht die Spezifikation für den Entwickler nicht nur unübersichtlich, sondern führt auch zu einer Einschränkung des Kausalprinzips von Steuerungen. Folgende Punkte beschreiben die hieraus hervorgehende Einschränkungen bezüglich der Nutzung des Zustandsdiagramm als Komponenten-Verhaltensdiagramm:

- Zustandsübergänge benötigen im Kontext der UML keine Ausführungszeit. Dies ist entscheidend zur Beschreibung von Steuerungen, da diese sich stets in einem definierten Zustand und bereit zur Reaktion auf eingehende Nachrichten bzw. Ereignisse sein müssen (siehe Tabelle 2.1). Wird eine Aktion beim Verlassen eines Zustands und eine weitere während des Zustandsübergangs ausgeführt, so entsteht ein mit zeitlicher Verzögerung behafteter Zustandsübergang. Aus diesem Grund kommen bezüglich der ODEMA-Methode der Aktionsbezeichner *exit* und die Ausführung einer atomaren Aktion während eines Zustandsübergangs nicht zum Einsatz. Ausnahme hierbei ist die Zuweisung des Parameterwertes einer Nachricht, die einen Zustandsübergang auslöst. Hierbei ist die Zuweisung eines solchen Wertes an eine Komponenten-interne Variable als Teil der atomaren Ausführung des Zustandsübergangs anzusehen (siehe Bild 6.18).
- Die Hierarchisierbarkeit von Zuständen führt dazu, dass bei Eintritt in einen Unterzustand oder bei Verlassen eines Unterzustandes entsprechende durch die Aktionsbezeichner *entry* und *exit* gekennzeichnete Aktionen ausgeführt werden. Auch hierbei entstehen zeitliche Verzögerungen bei Zustandsübergängen und dadurch bedingt der undefinierte Zustand einer Instanz. Das Komponenten-Verhaltensdiagramm lässt Aktionsbezeichner daher ausschließlich in den am weitesten dekomponierten, also einfachen Zuständen zu.
- Aktivitäten im Kontext von Zustandsdiagrammen werden durch den Aktionsbezeichner *do* aufgerufen und ihre Ausführung kann solange andauern, bis ein entsprechender Zustandsübergang ausgelöst wird. Nicht-unterbrechbare Aktivitäten, wie sie für das ODEMA-Beschreibungsmittel eingeführt wurden, verfügen über die Spezifikation einer maximalen Ausführungszeit (Eigenschaftswert *ODrequiredTime*) und können durch eingehende Nachrichten bzw. Ereignisse nicht unterbrochen werden.

Prinzipiell wird die Verwendung von orthogonalen Zuständen ausgeschlossen, da es innerhalb einer Steuerungskomponente gemäß der Definition nur einen Kontrollfluss geben kann (UML-Funktionsblock). Neben Aktionen und Aktivitäten, die mittels ihrer Operations-Deklaration aufgerufen werden, lässt die UML auch die Formulierung eines Ausdrucks (*expression*) zu, der einfache algebraische oder logische Operationen beschreibt. Ausdrücke werden direkt im Zustandsdiagramm angege-

ben, verfügen über keine Operations-Deklaration und sind als atomar anzusehen. Da die UML in der Version 1.5 noch keine so genannte *Action Language* spezifiziert, sind diese Ausdrücke lediglich informal. Um das Kausalprinzip von Steuerungen vollständig zu spezifizieren, wird zu diesem Zweck der *ausführbare Ausdruck* eingeführt (siehe Kapitel 6.3.2.2).

Abschließend muss die Ableitung (*Vererbung*) von Rollen berücksichtigt werden, die im Kontext des *Agenten-* und *Steuerungen-Dekompositionsdiagramms* eingeführt wurde (siehe Kapitel 6.1.3.1). Allgemein geht die UML davon aus, dass Ableitungen von Klassen im Kontext der dazugehörigen Zustandsdiagramme durch Hierarchisierung von Zuständen berücksichtigt werden (siehe Kapitel 5.4.2). Die Ableitung von Rollen im Kontext der ODEMA-Methode erfordert es darüber hinaus, dass jeder aus einer Rolle abgeleiteter Rollen-Zustand seine eigene Verhaltens-Beschreibung besitzt und im Fall, dass dieser aus einer anderen Rolle abgeleitet ist, ebenso auf die eingehenden und damit geerbten Nachrichten reagieren kann. Hieraus folgt, dass eine abgeleitete Rolle zu einem Unterzustand (*Kind-Zustand)* eines Rollen-Zustand (*Eltern-Zustand*) führt, der selbst wiederum ein Rollen-Zustand ist. Beide Rollen-Zustände sind unabhängig von einander durch Transitionen erreichbar. Ausgelöst werden diese Transitionen i.d.R. durch entsprechende Änderungsereignisse gemäß dem Rollen-Begriff.

Ausnahme bildet hierbei die Trabsisition zwischen einem Kind- und einem Elternzustand, wenn beide als Rollenzustand spezifiziert sind (siehe Tabelle 6.11). Grund für diese Ausnahme ist, dass von ausserhalb eines Steuerungsprozesses gesehen nicht zu erkennen ist, in welcher der voneinander abgeleiteten Rollen ein Steuerungsprozess sich gerade befindet. Es findet kein expliziter d.h. beobachtbarer Rollen-Wechsel statt. Beide Rollen werden von außen betrachtet quasi gleichzeitig eingenommen.

6.3.2.2 Einführung des Ausführbaren Ausdrucks

Mit der *Object Constraint Language* (OCL) steht in der UML eine formale Beschreibungssprache zur Verfügung, mit deren Hilfe algebraische und logische Ausdrücke formuliert werden können. Allerdings ist die OCL eine Sprache ohne so genannten *Seiten-Effekt*, d.h. es können ausschließlich *Einschränkungen* (z.B. Invarianten) bezogen auf eine UML-Beschreibung spezifiziert werden [UML_03].

Das Konzept der Verknüpfung von Elementare OCL-Datentypen (*Integer*, *Real*, *Boolean* und *String*) mittels algebraisch und logischen Operationen (z.B. +, -, *and* und *or*) ist Grundlage des hier zu entwickeln *Ausführbaren Ausdruck*. Dieser Ausdruck wird eingeführt, um Berechnungen von Variablen zu beschreiben und ist daher mit einem Seiten-Effekt behaftet. Hierfür wird der Zuweisungs-Operator *:=* eingeführt, der es erlaubt einer Variablen (*Attribut* einer Klasse oder *Parameter* einer Nachricht) einen Wert bzw. das Ergebnis einer Operation zuzuweisen.

Zudem werden die bezüglich der ODEMA-Methode eingeführten elementaren Datentypen verwandt. Ausführbare Ausdrücke können an Stelle von atomaren Aktionen in Komponenten-Verhaltensdiagrammen und auch Algorithmen-Spezifikationen eingesetzt werden (siehe Bild 6.19 und Bild 6.20). Prinzipiell können auch Kombinationen von verschiedenen ausführbaren Ausdrücken ato-

mar behandelt werden. Dies gilt z.B. für die Spezifikation mehrerer ausführbare Ausdrücke im Kontext eines Aktionsbezeichners (z.B. *entry/ ausdruck1, Ausdruck2, ...*)

Stereotyp	Basis-Klasse	Einschränkungen (*formal*/informal)
Ausführbarer Ausdruck << *ODExecutivExpression* >>	Expression	• Ausdruck, der im Komponenten-Verhaltensdiagramm und in der Algorithmen-Spezifikation eingesetzt wird • ist atomar • ist nicht frei von Seiten-Effekten

Tabelle 6.13: Abschnitt des UML-Profils der ODEMA-Methode bezüglich des *Ausführbaren Ausdrucks* (es werden keine Eltern-Klassen und Eigenschaftswerte verwendet)

6.3.2.3 Entwicklung der Algorithmen-Spezifikation

Atomare Aktionen und nicht-unterbrechbare Aktivitäten wurden bisher ausschließlich in Form von Operations-Deklarationen und Prozedur-Aufrufen berücksichtigt. Um komplexe Algorithmen zu beschreiben, also Berechnungen, die sich nicht durch einen einfachen ausführbaren Ausdruck beschreiben lassen, werden gemäß der Konzeption Aktivitätsdiagramme verwendet. Ebenso wie im Fall des Komponenten-Verhaltensdiagramms müssen Einschränkungen bezüglich der durch die UML verfügbaren Beschreibungstechniken durchgeführt werden. Weitere Stereotypen oder Eigenschaftswerte werden nicht eingeführt.

Grundsätzlich werden zur Beschreibung der UML-Diagrammtyp-Rolle *Algorithmen-Spezfikation* ausschließlich die Aspekte des Programmier-technisch orientierten Beschreibungsmittels verwandt (siehe Kapitel 2.5). Hieraus folgt, dass sog. S*wimlanes*, die Rollen in den Kontext von Aktivitätsdiagrammen abbilden, nicht benötigt werden. Auch die Möglichkeit zur Spezifikation des Objektflusses (*object flow*) wird nicht verwendet.

Geht man zunächst von der ausschließlichen Anwendung von ausführbaren Ausdrücken in Aktions-Zuständen (*action state*) aus, so unterscheidet eine atomare Aktion und eine nicht-unterbrechbare Aktivität lediglich die Entscheidung des Entwicklers, ob die Ausführungszeit vernachlässigt werden kann oder nicht.

Atomare Aktionen und ausführbare Ausdrücke unterscheiden sich aber in der Hinsicht, dass die Kombination von Aufrufen mehrerer ausführbarer Ausdrücke immer noch atomar sein kann. Die Aufrufe von ausführbaren Ausdrücken in einem Aktivitätsdiagramm ist eine solche Kombination. Kombiniert man Aufrufe von atomare Aktionen, so ist diese Algorithmen-Spezifikation stets als nicht-unterbrechbare Aktivität anzusehen. Nicht-unterbrechbare Aktivitäten dagegen können beliebig viele weitere nicht-unterbrechbare Aktivitäten enthalten (siehe Bild 6.20).

Grundsätzlich gilt für die Algorithmen-Spezifikation, dass sie keine Zustandsübergänge auslösen können und auch keine Nachrichten bzw. Ereignisse versendet werden können. Dies folgt unmittelbar aus der Konzeption des UML-Funktionsblocks, der eine strikte Trennung des Ereignis-basierten und des algorithmischen Teils der Verhaltens-Beschreibung fordert.

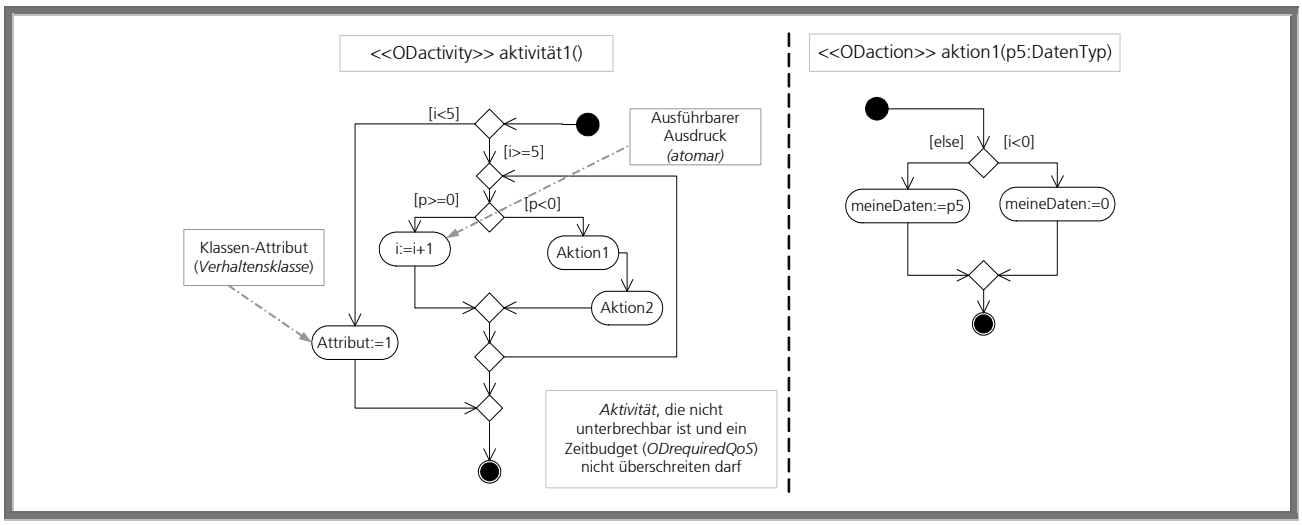

Bild 6.20: Exemplarische Darstellung der UML-Diagrammtyp-Rolle *Algorithmen-Spezifikation*

Die in Bild 6.20 exemplarisch dargestellten Algorithmus-Spezifikationen verwenden neben Parametern und Klassen-Attributen (Verhaltensklasse oder Fassaden-Klasse) auch lokale Variablen aus dem Namensraum einer Operation (*i* und *p*, siehe Bild 6.20). Diese Variablen haben i. d. R. Hilfsfunktionen (z.B. ist *i* Zählvariablen in einer Schleife) und sind nicht bezüglich ihres Datentyps deklariert, da die UML hierfür keine Beschreibungstechnik zur Verfügung stellt. Ein CASE-Tool muss dem Entwickler daher im Fall der Programmcode-Generierung entsprechende Konfigurations-Möglichkeiten zur Verfügung stellen. Grundsätzlich gilt aus Gründen der Konsistenz, dass auch lokale Variablen auf elementare Datentypen abgebildet werden müssen.

6.3.3 Verfahren zur Prüfung des objektorientierten Steuerungsentwurfs

Der *Objektorientierte Steuerungsentwurf* enthält als einziges Spezifikationsdokument der ODEMA-Methode eine Beschreibung des spezifizierten Verhaltens einer Steuerung. Somit gehört zu dem Verfahren *Prüfung des objektorientierten Steuerungsentwurf* (siehe Tabelle 6.15 und 6.16) ergänzend ein Verfahren zur Durchführung eines *Test*. Diese Verfahren wird im rahmen dieser Arbeit nicht entwickelt, sondern aus dem Stand der Technik übernommen (siehe z.B. Kapitel 2.9.2).

Verfahren		**Konsistenz-sichernder Diagrammtyp**	**Prüfkriterien**
Prüfung des objektorientierten Steuerungsentwurfs	Syntaktische Konsistenz	-	Die Schnittstellen-Klassen, die von einer aktiven Steuerungskomponente exportiert werden, müssen von der dazugehörigen Verhaltensklasse implementiert werden.
		-	Die Operations-Deklarationen entweder aller *lokaler* oder aller *horizontaler Nachrichten*, die in Form einer Schnittstellen-Klassen von einer aktiven Steuerungskomponente importiert werden, müssen von mindestens einer Stellvertreter-Klasse implementiert werden. Die Schnittstellen-Klasse ist hierbei mittels einer UML-Abhängigkeit mit der Stellvertreter-Klasse verbunden.
		-	Die Schnittstellen-Klassen, die von einer passiven Steuerungskomponente exportiert werden, müssen von der dazugehörigen Fassaden-Klasse implementiert werden.
		Agenten- und Steuerungen-Dekompositionsdiagramm	Alle aus einem internen Akteur dekomponierten Klassen müssen in der entsprechenden Diagrammtyp-Rolle Komponenten-Spezifikation berücksichtigt werden.

		-	Werden Parameter von Nachrichten auf exportierten oder importierten Schnittstellen-Klassen durch Daten-Klassen spezifiziert, so müssen diese Daten-Klasse innerhalb der Steuerungskomponente bekannt sein (Komponenten-Spezifikation oder Daten-Diagramm).
		-	Jede beschreibende Klasse darf innerhalb einer Steuerungskomponente nur einmal instanziert werden (*Singleton*).
		-	Jede exportierte Schnittstellen-Klasse einer Steuerungskomponente führt zu einem entsprechenden Rollen-Zustand im Komponenten-Verhaltensdiagramm.
		-	In einem Rollen-Zustand dürfen nur die Ereignisse bzw. Nachrichten zur Spezifikation des Verhaltens verwendet werden, die auf der entsprechenden Schnittstellen-Klasse spezifiziert sind.
		-	Die syntaktische Konsistenz der temporalen Spezifikationen *Maximale Ausführungszeit* und *Schleifen-Zustand* aus Tabelle 6.15 müssen erfüllt sein.
	Semantische Widerspruchs-freiheit	-	Bedingungen (Stereotyp *ODcondition*) dürfen von Algorithmen-Spezifikationen nicht verändert werden.
		-	Die Widerspruchsfreiheit der temporalen Spezifikationen *Schleifen-Aufruf* und *Überwachungszeit* aus Tabelle 6.15 muss erfüllt sein.

Tabelle 6.14: Prüfkriterien des Verfahrens zur Prüfung des Objektorientierten Steuerungsentwurfs (- = es wird kein Konsistenz-sichernder Diagrammtyp benötigt)

Temporale Spezifikation	**QoS**		**UML-Diagrammtyp-Rolle**	**Definition**
	of.	**req.**		
Maximale Ausführungszeit (syntaktische Konsistenz)		X	• Komponenten-Spezifikation • Algorithmen-Spezifikation	Die maximale Ausführungszeit $\Delta t_{EXE,MAX,1}$ (*ODrequiredTime*) einer nicht-unterbrechbaren Aktivität, die innerhalb einer weiteren nicht-unterbrechbaren Aktivität aufgerufen wird, muss kleiner als deren maximale Ausführungszeit $\Delta t_{EXE,MAX,2}$ sein. Es ist gefordert: $\Delta t_{EXE,MAX,2} > \Delta t_{EXE,MAX,1}$
Schleifen-Zustand (syntaktische Konsistenz)		X	• Komponenten-Spezifikation • Komponenten-Verhaltensdiagramm	Wird eine nicht-unterbrechbaren Aktivität in einem Schleifen-Zustand aufgerufen, so muss die maximale Ausführungszeit $\Delta t_{EXE.MAX}$ kleiner sein als der Zeit-Ausdruck Δt_{everv} des Zeit-Ereignisses *every()*. Es ist gefordert: $\Delta t_{every} > \Delta t_{EXE,MAX}$
Schleifen-Aufruf: Ein zeitgesteuerter Schleifen-Aufruf wird durch einen zeitgesteuerten Schleifen-Zustand umgesetzt *(syntaktische Konsistenz*).		X	• System- und Steuerungen-Interaktionsdiagramm • Komponenten-Verhaltensdiagramm	Wird ein zeitgesteuerter Schleifen-Aufruf durch die Zykluszeit Δt_{loop} und der Zeit-Ausdruck des Zeit-Ereignisses *every()* durch Δt_{every} spezifiziert so ist gefordert: $\Delta t_{loop} = \Delta t_{every}$
Überwachungszeit: Zwei Nachrichten werden von der identischen Instanz gesendet und mit einer Echtzeitschranke versehen (*semantische Widerspruchsfreiheit*).		X	• System- und Steuerungen-Interaktionsdiagramm • Komponenten-Verhaltensdiagramm	Wird die erste Nachricht zum Zeitpunkt t_1 und die zweite zum Zeitpunkt t_2 gesendet, wobei eine Zeitschranke Δt zwischen diesen Zeitpunkten spezifiziert ist, so gilt für das Komponenten-Verhaltensdiagramm, dass die Summe der Verzögerungszeiten, verursacht durch Zeit-Ereignisse (z.B. *after()*) und nicht-unterbrechbarer Aktivitäten, in allen Ausführungspfaden dieses Diagramms die Echtzeitschranke Δt nicht verletzen darf.

Tabelle 6.15: Varianten temporaler Spezifikation und die entsprechenden Definitionen der syntaktischen Konsitenz und semantischen Widerspruchsfreiheit (of.=offered, req.=required)

7 Anwendung und Validation der Methode zur Steuerungsentwicklung

Nach Abschluss der Entwicklung der ODEMA-Methode ist die Validation dieser Methode erforderlich. Dies soll anhand der Beschreibung eines beispielhaften Produktionssystems durchgeführt werden. Die so genannte *Referenzfallstudie Produktionstechnik* ist nach den Vorgaben der VDI/VDE-Richtlinie 3694 als Lastenheft beschrieben, ohne dass hierbei semiformale oder formale Beschreibungsmittel verwendet wurden (siehe Kapitel 10). Hierbei beschreibt die Grundstufe der Fallstudie die minimalen Anforderungen an das Produktionssystem und die so genannte Ausbaustufe die optionalen Funktionalitäten. Im Zusammenhang mit der Darstellung der Validation und Anwendung der ODEMA-Methode wird ausschließlich, wenn nicht explizit anders beschrieben, die Grundstufe berücksichtigt.

Es ergibt sich hieraus, dass zunächst zum Zweck der Validation der entwickelten Methode die funktionalen Anforderungen in die Darstellung des ODEMA-Beschreibungsmittels überführt werden, um daraus die Spezifikationen auf Basis der UML-Diagrammtyp-Rollen des Pflichtenheftes und des Objektorientierten Steuerungsentwurfs abzuleiten.

Die zur Bewertung von Spezifikationsmethoden eingeführten Methoden-Axiome werden auch zu Validation heran gezogen (siehe Kapitel 2.3). Tabelle 7.1 formuliert diese Axiome in Bezug auf die Anforderungen, die in der Fallstudie beschrieben sind.

Methoden-Axiom	Anforderung	Beschreibung aus der Fallstudie
Strukturprinzip	1	WZM, HTF, EIN und AUS verfügen jeweils über eine dezentrale Steuerung
	2	Dynamische Anzahl von HTF (Ausbaustufe)
System-Dekompositionsprinzip	3	Dezentrale Steuerungen sind durch ein Broadcast-Kommunikationssystem verbunden
	4	Beschreibung der Prozess- und Kommunikationsdaten (siehe Tabelle 10.15 und 10.16)
Software-Dekompositionsprinzip	5	Beschreibung der System-Schnittstellen (siehe Tabelle 10.17 bis 10.20)
Kausalprinzip	6	Beschreibung der Ablaufsteuerung (siehe Tabelle 10.13 und 10.14)
	7	Berücksichtigung einer Bedienersteuerung (Externes Leitsystem)
	8	Berücksichtigung der Steuerung von Nebenfunktionen (*Produktionsdatenbank*, siehe Kapitel 10.2.7)
	9	Verhandlungsbasierte Verteilung von Transportaufträgen (siehe Tabelle 10.13)
Temporalprinzip	10	Weiche Echtzeitschranken werden im Kontext der Maschinen-Funktionen beschrieben (siehe Tabelle 10.3, 10.8, 10.10 und 10.12)
	11	Anwendung der eingeführten Verfahren zur Prüfung der semantischen Widerspruchsfreiheit (siehe Tabelle 6.8, 6.9 und 6.15)
Projekt-Dekompositionsprinzip	12	Beschreibung des Produktionssystems bezüglich des Projektumfanges als Grund und Ausbaustufe
Anwendbarkeit	13	siehe Tabelle 2.1
Prüfbarkeit	14	Anwendung der bereits eingeführten Verfahren zur Prüfung auf syntaktische Konsistenz (siehe Tabelle 6.5, 6.7 und 6.14)

Tabelle 7.1: Methoden-Axiome im Kontext der *Referenzfallszudie Produktionstechnik*

7.1 Spezifikation des Lastenheftes der Fallstudie

Die Fallstudie beschreibt die Neuentwicklung eines Produktionssystems (*Werkstatt-orientiertes Fertigungssystem*) als PA-MAS ohne Berücksichtigung eines bestehenden Produktionssystems und ohne Einbeziehung einer entsprechenden Modell-Bibliothek zur Beschreibung des Lastenheftes (z.B. das *Agenten-Protokoll*).

7.1.1 Validation der Spezifikation *Geplantes Produktionssystem*

Informelle Grundlage der Spezifikation mittels System-Anwendungsfällen, der Systemarchitektur und den System-Interaktionsdiagrammen sind die Abschnitte *Aufgabenstellung* (siehe Kapitel 10.2.1) und *Ablaufbeschreibung* (siehe Kapitel 10.3.1 und 10.3.2) der Fallstudie. Das in der Ablaufbeschreibung dargestellte Szenario 8 des regulären Betriebs (siehe Tabelle 10.13) findet sich nicht in der Beschreibung der System-Anwendungsfälle wieder (siehe Bilder 7.1 und 7.2). Da die Funktion einer Werkzeugmaschine ohne jegliche Interaktion mit anderen Produktionsagenten des PA-MAS ausgeführt wird, existiert zwar eine entsprechende Rollen-Beschreibung (*Produzent*) für die Werkzeugmaschine, aber ein dazugehöriger System-Anwendungsfall ist nicht zu spezifizieren.

Die Beschreibungstechnik *Interner Akteur* erlaubt es, dass das Werkstatt-orientierte Fertigungssystem als PA-MAS beschrieben werden kann und mittels der Beschreibungstechnik *Agenten-Gruppe* ist es möglich, die dynamische Anzahl der HTF zu spezifizieren (z.B. *Bieter:HTS*, siehe Bild 7.1). Anforderung 1 und 2 bezüglich der Validation der ODEMA-Methode sind somit erfüllt (siehe Tabelle 7.1). Besonders zu bemerken ist, dass für die Erweiterungsabhängigkeit zwischen den System-Anwendungsfällen *Verhandlung der Transportaufträge* und *Beendigung der Fertigung* keine entsprechende Bedingung für einen Rollen-Wechsel spezifiziert ist. Da die Rolle *Bieter* aus der Rolle *Abmelder* abgeleitet wird und der Agententyp *AUS* die Rolle *Lagerist* nach der Initialisierung durchgängig einnimmt, gibt es keinen bedingten Rollenwechsel zwischen diesen System-Anwendungsfällen (siehe Bild 7.2).

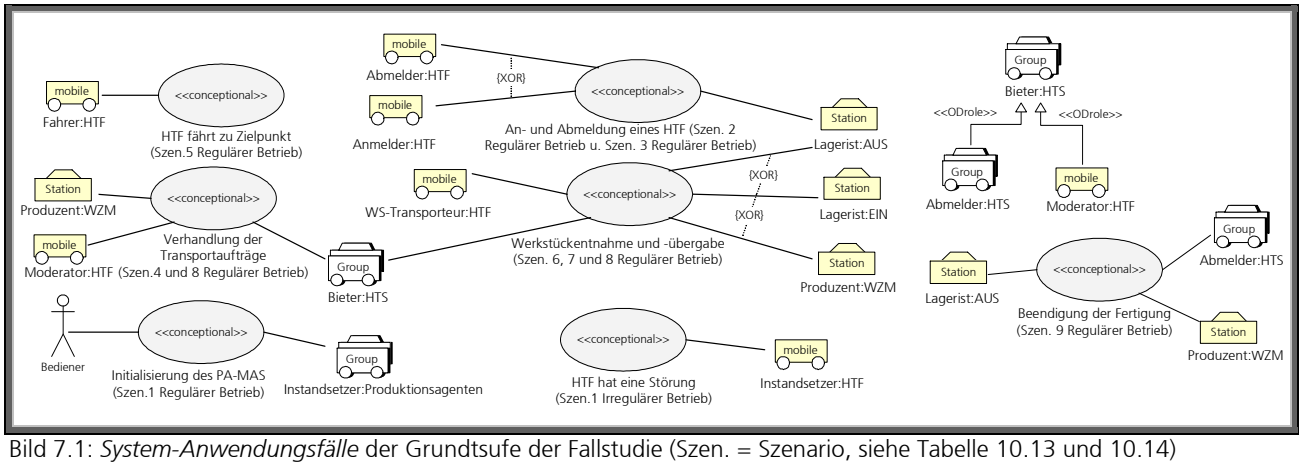

Bild 7.1: *System-Anwendungsfälle* der Grundtsufe der Fallstudie (Szen. = Szenario, siehe Tabelle 10.13 und 10.14)

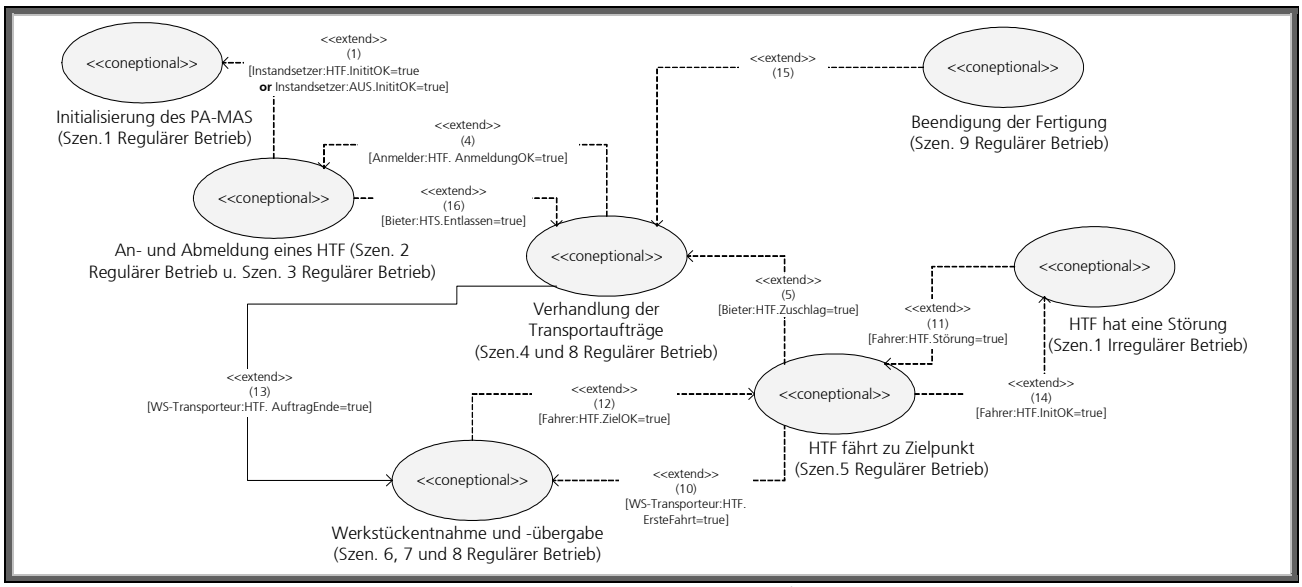

Bild 7.2 : Spezifikation der *Systemarchitektur* der Fallstudie (Szen. = Szenario)

Die Verhandlungs-basierte Verteilung von Transportaufträgen ist als System-Interaktionsdiagramm in Bild 7.4 dargestellt (Gehört zum System-Anwendungsfall *Verhandlung der Transportaufträge*). Die Instanz der Agenten-Gruppe *dasHTS* ermöglicht es hierbei zum einen die Broadcast-Kommunikationsverbindung darzustellen. Zum anderen ist in Verbindung mit dem Rollenwechsel die Spezifikation der Ernennung eines Moderators möglich. Die Beschreibung der Rolle *Moderator* als eine aus der Rolle *Bieter* abgeleitete Rolle ermöglicht es darüber hinaus, dass ein Moderator mit allen HTF kommunizieren kann, unabhängig davon, ob sie sich in der Rolle *Bieter* oder *Moderator* befinden. Dies ist notwendig, da von identischen HTF ausgegangen werden muss und daher von der Möglichkeit, dass mehr als ein HTF gleichzeitig einen Auftrag erkennt und sich zum Moderator der Verhandlung ernennt, bevor das Verhandlungsprotokoll mit Hilfe des Zeitstempels dafür sorgt, dass ein einziger Moderator übrig bleibt und die Verhandlung leiten kann. Anforderung 3 und 9 aus Tabelle 7.1 werden hiermit erfüllt.

Aus den hier dargestellten System-Interaktiondiagrammen lassen sich die drei verschiedene Varianten der nachrichtenbasierten Kommunikation aufzeigen. Tauschen zwei bekannte, d.h. mit einem Namen spezifizierte Instanzen (z.B. *dasAUS* in Bild 7.3) Nachrichten aus, so wird von einer Zwei-Punkt-Kommunikationsverbindung ausgegangen. Ist eine Instanz oder sind beide Instanzen anonym, so wird *impliziter Broadcast* angenommen und eine Sender- bzw. eine Empfänger-Identifikation müssen mit der Nachricht übersendet werden (z.B. der Parameter *meine ID* als Sender-Identifikation, siehe Bild 7.3 und 7.4). Diese Art der Identifikation von Sender und Empfänger einer Nachricht ist durch das Kommunikationsdatum *Maschinenkennung* auch von der Referenzfallstudie gefordert (siehe Tabelle 10.15). Die dritte Variante ist der *explizite Broadcast* mittels Agenten- bzw. Steuerungen-Gruppen. Die Konsequenzen bezüglich der Spezifikation von Kommunikationskanälen wurden bereits in Kapitel 6.2.2.1 dargestellt.

Eine weitere Besonderheit ist im System-Interaktionsdiagramm des System-Anwendungsfalls *Werkstückübergabe und –entnahme* beschrieben (siehe Bild 7.3). Die Spezifikation der Exklusiv-Oder-

Einschränkung im Kontext dieses System-Anwendungsfalls wird mittels einer bedingten Interaktion (Bedingung *Station*) im entsprechenden System-Interaktionsdiagramm berücksichtigt. Auf diese Weise kann die Anzahl der zu spezifizieren System-Interaktionsdiagramme reduziert werden.

Die temporalen Spezifikationen, wie durch Anforderung 10 in Tabelle 7.1 formuliert, werden im Kontext der System-Interaktionsdiagramme durch die entsprechenden zeitbezogenen Ausdrücken spezifiziert. Hierdurch lassen sich alle temporalen Spezifikationen (*offered* oder *required QoS*) bezüglich der zu spezifizierenden funktionalen Anforderungen berücksichtigen (siehe Bild 7.3 und 7.4).

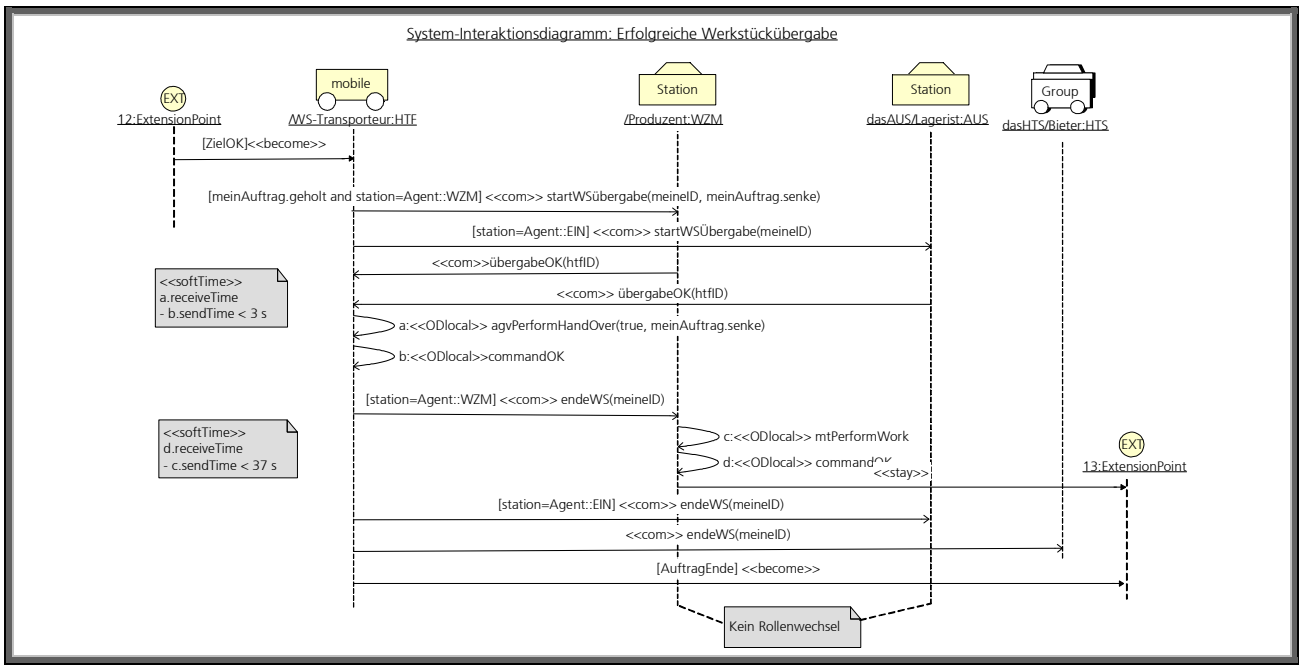

Bild 7.3: Erstes Beispiel für ein *System-Interaktionsdiagramm* der Fallstudie (Anwendungsfall *Werkstückentnahme und –übergabe*)

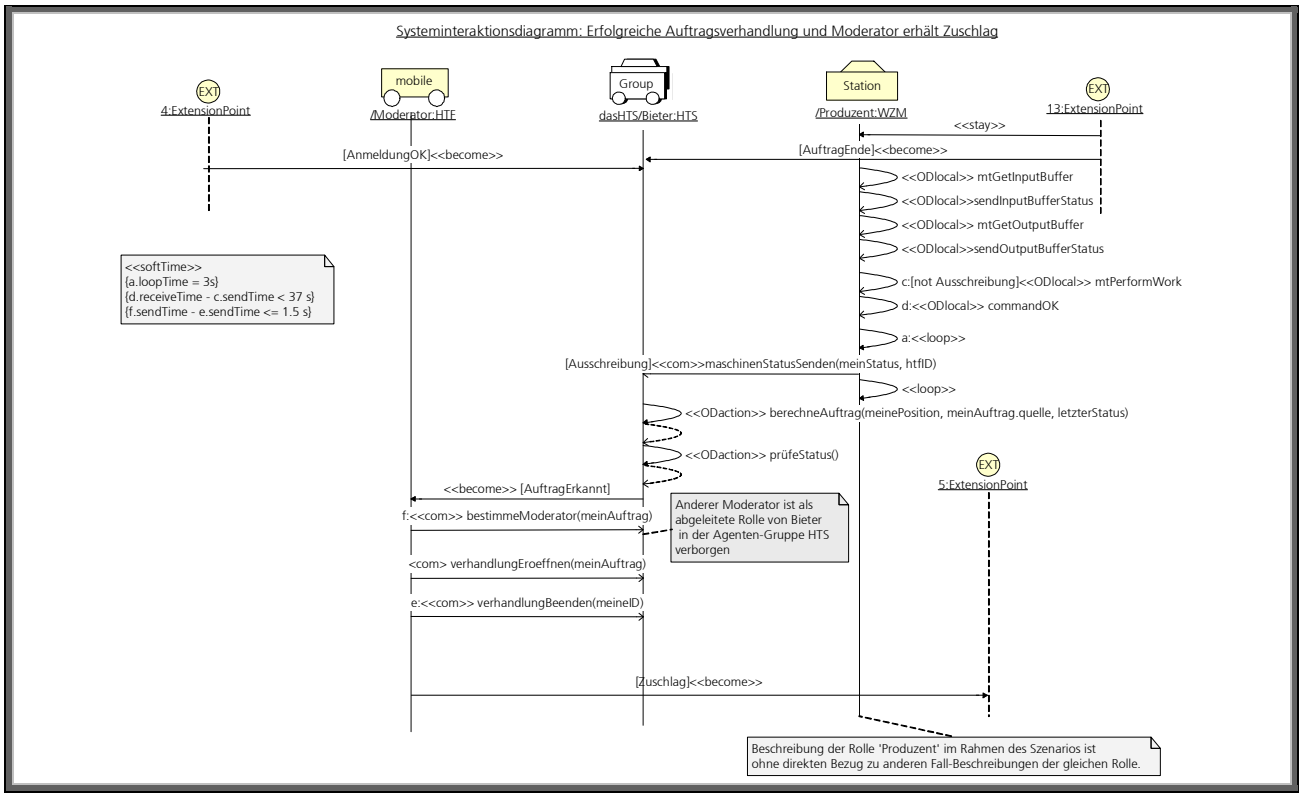

Bild 7.4: Zweites Beispiel für ein *System-Interaktionsdiagramm* der Fallstudie (Anwendungsfall *Verhandlung der Transportaufträge*)

7.1.2 Validation der Spezifikation des zukünftigen Produktionssystems

Das Lastenheft gemäß der VDI/VDE-Richtlinie 3694 sieht im Kontext der Aufgabenbeschreibung eines geplanten Produktionssystems die Beschreibung zukünftig geplanter Funktionen vor (Abschnitt 3.5 der VDI/VDE-Richtlinie). Hierzu werden die gleichen UML-Diagrammtyp-Rollen verwendet, die auch zur Spezifikation des geplanten Produktionssystems zum Einsatz kommen (siehe Bild 5.19).

Die Fallstudie beschreibt eine Ausbaustufe des Werkstatt-orientierten Fertigungssystems, die als zukünftig zu planendes Produktionssystem zu verstehen ist. Die Szenarien 2 bis 4 des Abschnitts Irregulärer Betrieb (siehe Kapitel 10.3.2) sind Teil dieser Ausbaustufe und in Bild 7.5 in Form erweiterter oder neu spezifizierter System-Anwendungsfälle beschrieben (Erfüllung der Anforderung 12 aus Tabelle 7.1).

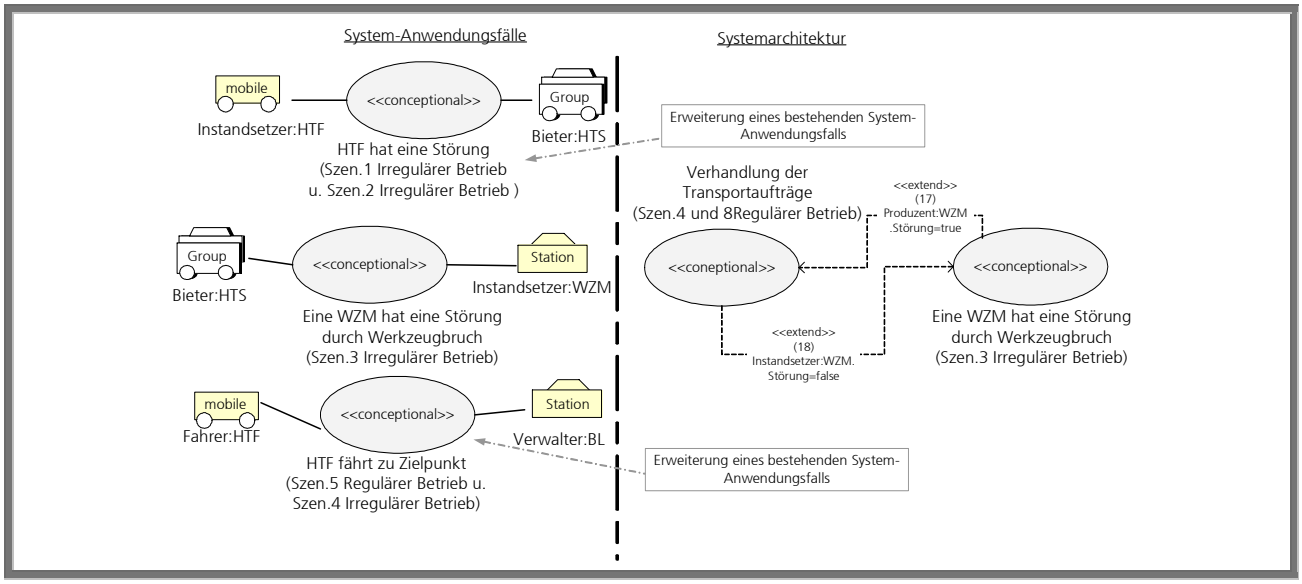

Bild 7.5 : Erweiterte System-Anwendungsfälle und Systemarchitektur des geplanten PA-MAS der Fallstudie (Ausbaustufe)

7.1.3 Validation der Spezifikation der Schnittstellen der Fallstudie

Die Typen der Produktionsagenten HTF, EIN, AUS und WZM werden als UML-Funktionsblöcke abgebildet. Die Rollen-Schnittstellen (z.B. *WS-Transporteur* und *Produzent*) sind bereits im Rahmen der System-Anwendungsfälle und System-Interaktionsdiagramme spezifiziert worden, wobei auf die Darstellung des Agenten-Dekompositionsdiagramms verzichtet wurde (siehe Bilder 7.1 bis 7.4). Die Schnittstellen-Klassen, die den Zugriff auf die lokalen Steuerungen der Produktionsagenten ermöglichen (z.B. die Schnittstellen-Klasse *LokaleSteuerungHTF*) müssen nicht mittels eines Agenten-Dekompositionsdiagramms hergeleitet werden, weil sie durch die Fallstudie bereits beschrieben sind und als bestehende Implementierung angesehen werden können (siehe Kapitel 10.5).

Die Beschreibung der Fallstudie stellt für bestimmte Funktionen der lokalen Steuerung synchrone sowie asynchrone Kommandos zur Verfügung (z.B. das asynchrone Kommando a*gvStartMoveAbs* und das synchrone *agvPerformMoveAbs,* siehe Tabelle 10.5.1). Zur Spezifikation der Rollen-Schnittstelle von UML-Funktionsblöcken werden gemäß der Konzeption der nachrichtenbasierten Kommunikation ausschließlich asynchrone Kommandos verwandt. Dort, wo die Nachrichten mit Namen von synchro-

nen Kommandos deklariert sind, sind diese als Nachrichten zu verstehen, die eine weitere Nachricht als Rückantwort erst nach Ausführung des gewünschten Kommandos erwarten. Diese Rückantwort, z.B. bezüglich der erfolgreichen oder fehlerhaften Ausführung eines Kommandos, wird wiederum als vertikale Nachricht (*commandOK()* bzw. *commandFailure()*) im Kontext der entsprechenden Schnittstellen-Klasse spezifiziert (z.B. *Fahrer*, siehe Bild 7.7). Die Datentypen (Prozess- und Kommunikationsdaten, siehe Kapitel 10.4), die für die Fallstudie spezifiziert sind, sind als Daten-Bibliothek in Bild 7.8 dargestellt. Die Axiome *System*- und *Software-Dekomposition* (Anforderung 4 und 5 aus Tabelle 7.1) werden aus Sicht der Fallstudie auf diese Weise erfüllt.

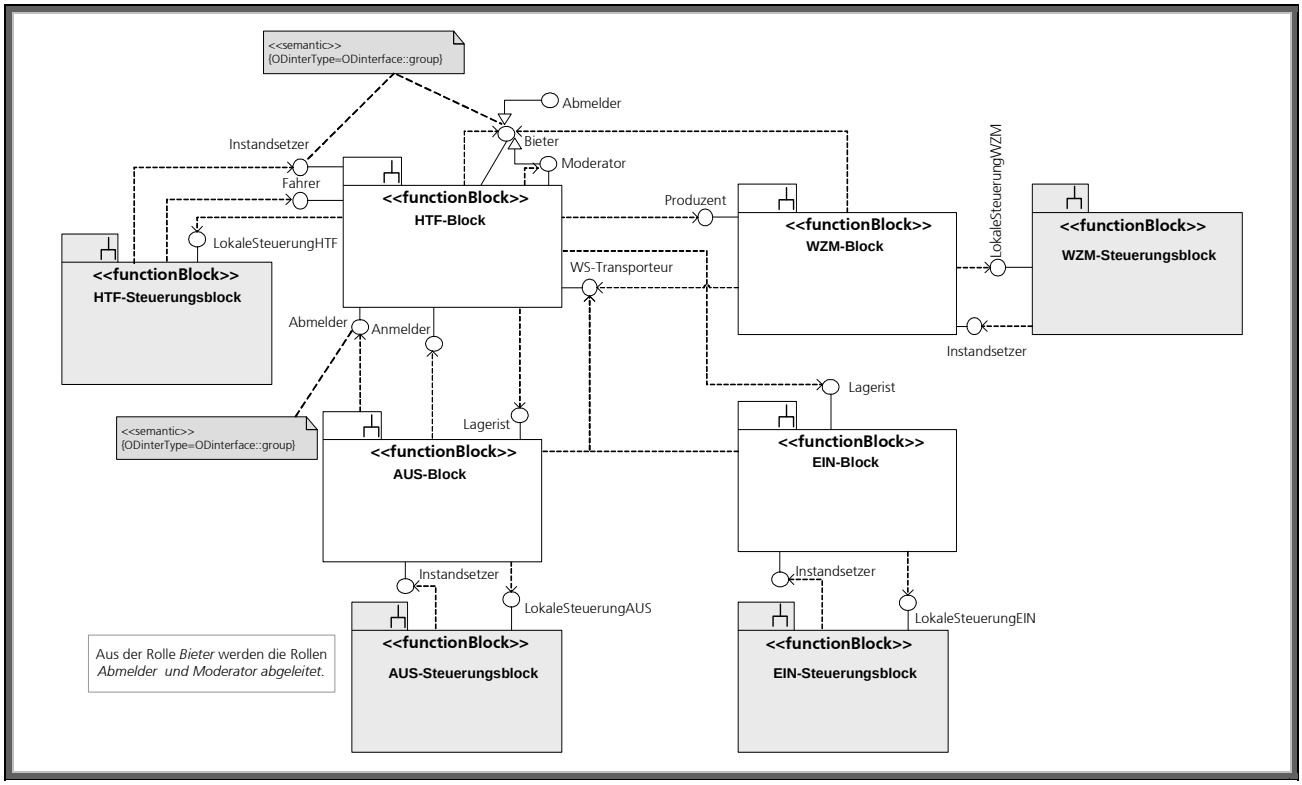

Bild 7.6: Darstellung der *PF-Funktionsblockarchitektur* der Fallstudie

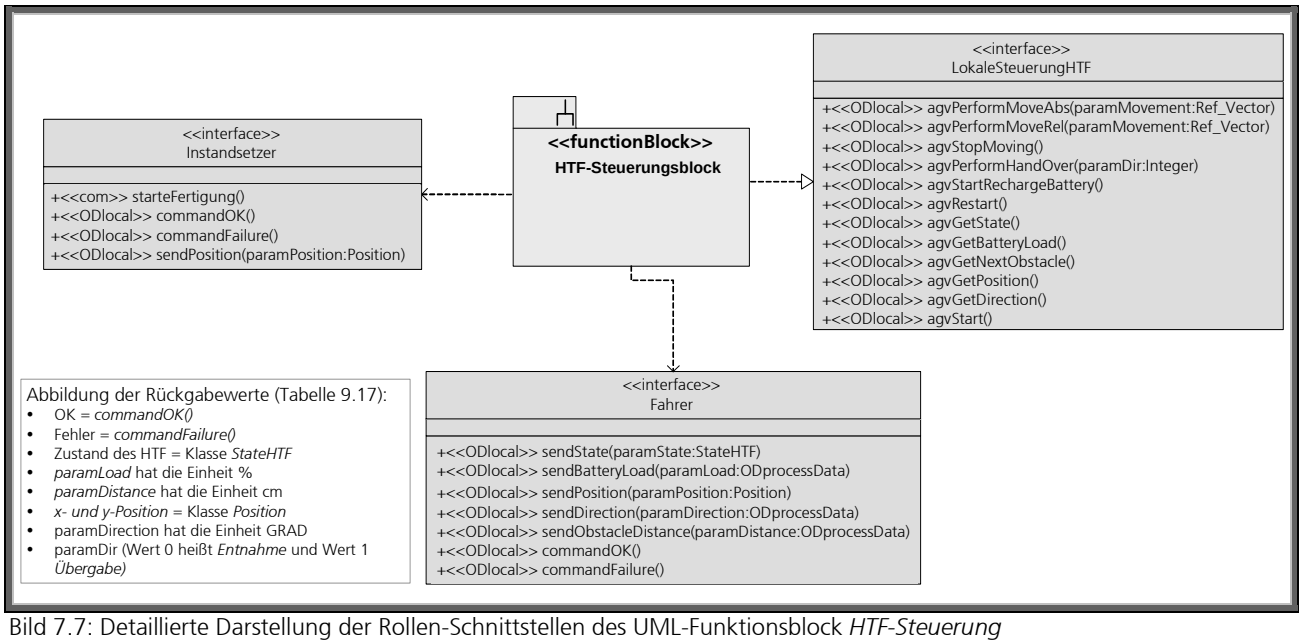

Bild 7.7: Detaillierte Darstellung der Rollen-Schnittstellen des UML-Funktionsblock *HTF-Steuerung*

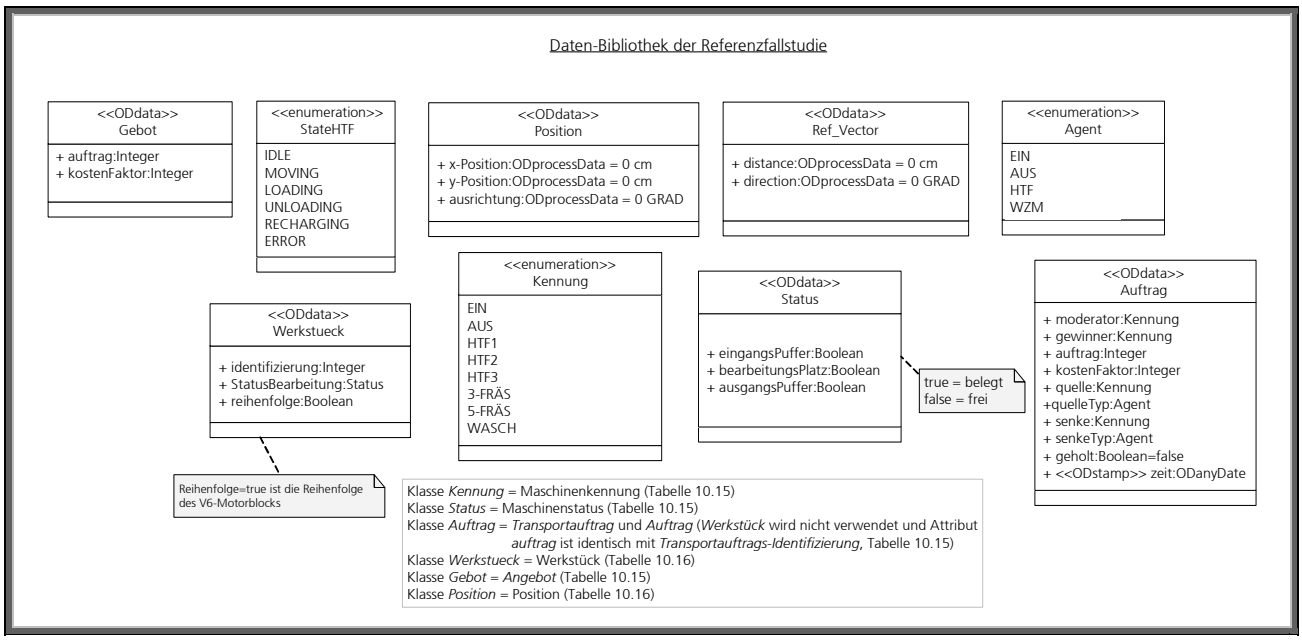

Bild 7.8: Darstellung der *Daten-Bibliothek* der Fallstudie

7.1.4 Anwendung der *Prüfung der funktionalen Anforderungen*

Das in Tabelle 6.5 beschriebenen Verfahren *Prüfung der funktionalen Anforderungen* lässt sich anhand der in diesem Kapitel dargestellten Spezifikationen (siehe Bilder 7.1 bis 7.8) der Fallstudie anwenden (Anforderung 14, siehe Tabelle 7.1).

Es kann unmittelbar durch Anschauung der erweiterten UML-Diagramme geprüft werden, dass die Erweiterungspunkte (*System-Interaktionsdiagramm*, siehe Bild 7.3 und 7.4) eine Untermenge der entsprechenden Spezifikationen aus der Systemarchitektur sind (siehe Bild 7.2). Hierbei enthält das System-Interaktionsdiagramm, das die Werkstückübergabe beschreibt, die Erweiterungspunkte 12 und 13, die ebenfalls im Kontext der Erweiterungsabhängigkeiten spezifiziert wurden und mit dem dazugehörigen Anwendungsfall *Werkstückentnahme und –übergabe* verbunden sind. Die gleiche Prüfung auf syntaktische Konsistenz kann auch für die Instanzen der internen Akteure durchgeführt werden (UML-Diagrammtyp-Rolle *Systemszenarien* ist nicht dargestellt).

Für die internen Akteure und ihrer Rollen kann anhand der PF-Funktionsblockarchitektur die Anwendung der Prüfung der Dekomposition der Rollen bezüglich Rollen- und Gruppenschnittstellen gezeigt werden (siehe Bild 7.1 und 7.6). Im Kontext der System-Anwendungsfälle wird spezifiziert, dass der stationäre Produktionsagent AUS die Rolle *Lagerist* einnimmt. Da dieser auch zu der Agenten-Gruppe *Produktionsagenten* gehört (ein entsprechendes *Agenten-Diagramm* ist nicht dargestellt), kann AUS zusätzlich die Rolle *Instandsetzer* einnehmen. Die entsprechenden Rollen-Schnittstellen finden sich in der Beschreibung der Funktionsblockarchitektur als die vom UML-Funktionsblock *AUS-Block* exportierten Schnittstellen-Klassen wieder, wobei die Rolle *Instandsetzer* durch den Eigenschaftswert *ODinter-Type* als Gruppen-Schnittstelle gekennzeichnet ist. Weithin können die horizontalen und vertikalen Nachrichten auf ihre korrekte Deklaration hin überprüft werden. So findet sich die vertikale *Nachricht agvPerformHandOver* auch als Operationsdeklaration auf der Schnittstellen-Klasse *LokaleSteue-*

rungHTF wieder, die von dem UML-Funktionsblock *HTF-Steuerungsblock* exportiert wird. Hierbei ist zu beachten, dass im Kontext eines Sequenzdiagramms ein Parameter einer Nachricht als Wert oder als Objekt bzw. im Verständnis der ODEMA-Methode als Kommunikationsobjekt angegeben wird. Der Name eines solchen Kommunikationsobjektes ist dem Namensraum der sendenden Instanz entnommen und muss beim Empfänger nicht identisch bezeichnet werden.

Die Daten-Bibliothek beschreibt die direkte Umsetzung der Kommunikations- und Prozessdaten aus den Tabellen 10.15 und 10.16 (Anforderung 4 aus Tabelle 7.1, siehe Bild 7.8). Anhand der Daten-Klasse *Auftrag* wird deutlich, dass die Spezifikation über das hinaus gehen kann, was das Lastenheft fordert. Der Parameter *geholt* der Daten-Klasse *Auftrag* z.B. ist nicht im Lastenheft beschrieben und ergibt sich erst durch die Beschreibung des Komponenten-Verhaltensdiagramms des UML-Funktionsblocks *HTF-Block*. Hierbei wird die Klasse *Auftrag* nicht nur zur Beschreibung eines zu verhandelnden Transportauftrages herangezogen, sondern auch zur Speicherung des Verlaufes der Abarbeitung eines Transportauftrages (siehe auch Bild 7.16)

7.2 Spezifikation des Pflichtenheftes der Fallstudie

Das Pflichtenheft der Fallstudie verringert sich bezüglich der Komplexität der Spezifikation dadurch, dass im Lastenheft die UML-Funktionsblöcke der Prozesssteuerungsebene als bereits implementiert beschrieben werden und ihre Schnittstellen-Beschreibungen gegeben sind (siehe Tabelle 10.17 bis 10.20).

7.2.1 Validation der Spezifikation der *Systemtechnische Lösung*

Eine Vervollständigung der PF-Funktionsblockarchitektur der Fallstudie (siehe Bild 7.6) ist aufgrund der verringerten Komplexität des Pflichtenheftes nicht notwendig. Die UML-Funktionsblöcke der Prozesssteuerungsebene sind als eine geschlossene Komponenten-Implementierung mit entsprechender Schnittstellen-Beschreibung zu spezifizieren und entsprechen somit einer wieder zu verwenden Softwarekomponenten-Implementierung (siehe Kapitel 6.2.2).

Das ausführbare Programm *HTF-Rumpf-Steuerung* z.B. beschreibt in diesem Kontext die Steuerung des PA-Rumpfes eines HTF. Dies gilt auch für die Rumpf-Steuerungen der Produktionsagenten WZM, EIN und AUS. Somit entfällt ebenso die separate Spezifikation der PS-Funktionsblockarchitektur. Darüber hinaus sind die PF-Komponentenarchitekturen der Fallstudie als trivial anzusehen, da sie lediglich aus einer aktiven Steuerungskomponente bestehen. Lediglich der *HTF-Block* verfügt zusätzlich über eine weitere passive Steuerungskomponente (siehe Bild 7.9).

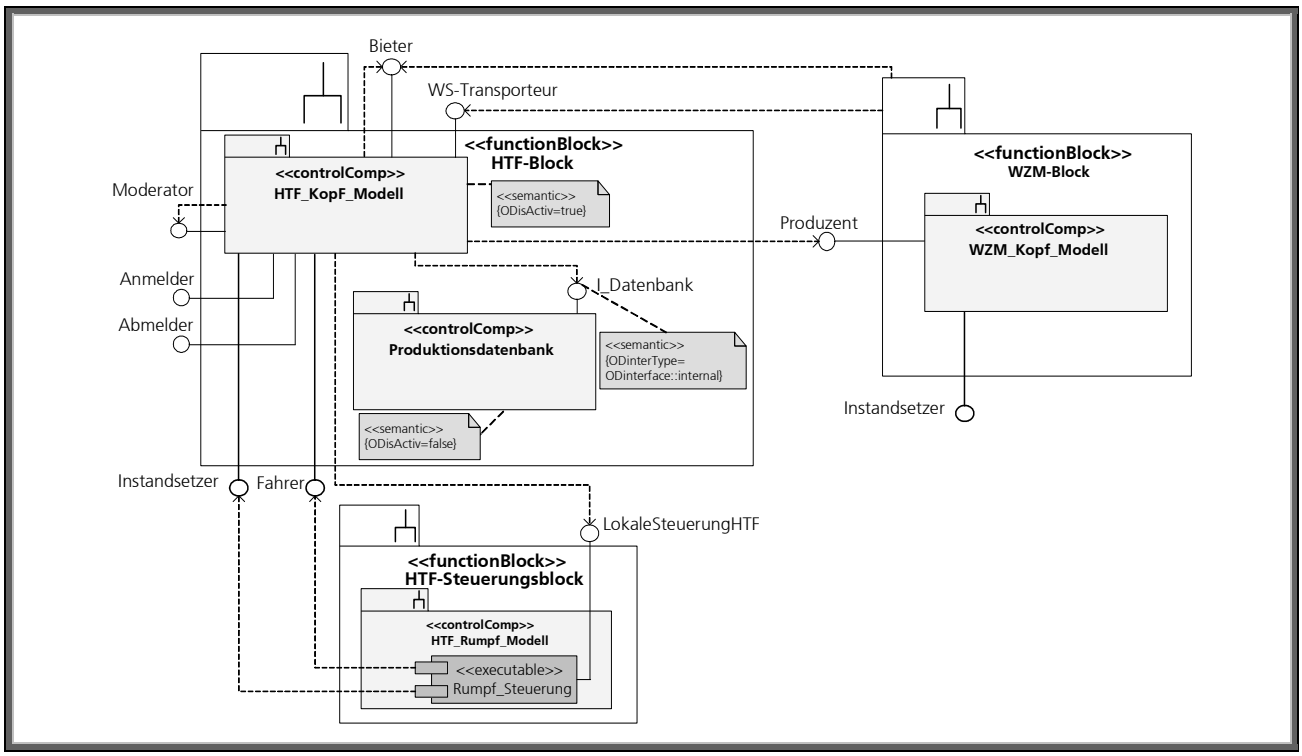

Bild 7.9: Ausschnitt der PF-Komponentenarchitektur der Fallstudie

Die nach dem Lastenheft geforderte Produktionsdatenbank für das HTF wird durch die passive Steuerungskomponente *Produktionsdatenbank* und die internen Schnittstellen-Klasse *I_Datenbank* umgesetzt. Mit dem Abschluss der PF-Komponentenarchitektur wird die Forderung nach Beschreibung der Software-Dekomposition gemäß der Anforderung 5 aus Tabelle 7.1 erfüllt.

7.2.2 Validation der Spezifikation der *Systemtechnik*

Da jede passive oder aktive Steuerungskomponente genau einer UML-Komponente in der Implementierungssicht der UML entspricht, lässt sich das PF-Implementierungsdiagramm der Fallstudie direkt aus der PF-Komponentenarchitektur ableiten. Aus diesem Grund steht die Spezifikation der Kommunikationskanäle im Mittelpunkt der Entwurfsentscheidungen des Entwicklers.

Auf die Darstellung eines separaten PS-Implementierungsdiagramms kann aufgrund der Trivialität verzichtet werden und somit zeigt Bild 7.10 das PF-Implementierungsdiagramm als zusammenfassende Beschreibung aller Programm-Bibliotheken und ausführbaren Programme. Beschreibende Implementierungen sind aus Gründen der Übersichtlichkeit ausschließlich im Installationsdiagramm dargestellt (siehe Bild 7.12).

Da die Fallstudie ausschließlich weiche Echtzeitanforderungen spezifiziert, ist das Entwurfsziel bezüglich der Auslegung der Kommunikationskanäle zum einen, dass möglichst viele Schnittstellen-Klassen zu einem Kommunikationskanal zusammengefasst werden, um die Komplexität des entsprechenden UML-Diagramms zu reduzieren (z.B. die Kommunikationskanäle *Kanal1*, *Kanal10* und *Kanal11* in Bild 7.10). Zum anderen ist zu berücksichtigen, dass auf die ausführbaren Programme und Kommunikationskanäle auch die unterschiedlichen Varianten der Kommunikation aus den System-Interaktionsdiagrammen einen Einfluss haben (siehe Kapitel 7.1.1 und Bild 7.10).

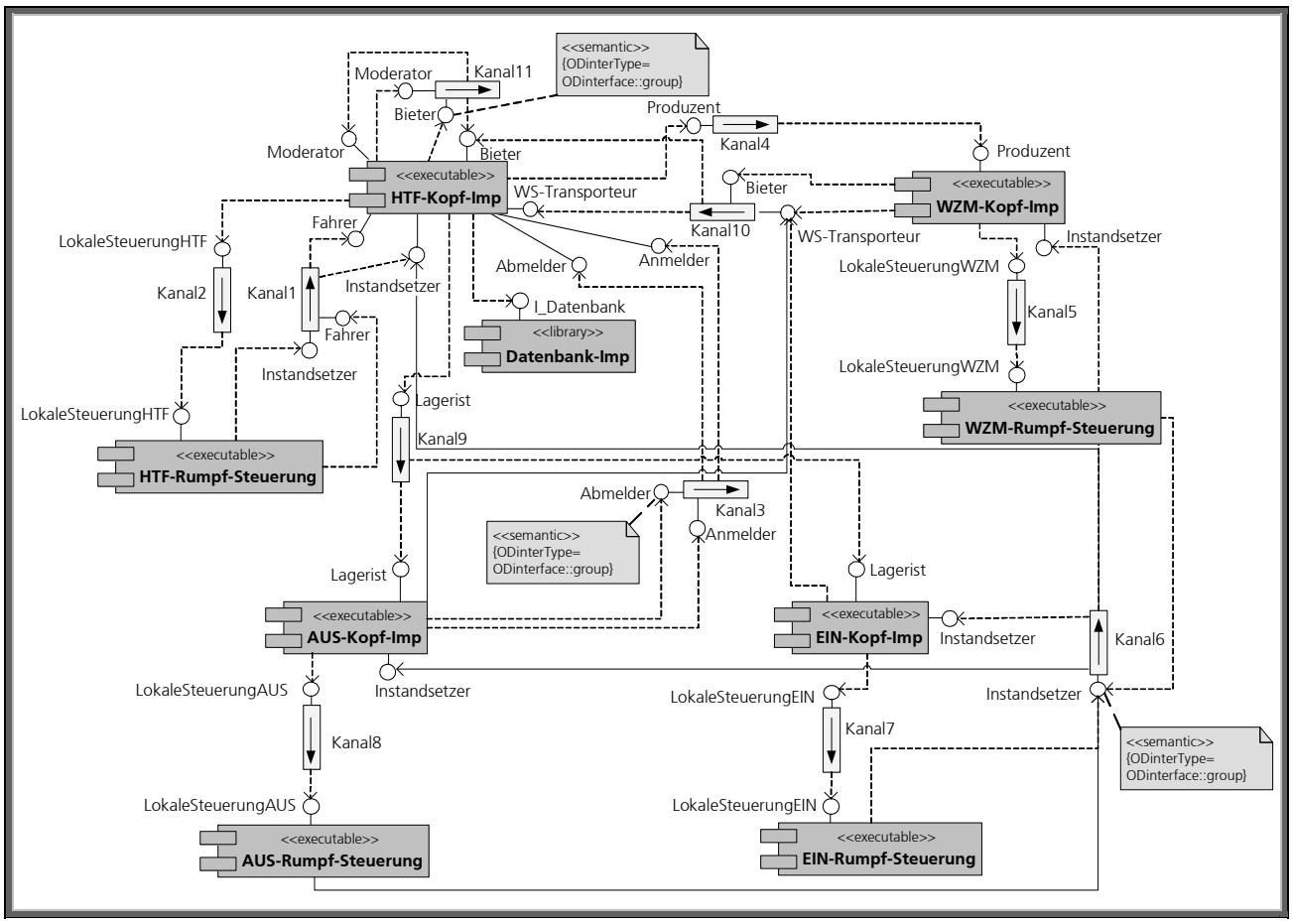

Bild 7.10: PF-Implementierungsdiagramm der Fallstudie ohne die Darstellung beschreibender Implementierungen (Ableitungen zwischen UML-Schnittstelllen-Klassen sind nicht dargestellt)

Die Produktionsagenten AUS und EIN sind als Instanzen in den System-Interaktionsdiagrammen explizit bezeichnet, da sie als Einrichtungen des Produktionssystems jeweils nur einmal vorgesehen sind. Aus diesem Grund wird der Kommunikationskanal *Kanal9* im Installationsdiagramm jeweils für einen der beiden Produktionsagenten instanziert (siehe Bild 7.12). Da *Knanal6* ebenfalls zweimal instanziert wird und zudem jeweils einem anderen lokalen Kommunikationssystem zugeordnet wird (äquivalent zum Installationsdiagramm von HTF-1, siehe Bild 7.12), werden die Instanzen von *AUS-Kopf-Imp* und *EIN-Kopf-Imp* direkt adressiert, so wie im System-Interaktionsdiagramm beschrieben.

Die Produktionsagenten HTF und WZM als Instanzen im System-Interaktionsdiagramm werden als anonyme Instanzen beschrieben. Bezüglich der Spezifikation der Kommunikationsverbindung mittels Kommunikationskanälen bedeutet dies, dass explizit Gruppen-Schnittstellen verwendet werden können, sofern entsprechende Agenten-Gruppen spezifiziert wurden (gilt z.B. für die Schnittstellen-Klasse *Abmelder*). Die zweite Möglichkeit ist der bereits eingeführte *implizite Broadcast* (siehe auch Kapitel 7.1.1). Diese UML-Beschreibungstechnik wird bezüglich der Fallstudie z.B. für die Schnittstellen-Klasse *Produzent* und dem dazugehörigen Kommunikationskanal *comWZM:Kanal4* angewandt. Die Spezifikation der Schnittstellen-Klasse *Produzent* weist daher für die entsprechenden Deklarationen der horizontalen Nachrichten entsprechende Sender- und Empfänger-Identifizierungen als Parameter auf (z.B. *startWSentnahme(paramSenderID:Kennung, paramEmpängerID:Kennung)*).

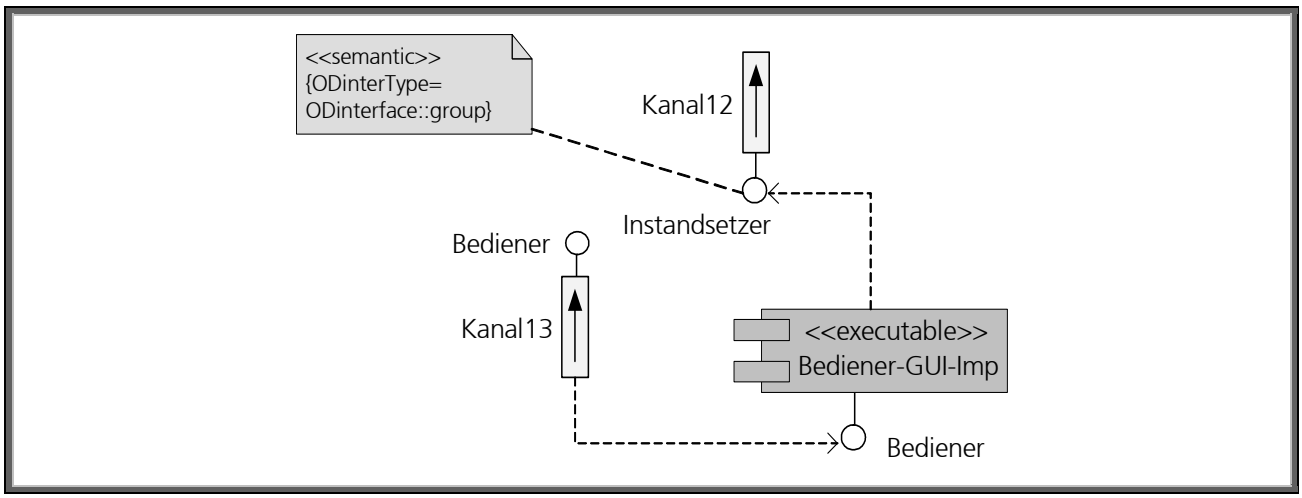

Bild 7.11: Einführung des Bedieners in das PF-Implementierungsdiagramm der Fallstudie

Die bisherige Beschreibung der Fallstudie hat den menschlichen Bediener als externen Akteur nicht ausreichend berücksichtigt. Externe Akteure können grundsätzlich auf die gleiche Weise dekomponiert werden wie interne Akteure. Bezüglich eines menschlichen Bedieners als externen Akteur steht hierbei allerdings bereits vor der Dekomposition fest, dass ausschließlich die Steuerungsfunktionen *Bedienersteuerung* zu implementieren ist (siehe Tabelle 2.2). Diese wird in der Regel mit speziellen Entwicklungswerkzeugen realisiert und der Entwickler hat hierbei lediglich die Aufgabe, die spezifizierten Schnittstellen in die Spezifikation der Steuerung einzubinden. Bild 7.11 zeigt die dazu gehörige Ergänzung des PF-Implementierungsdiagramms.

Im Kontext der UML hat ein externer Akteur nur eine Rolle, die sich bezüglich der Fallstudie als Schnittstellen-Klasse *Bediener* wieder findet (siehe auch Bild 7.1 System-Anwendungsfall *Initialisierung des PA-MAS*). Anforderung 7 aus Tabelle 7.1 wird hiermit erfüllt.

Nachteilig bezüglich der Anwendung des Beschreibungsmittels der ODEMA-Methode ist festzustellen, dass die gemeinsame Spezifikation von lokalen und horizontalen Nachrichten durch eine einzige Rollen- oder Gruppen-Schnittstelle zu einer umständlichen und unübersichtlichen Beschreibung der Gruppen-Schnittstelle *Instandsetzer* führt (Anforderung 13, siehe Tabelle 7.1). *Kanal1* überträgt die vertikalen Nachrichten dieser Gruppen-Schnittstelle und ist als Bestandteil der lokalen Kommunikation eines HTF zu verstehen (Mehr-Punkt-System *Bus1* in Bild 7.12).

Die anderen Kommunikationskanäle mit dieser Gruppen-Schnittstelle (*Kanal6* und *Kanal12*) verwenden *Instandsetzer* zur horizontalen Kommunikation zwischen den Produktionsagenten und nur hier wirkt diese Schnittstellen-Klasse als expliziter Broadcast (Gruppen-Schnittstelle).

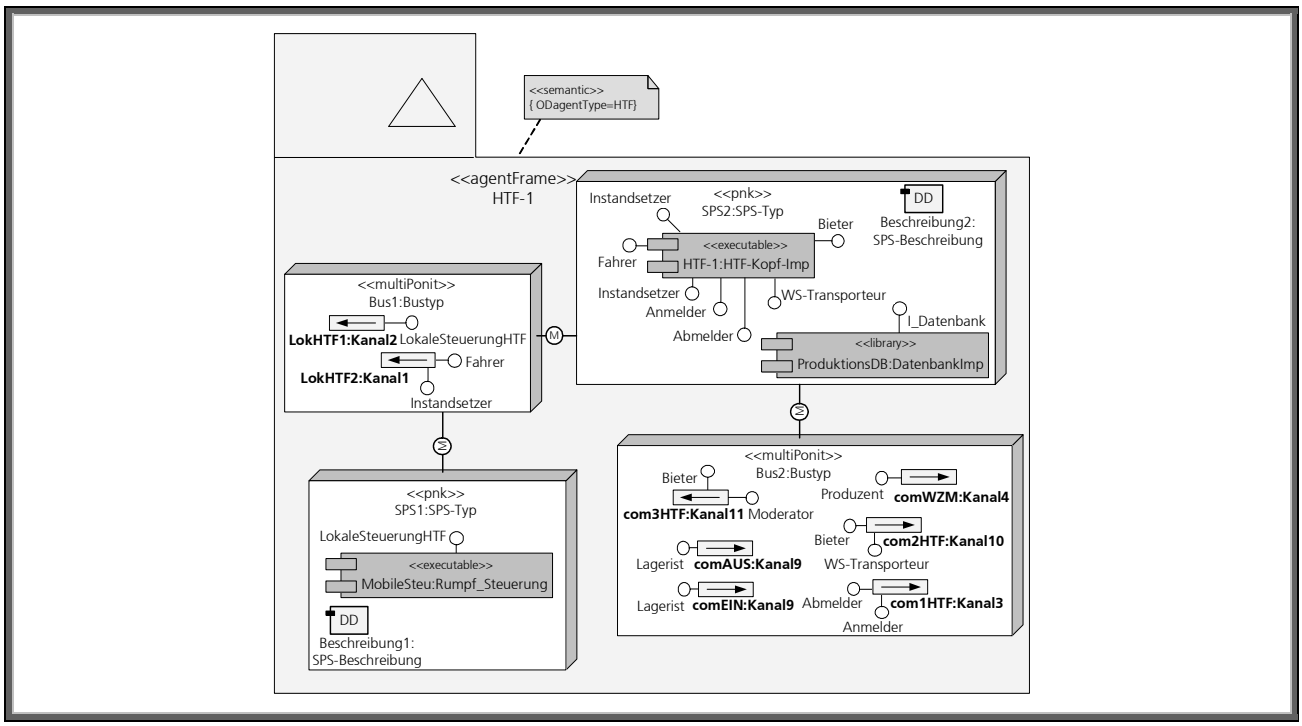

Bild 7.12: Installationsdiagramm des HTF-1 der Fallstudie (Darstellung ohne UML-Abhängigkeiten und ohne Komunikationskanal 12 und 13)

7.2.3 Anwendung der *Prüfung des Pflichtenheftes auf Konsistenz*

Das in Tabelle 6.7 beschriebenen Verfahren *Prüfung des Pflichtenheftes auf Konsistenz* lässt sich anhand der UML-Diagrammtyp-Rollen PF-*Komponentenarchitektur* (siehe Bild 7.9) *PF-Implementierungsdiagramm* (siehe Bild 7.10) und dem *Installationsdiagramm* (siehe Bild 7.12) anwenden.

Da im Kontext der Spezifikation des Lastenheftes keinerlei Sammel-Schnittstellen zum Einsatz kommen ist die Dekomposition dieser Schnittstellen-Klassen und damit der entsprechenden UML-Funktionsblöcke nicht durchzuführen. Aufgrund der Trivialität der PF-Komponentenarchitektur ist lediglich die syntaktische Prüfung bezüglich der Kommunikationskanäle und ihrer Schnittstellen-Klassen durchzuführen. Dies kann ohne weiteres anhand des PF-Implementierungsdiagramms geschehen (Bild 7.10). Unübersichtlich ist in diesem Zusammenhang aber die Instanzierung der UML-Komponenten im Installationsdiagramm. In Bild 7.12 wurde aus diesem Grund auf die Darstellung der Abhängigkeiten verzichtet. Allerdings geht hierbei ein Teil der Information bezüglich der Abhängigkeiten zwischen den Instanzen von Kommunikationskanälen und der von ausführbaren Programmen verloren. Das Installationsdiagramm des *HTF-1* zeigt zwei Instanzen des Kommunikationskanals *Kanal9*. Aus dem PF-Implementierungsdiagramm geht aber nicht eindeutig hervor, welche Instanzen ausführbarer Programme mit diesem Kommunikationskanal verbunden sind. Aus diesem Grund erfährt die Anforderung 13 der Tabelle 7.1 eine Einschränkung bezüglich ihrer Erfüllung.

Eine wesentliche Erweiterung des Verfahrens *Prüfung des Pflichtenheftes auf Konsistenz* gegenüber dem, welches bezüglich des Lastenheftes zum Einsatz kommt, ist, dass temporale Spezifikationen berücksichtigt werden (Anforderung 11 aus Tabelle 7.1). Die in Tabelle 6.8 beschriebene Definitionen der

syntaktischen Konsistenz sind in den hier dargestellten System-Interaktionsdiagrammen unmittelbar durchzuführen, da keine verschachtelten temporalen Spezifikationen vorliegen (siehe Bild 7.3 und 7.4). Die Anwendung der Prüfung auf semantischen Widerspruchsfreiheit, wie sie in Tabelle 6.9 dargestellt ist, wird exemplarisch anhand der Fallstudie in Tabelle 7.2 dargestellt.

<table>
<tr><td colspan="3">Beispielrechnung (siehe Bild 7.4): Zwischen den Marken f und e ist die Verhandlungsdauer von durchschnittlich 1.5 s spezifiziert (siehe auch Tabelle 10.12). Die durchschnittliche Verhandlungsdauer wurde im System-Interaktionsdiagramm als weiche Echtzeitschranke formuliert. Aus dem PF-Implementierungsdiagramm (siehe Bild 7.10) und dem Installationsdiagramm (siehe Bild 7.12) geht hervor, dass der hierfür relevante Kommunikationskanal der com3HTF:Kanal11 ist (BeschreibungKanal11.asyncDelay). Hierbei werden zwei Fallunterscheidungen getroffen.
Es gilt nach Gleichung 4 (Tabelle 6.9): $\mathbf{t_{Verhandlung} \geq 2 * t_{V,Kanal11} + t_{v,Bieter}}$</td></tr>
<tr><td>Fall 1: Es wird angenommen, dass der Steuerungsprozess HTF in der Rolle Bieter bzw. Moderator zur Feststellung seines Kostenfaktors im Vergleich zur Verhandlungsdauer keine Zeit benötigt (siehe auch Bild 7.16).</td><td>$t_{V,\,Kanal11} \leq (t_{Verhandlung} - t_{v,Bieter}) / 2$
$t_{V,\,Kanal11} \leq 0.75$ s</td><td rowspan="2">$t_{V,Kanal11}$ wird dem Attribut asyncDelay der entsprechenden Instanz der beschreibenden Klasse BeschreibungKanal11 zugewiesen.</td></tr>
<tr><td>Fall 2 (erweitert): Um den HTF, die einen Transportauftrag gerade beenden, die Möglichkeit zu geben an einer laufenden Verhandlung noch teilzunehmen, soll den entsprechenden Steuerungsprozessen im Durchschnitt eine Antwortzeit von 1s garantiert zugestanden werden.</td><td>$t_{V,\,Kanal11} \leq (t_{Verhandlung} - t_{v,Bieter}) / 2$
$t_{V,\,Kanal11} \leq 0.25$ s</td></tr>
</table>

Tabelle 7.2: Anwendung der Prüfung auf semantische Widerspruchsfreiheit bezüglich des Pflichtenheftes der Fallstudie

7.3 Spezifikation des Objektorientierten Steuerungsentwurfs der Fallstudie

7.3.1 Validation der Spezifikation von Komponenten-Spezifikation und -Verhaltensdiagramm

Die Validation des objektorientierten Steuerungsentwurfs erfolgt anhand der Komponenten-Spezifikation (siehe Bild 7.10) und dem Komponenten-Verhaltensdiagramm (siehe Bild 7.11) der Steuerungskomponente *HTF_Kopf_Modell*.

Signifikant für die Komponenten-Spezifikation ist der hohe Anteil an generischen UML-Beschreibungselementen. Alle vertikalen und horizontalen Nachrichten, die von der Verhaltensklasse spezifiziert werden, werden durch die exportierten Schnittstellen-Klassen definiert. Atomare Aktionen und nicht-unterbrechbare Aktivitäten werden bereits durch das Agenten-Dekompositionsdiagramm vorgegeben (siehe Kapitel 6.1.3.1). Auch die zu importierenden Schnittstellen-Klassen sind mit der Beschreibung der PF-Komponentenarchitektur vollständig bekannt. Dem Entwickler verbleibt die Möglichkeit mittels der Beschreibung von Attribute bezüglich der Verhaltensklasse und Verknüpfung dieser Klasse mit entsprechenden Daten-Klassen eigene Entwurfsentscheidungen zu spezifizieren. Darüber hinaus lässt die Definition der Stellvertreter-Klasse die Möglichkeit offen, auf welche Weise andere aktive Steuerungskomponenten angesprochen werden. In der in Bild 7.13 dargestellten Komponenten-Spezifikation sind die Objekte *dasEIN* und *dasAUS* durch eine gemeinsame Stellvertreter-Klasse beschrieben. Eine andere Lösung wäre die Beschreibung mittels zweier separater Stellvertreter-Klassen. Da aber die Kopfsteuerungen des AUS und des EIN durch die gleiche Schnittstellen-Klasse *Lagerist* mit

der Kopfsteuerung eines HTF verbunden sind, ist die vereinfachende Beschreibung durch eine gemeinsame Stellvertreter-Klasse die bessere Beschreibungsvariante.

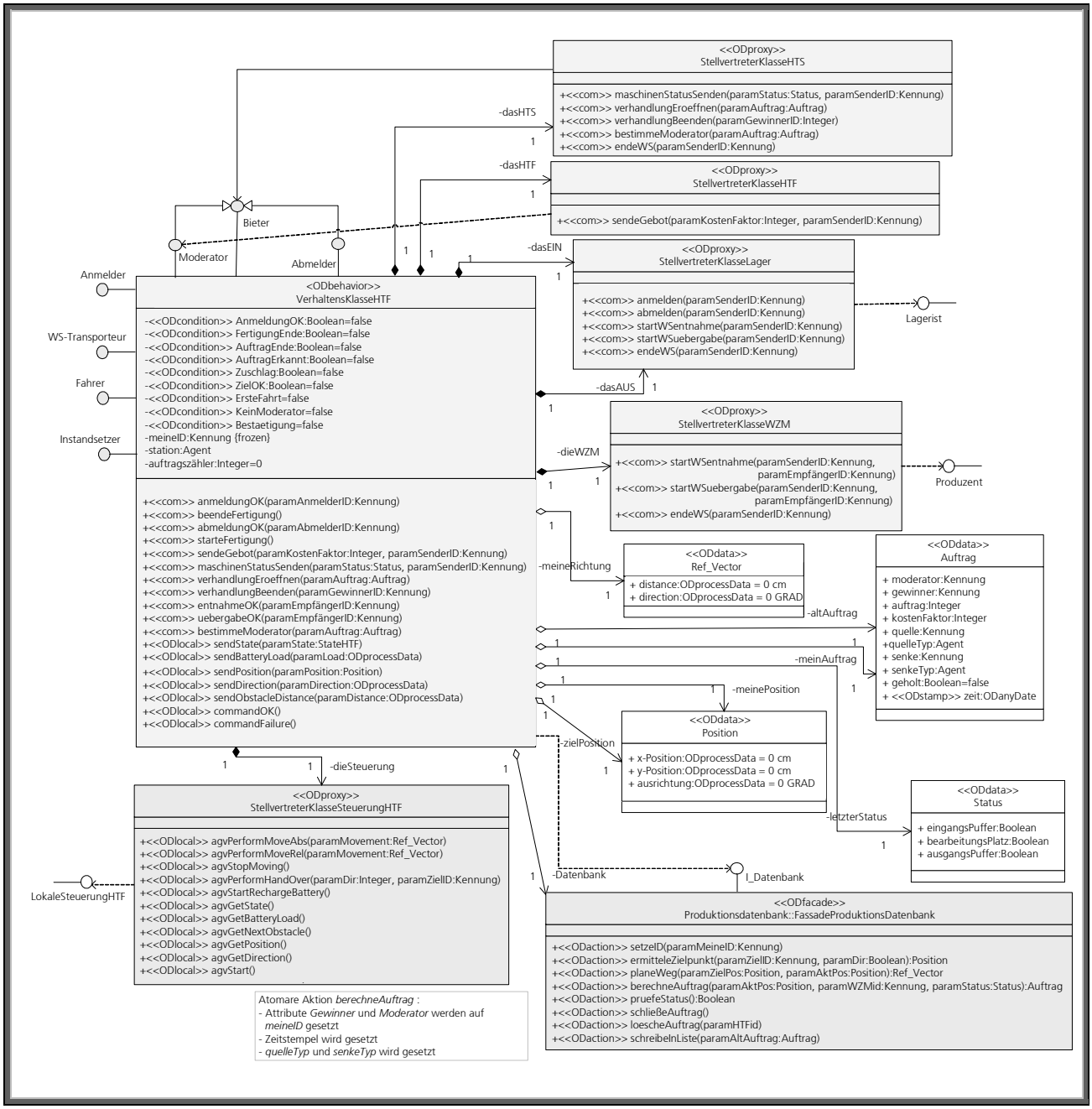

Bild 7.13: Komponenen-Spezifikation der aktiven Steuerungskomponente *HTF_Kopf_Modell*

Das Datendiagramm der Steuerungskomponente *HTF_Kopf_Modell* ist in Bild 7.14 dargestellt. Da keine zusammengesetzten Datenklassen verwendet werden, sind hier lediglich die Datenklassen beschrieben, zu denen die Verhaltensklasse *VerhaltensKlasseHTF* eine Abhängigkeit aufweist (z.B. durch das Attribut *station*). Die detaillierte Spezifikation der Interface-Klassen in Bild 7.15 ist keine eigenständige UML-Diagrammtyp-Rolle, sondern ist aus Gründen der Darstellbarkeit getrennt von der Komponenten-Spezifikation visualisiert.

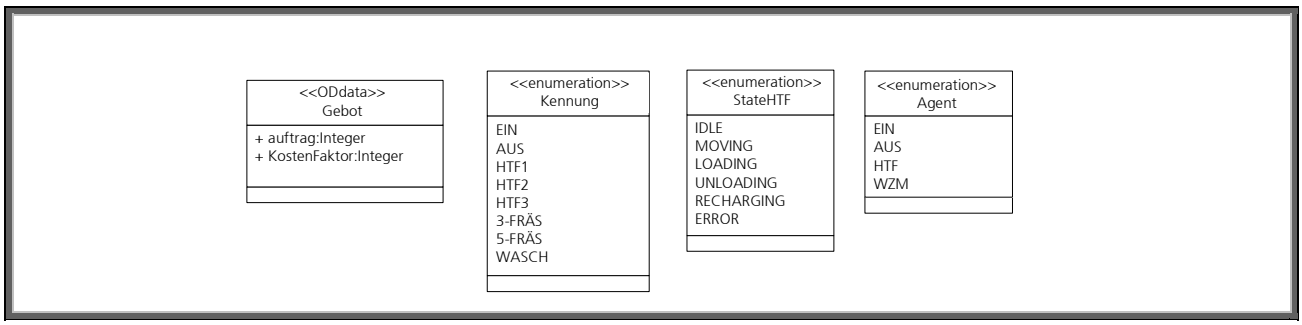

Bild 7.14: Datendiagramm der Steuerungskomponente *HTF_Kopf_Modell*

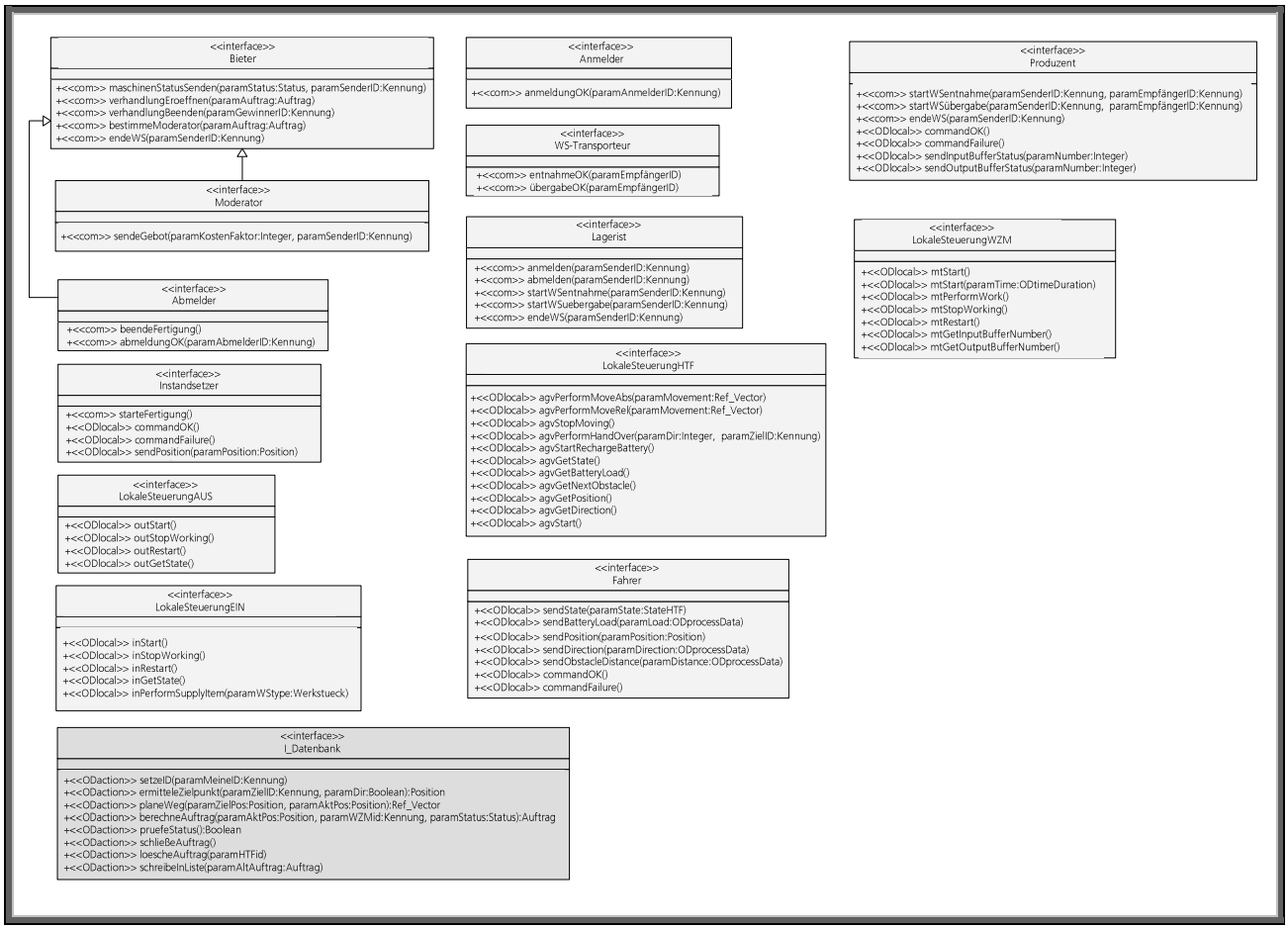

Bild 7.15: Detaillierte Beschreibung der Interface-Klassen der Fallstudie

Das Komponenten-Verhaltensdiagramm weist im Gegensatz zu Komponente-Spezifikation nur einen geringen Anteil an generischen Beschreibungselementen auf. Lediglich die Rollen-Zustände und die Änderungsereignisse, die zu einem Wechsel des Rollen-Zustandes bzw. der Rolle führen, können aus der Spezifikation der entsprechenden Steuerungskomponente und den dazugehörigen System-Interaktionsdiagrammen generiert werden bzw. sind aus Gründen der Konsistenz vorgegeben. Die möglichen zu empfangenen oder zu sendenden Nachrichten und auch die aufrufbaren, atomaren Aktionen und nicht-unterbrechbaren Aktivitäten sind zwar durch die Komponenten-Spezifikation vorgegeben, jedoch wird hierdurch lediglich der Spielraum beschrieben, den der Entwickler zur Spezifikation des Verhaltens hat. Das Verhalten der Steuerungskomponente selbst ist damit in keiner Weise bestimmt.

Die Spezifikation des Komponenten-Verhaltensdiagramms zeigt beide Beschreibungstechniken bezüglich der Anwendung von Rollen-Zuständen. Zum einen ergibt sich die sequentielle Anordnung von Rol-

len-Zuständen aus der Spezifikation von Gruppen- und Rollen-Schnittstellen, die nicht keine Vererbungen aufweisen (Rollen-Zustände *Instandsetzer*, *Anmelder*, *Fahrer* und *WS-Transporteur*). Hieraus folgt eine gute Kompositionalität des Verhaltens einer Steuerungskomponente, da sie ausschließlich mittels Änderungsereignissen und auf der obersten Ebene der Zustandshierarchisierung verbunden sind. Rollen, die mittels Vererbungen spezifiziert sind, führen zu hierarchisch angeordneten Rollen-Zuständen. Dies gilt für die Rollen *Moderator* und *Abmelder*. Die Kompositionalität ist weniger ausgeprägt, da Transitionen auch direkt auf Sub-Zustände führen können (siehe Rollen-Zustand *Abmelder* in 7.16).

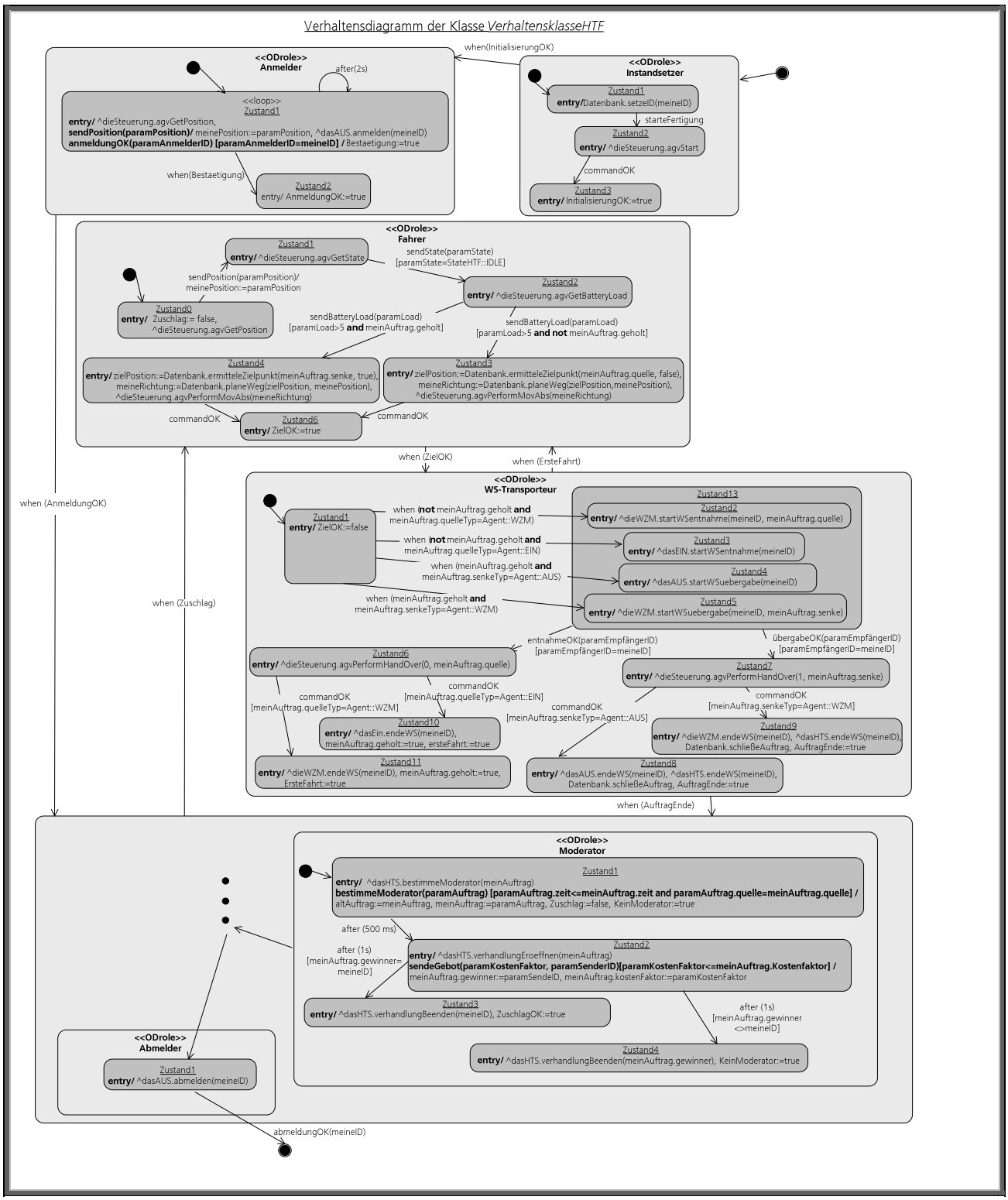

Bild 7.16: Komponenten-Verhaltensdiagramm der Steuerungskomponente HTF_Kopf_Modell bzw. seiner Verhaltensklasse

Ein wichtiger Aspekt, der im Zusammenhang mit der Beschreibung des Komponenten-Verhaltensdiagramm zu nennen ist, ist der direkte Zusammenhang zwischen Beschreibungstechniken des System-Interaktionsdiagramms und denen eines Komponenten-Verhaltensdiagramms. Versteht man die Beschreibung der Werkstückübergabe bzw. -entnahme mit Hilfe von bedingten, horizontalen Nachrichten als ein *Beschreibungsmuster*, so findet man dieses Muster im Rollen-Zustand *WS-Transporteur* wieder. Zwischen den Sub-Zuständen *Zustand1* und *Zustand13* werden mittels der booleschen Attribute *meinAuftrag.geholt* und *meinAuftrag.quelleTyp* bzw. *senkeTyp* die Transitionen geschaltet, die die Werkstückübergabe von der -entnahme und den Typ des adressierten Produktionsagenten unterscheiden. Auf diese Weise wird durch Transitionen eine logische Verknüpfung ausgedrückt (siehe Kapitel 2.4). Dies ist im Sinne der Anwendbarkeit eine für den Entwickler unübersichtliche Beschreibung. Die Adressierung der Produktionsagenten lässt sich alternativ, wie in der Konstellation der Rollen-Zustände *Bieter* und *Moderator* beschrieben, auch über eine Agenten-Gruppe mit einer gemeinsamen Rolle durchführen (siehe Bild 7.4). Hierbei dienen die Nachrichten-Parameter param*EmpfängerID* bzw. param*SenderID* zur Unterscheidung der Adressaten bzw. zur Entscheidung beim Empfänger, ob die Nachricht für ihn bestimmt ist (siehe Kapitel 6.2.2).
Nachteilig ist bei diesem Beschreibungsmuster, dass im Kontext des System-Interaktionsdiagramms die beteiligten Produktionsagenten nur dann als Gruppe vereinfacht werden können, wenn sie kein individuelles Verhalten aufweisen.
Zusammenfassend kann festgestellt werden, dass die Ablaufsteuerung der Fallstudie, die in der Spezifikation des regulären und irregulären Betriebes (siehe Kapitel 10.3.1 und 10.3.2) beschrieben ist, durch das Komponenten-Verhaltensdiagramm gut abgebildet werden kann. Hierbei ist aber zu berücksichtigen, dass die ODEMA-Methode konzeptionsgemäß zu einer dezentralen Ablaufsteuerung führt und ein Komponenten-Verhaltensdiagramm nur einen dezentralen Teil der Ablaufsteuerung beschreiben kann (Anforderung 6 aus Tabelle 7.1).
Im Zusammenhang mit der Komponenten-Spezifikation wird die Schnittstelle zur Produktionsdatenbank *I_Datenbank* beschrieben (Objekt *Datenbank*, siehe Bild 7.13). Der Import der Fassaden-Klasse *FassadeProduktionsdatenbank* aus der passiven Steuerungskomponente *Produktionsdatenbank* ermöglicht aus der Perspektive des Komponenten-Verhaltensdiagramms somit die Beschreibung des Zugriffs auf die dort abgelegten Daten (Anforderung 8 aus Tabelle 7.1).

7.3.2 Anwendung der *Prüfung des objektorientierten Steuerungsentwurfs auf Konsistenz*

Die in Tabelle 6.14 beschriebenen Prüfkriterien für die syntaktische Konsistenz bezüglich der importierten und exportierten Schnittstellen-Klassen sowie der Fassaden- und Verhaltensklasse lässt sich anhand der erweiterten UML-Diagramme (siehe Bild 7.13 bis 7.16) unmittelbar durch Anschauung anwenden. Die Definition der Verhaltensklasse als Singleton lässt sich nicht unmittelbar feststellen, da im Zusammenhang der Spezifikation von Steuerungskomponenten keine UML-Diagrammtypen verwendet werden, die eine Beschreibung der entsprechenden Objekte beinhalten. Anhand der Multiplizitäten von

den mit der Verhaltensklasse verbundenen Aggregationen und Kompositionen, die nicht größer als eins sein dürfen, lässt sich aber feststellen, ob diese Forderung erfüllt wurde.

Die syntaktische Konsistenz der Abbildung der Rollen auf entsprechende Rollen-Zustände bzw. die durch die Schnittstellen-Klassen eingeschränkte Nutzung der zu empfangenden oder zu sendenden Nachrichten lässt sich ebenfalls unmittelbar aus der Anschauung der Komponenten-Spezifikation und des Komponenten-Verhaltensdiagramms prüfen (siehe Bild 7.13 und 7.16). Die geforderte Konsistenz der temporalen Spezifikation *Maximale Ausführungszeit* und *Schleifen-Zustand* muss bezüglich der hier dargestellten Spezifikationen der Fallstudie nicht überprüft werden, da keine nicht-unterbrechbaren Aktivitäten mit entsprechenden maximalen Ausführungszeiten zur Anwendung kommen. Im Fall der Prüfung auf semantische Widerspruchsfreiheit kann eine Prüfung bezüglich der Verhandlungszeit eines Transportauftrages (*1.5 s*, siehe Bild 7.4) durchgeführt werden. Tabelle 6.15 beschriebt hierfür das Verfahren *Überwachungszeit*, das auf die Ausführungspfade zwischen *Zustand1* und *Zustand3* (Ausführungspfad A) bzw. auf *Zustand1* und *Zustand4* (Ausführungspfad B) im Rollen-Zustand Moderator angewendet werden muss. Hierbei sind lediglich die beiden Zeit-Ereignisse *after(500 ms)* und *after(1 s)* zu addieren. Die Zeitschranke aus dem System-Interaktionsdiagramm wird hierbei nicht verletzt (Anforderung 11 aus Tabelle 7.1).

7.4 Abschließende Bewertung der Validation und Anwendung der Methode

Die Validation der ODEMA-Methode anhand der *Referenzfallstudie Produktionstechnik* hat gezeigt, dass gemäß der Konzeption die Methode dort ihre Stäken hat, wo die Methoden-Axiome erfüllt werden müssen, die auf das Beschreibungsmittel abzielen. Dies sind die Methoden-Axiome *Strukturprinzip*, *System-* und *Sofware-Dekompositionsprinzip* sowie *Kausal-* und *Temporalprinzip* (Einschränkung gibt es lediglich bezüglich Anforderung 5). Die Integration der unterschiedlichen Modelle der Automatisierungstechnik begründet diese Eigenschaft.

Betrachtet man zusammenfassend die Anwendung der Verfahren zur Prüfung der Spezifikation (Anforderungen 11 und 14, siehe Tabelle 7.1 und 7.3), so kann für die Prüfung der syntaktischen Konsistenz und semantischen Widerspruchsfreiheit temporaler Spezifikationen festgestellt werden, dass nur ein kleiner Teil der Prüfkriterien aus den Tabellen 6.8, 6.9 und 6.15 bezüglich der Fallstudie zum Einsatz kommen, da die Anforderungen an diese durch die Fallstudie nur wenig komplex sind. Trotzdem kann davon ausgegangen, dass nicht nur im Kontext dieser Validation die genannten Verfahren die Anforderung 11 voll erfüllen, da sich an der Beschreibungstechnik temporaler Spezifikationen bei höherem Komplexitätsgrad prinzipiell nichts ändert. Nur die Anwendung eines entsprechenden Werkzeugs, dass die Prüfung automatisch durchführt, wird bei steigender Komplexität der Spezifikation voraussichtlich notwendig sein.

Die Verfahren der Prüfung der Spezifikation nach Anforderung 14 kommen deutlich umfangreicher zum Einsatz, aber auch hier ist durch das Fehlen der Spezifiktion von UML-Funktionsblöcken und Steuerungskomponenten der Prozesssteuerungsebene und damit auch der Prozessebene eine nur mittlere Komplexität der zu spezifizierenden Steuerung festzustellen. Dies hat auch für diese Verfahren

zur Folge, dass nur eine Untermenge der möglichen Prüfkriterien erfolgreich angewendet werden konnten.

Anforderung / Erfüllungsgrad	1	2	3	4	5	6	7	8	9	10	11	12	13	14
voll erfüllt	X	X	X	X		X	X	X	X	X	X	X		X
eingeschränkt erfüllt					X								X	
nicht erfüllt														

Tabelle 7.3: Bewertung der Anwendung der Methode anhand der Methoden-Axiome (siehe Tabelle 7.1)

Einschränkungen bezüglich des Erfüllungsgrades müssen lediglich für die Anforderung 5 und 13 - also bezüglich des *Software-Dekompositionsprinzips* und der *Anwendbarkeit* - gemacht werden. Die gemeinsame Spezifikation von horizontalen und vertikalen Nachrichten auf Schnittstellen-Klassen führt gerade im Zusammenhang mit der Spezifikation von Gruppen-Schnittstellen zu einer prinzipiellen Offenlegung von lokalen Funktionen eines Produktionsagenten, die explizit nicht gewünscht ist (sieh Kapitel 7.2.2). Dies kann zwar durch eine entsprechenden Entwurf der Rollen umgangen werden, bleibt aber prinzipiell möglich bzw. schränkt die Entwurfsfreiheit des Entwicklers ein (Einschränkung der Anforderung 5).

Der hohe Anteil an Konsistenz-sichernden UML-Beschreibungselementen bezüglich der UML-Diagrammtyp-Rolle *Komponenten-Spezifikation* führt zur eingeschränkten Erfüllung der Anforderung 13. Ergänzt wird diese Beurteilung durch nicht befriedigend gelöste Darstellung von Komplexen Zuständen, wie das Beispiel des Rollen-Zustand *Bieter* zeigt. Trotz der Anwendung abgeleiteter Rollen-Zustände, ist die Spezifikation dieses Rollen-Zustandes aufgrund der zahlreichen Fallunterscheidungen aus Anwendersicht noch zu unübersichtlich.

Zusammengefasst wird das Ergebnis der Validation in Tabelle 7.3.

8 Zusammenfassung und Ausblick

Der generelle Trend des steigenden Automatisierungsgrades in Produktionssystemen und –anlagen und die wachsende Dezentralisierung der dazu notwendigen Steuerungen haben zu einem starken Anwachsen von Software-basierten Steuerungsfunktionen in Produktionssystemen geführt. Aufgrund der Migration von Technologien aus der Informationstechnik in die Automatisierungstechnik lässt sich darüber hinaus eine weiter ansteigende Heterogenität bezüglich der Programmierung von Automatisierungsgeräten und Kommunikationssystemen beobachten. Um in diesem Umfeld komplexer werdender, Software-basierter Steuerungen kosteneffizient zu entwickeln, bedarf es einer entsprechenden Methode zur Spezifikation von Steuerungen. Ausgangspunkt hierfür sind die bewährten objektorientierten Methoden der Informationstechnik, die auf der *Unified Modeling Language* (UML) als Beschreibungsmittel basieren.

In der Analyse der Spezifikationsmethoden und Beschreibungsmittel, die zurzeit in der Automatisierungstechnik und Softwaretechnik zur Anwendung kommen, wurde gezeigt, dass diese nur in Teilen den Methoden-Axiomen, die als Anforderung verstanden werden, genügen. Weiterhin wurde herausgestellt, dass sich die wichtigsten dort aufgeführten Beschreibungsmittel als Diagrammtyp in der UML wieder finden und somit eine objektorientierte Methode basierend auf der UML am besten geeignet ist, die gestellte Aufgabe zu erfüllen.

Die im Stand der Wissenschaft vorgestellten Modelle aus der Automatisierungstechnik dienten im Kern zur Darstellung der Beschreibung der System-Dekomposition von Steuerungen. Es konnte gezeigt werden, dass diese Modelle mit ihren spezifischen Eigenschaften übertragen auf die UML diese als Beschreibungsmittel für Steuerungen notwendig ergänzen. Darüber hinaus konnte dargestellt werden, dass durch die Nutzung der Erweiterungsmechanismen der UML für diese Abbildung die Anwendbarkeit, d.h. die Nutzung von Standard-CASE-Tools, erhalten bleibt.

Konzeption und Entwicklung der so genannten ODEMA-Methode führten auf Basis des zu einem iterativ-inkrementellen Vorgehensmodell erweiterten Begriffs der *Spezifikationsmethode* und der um die entsprechenden Begriffsmodelle und Modellvorstellungen erweiterten UML zu einer objektorientierten Methode zur Spezifikation von Steuerungen. In diesem Kontext wurde ein so genanntes *UML-Profil* für ein Plattform-unabhängiges Beschreibungsmittel entwickelt, das den Einsatz der Methode in einem Standard-CASE-Tool möglich macht. Des Weiteren wurden in diesem Zusammenhang entsprechende Verfahren zur Prüfung der syntaktischen Konsistenz und semantischen Widerspruchsfreiheit eingeführt sowie deren Anwendung durch das Vorgehensmodell definiert.

Die abschließend vorgenommene Anwendung und Validation der Methode anhand der *Referenzfallstudie Produktionstechnik* hat ihre Anwendbarkeit nachgewiesen.

Die zahlreichen Erweiterungen der UML bezüglich der ODEMA-Methode, die während der Entwicklung dieser Methode nötig waren, haben aber auch deutlich gezeigt, dass vor allem im Bereich der physischen Sicht der UML noch Defizite in Syntax und Semantik zu finden sind. Auch die Beschreibung von UML-Subsystemen als Instanzen in z.B. einem Sequenzdiagramm ist semantisch zwar definiert,

aber es fehlt eine entsprechende Syntax. Nicht zuletzt werden gemeinsame elementare Datentypen der UML, physikalische Einheiten und eine Beschreibung algebraisch-logischer Operationen (*Action-Language*) vermisst. Hier wird die kommende, überarbeitete Version 2.0 der UML zwar in einigen Punkten Abhilfe schaffen, aber die Notwendigkeit die speziellen Anforderungen aus der Automatisierungs- bzw. Steuerungstechnik erfüllen zu können, wird weiterhin die Definition eines entsprechenden UML-Profils notwendig machen.

Ein grundlegende Anforderung an eine Spezifikation ist die Möglichkeit zur Verifikation bestimmter Eigenschaften. Die semiformale UML ist von diesem Standpunkt noch nicht ausreichend definiert. Neben den Bemühungen der *Object Management Group* den formalen Anteil der UML zu erhöhen, gibt es den Ansatz des formalen *Referenzmodells* [GRORHO_00]. Dieser Ansatz geht davon aus, dass anwendbare Beschreibungsmittel - also in der Anwendung weit verbreitete Beschreibungsmittel – stets einen vom Entwickler interpretierbaren und somit informalen Anteil haben und aus einer bestimmten Sicht angewendet werden. Die Verifikation auf Basis des Referenzmodells gründet sich auf der Annahme, dass ein System aus unterschiedlichen Sichten durch unterschiedliche, nicht zusammenhängend definierte Beschreibungsmittel spezifiziert wird und ausschließlich die Tatsache, dass sie ein identisches System beschreiben, den Schluss einer gemeinsamen formalen Basis zulässt. Die formale Basis ist im Referenzmodell z.B. durch Transformations- und Transitionssysteme gegeben. Ein Beschreibungsmittel einer bestimmten Sicht (z.B. Sequenzdiagramm) wird zunächst auf das Referenzmodell abgebildet und dann mit einem anderen Beschreibungsmittel der selben oder einer anderen Sicht verglichen und verifiziert. Auf diese Weise können semantische Widersprüche ermittelt werden. Für eine Auswahl von UML-Diagrammen (Klassen- und Zustandsdiagramm) wurde dies Verfahren bereits erfolgreich angewendet [GRORHO_00]. Da das ODEMA-Beschreibungsmittel vollständig auf der UML basiert, ist das Referenzmodell auch hier anwendbar. Da das Referenzmodell prinzipiell alle Sichtenorientierten Beschreibungsmittel abbilden kann, ist ein Werkzeug auf Basis des Referenzmodells in der Lage, neben den UML-Diagrammtypen auch Beschreibungen und Pläne aus der mechanischen bzw. elektromechanischen Konstruktion zu verifizieren.

Eine grundsätzliche Erweiterung der ODEMA-Methode könnte vom wissenschaftlichen Standpunkt aus notwendig werden, wenn man von den zukünftigen Anforderungen ausgeht, die durch neue Paradigmen bezüglich der Steuerung von Produktionssystemen ausgehen. Beispielhaft seinen die *Holonic* und *Bionic Manufacturing Systems* zu nennen. In diesen Systemen werden Steuerungen nicht mehr reaktiv, sondern z.B. durch *Gebote* und *Verbote* zu beschreiben sein, wie es beispielsweise mit Hilfe der zurzeit in der Informationstechnik untersuchten *Policy-Languages* möglich ist [DAMIAN_01].

Elementar für die prototypische und industrielle Anwendung der Methode ist die Verfügbarkeit von entsprechenden Werkzeugen. Aufgrund der Konzeption der Methode können Standard-CASE-Werkzeuge zur Modellierung zum Einsatz kommen. Um aber die zusätzlichen Möglichkeiten zur Prüfung und Validation der Spezifikationen nutzen zu können, sind Erweiterungen dieser Werkzeuge notwendig. Erste prototypische Werkzeuge wurden bereits vorgestellt [BRAATZ_03b].

Werkzeug-Neuentwicklungen sind aber auch im Bereich der Programmcode-Generierung notwendig, um aus einer UML-Beschreibung ablauffähige Steuerungen für eine bestimmte Plattform bzw. für ein Automatisierungsgerät zu generieren. Zurzeit werden zwei verschiedene Techniken hierfür vorgeschlagen. Zum einen wird eine Programmcodegenerierung mittels sog. *Templates* vorgeschlagen. Ebenfalls aussichtsreich sind zum anderen die Entwicklungen, die auf der XML-Transformationssprache *Extensible Stylesheet Language Transformation* (XSLT) basieren [BRAATZ_03b, STURM_02].
Die Techniken zur Programmcode-Generierung sind als entscheidend für die industrielle Anwendbarkeit der ODEMA-Methode anzusehen.

Development of a method for object-oriented specification of control systems

The still ongoing trend of increasing the level of automation within manufacturing systems and the also growing ratio of decentralized control systems leads to a stronger impact of software-based functions in these control systems. Looking from the programming point of view, it can be observed, that for developing these control applications within manufacturing systems heterogeneity is the main issue which has to be overcome. Migration of further technologies coming from the area of information technology (IT) enforces this kind of heterogeneity. Concerning these obstacles an integrated method for developing cost efficiently these control systems is needed.

For a very similar purpose object-oriented methods based on the Unified Modelling language (UML) were developed for different domains within the IT. These methods were chosen as a starting point for the new method (*Object-oriented Method for Developing Technical Multi-Agent-Systems, ODEMA)* should be developed within the present PhD thesis.

As a first step, present techniques for specifying and programming control systems were analysed, also as the already mentioned object-oriented methods for developing embedded realtime-systems were taken under consideration. A defined set of features, the so called method-axioms, was applied throughout analysis work as requirements which served as a proofing guideline for different description languages and methods. As a result it was shown, that the description languages for control systems can be seen as graphical programming languages (e.g. IEC 61131). Therefore the architectural aspects of control systems (*system-decomposition* and *structure*) could not be described sufficiently. It was also derived that description languages which describe causal and timed behaviour of control systems can be found also as diagrams within UML, which therefore can be understood as a toolbox for the approach of this PhD thesis.

As a second step basic models which describe architectural aspects of control systems, were analysed. These basic models cannot be used as description languages, but they can be seen as mental models used for illustration of basic principles. Therefore they must be integrated as a part of the UML-based description language, which can be accomplished by extension mechanisms the UML is offering. Following this rough approach UML CASE-Tools can also be applied by ODEMA.

Based of the understanding, that a method consist of a procedure and description language, conception and developing of the ODEMA-method was distinguished in two different lanes. At first, a procedure model which describes steps, products and participating roles was derived and assembled to a iterative, incremental, use case driven and architecture centric procedure model. Integrating the basic models, mainly the models of production-agent, functionblock and control-component, into the description language of ODEMA was the second lane resulting in a platform-independent UML-Profil. Procedures for proofing consistency of syntax and semantics of a particular specification cover both

lanes and are also part of ODEMA. Finally, the method was validated by applying a case study which specifies holonic controlled material flow.

As first result of validation it can be pointed out, that UML lacks instantiation of *UML-Functionblocks* modelled by *UML-subsystems* and a formalism for describing operations on data and objects (*action language*). Also a set of basic and platform-independent data types and physical units is missed. These open issues concerning to the UML-Standard would probably be solved by the next releases of the UML.

Moving the UML forward to a formal description language is a further step requested by application domains faced by safety requirements of software-based systems. Future UML releases would not cover this issue as far this can be foreseen concerning the actual UML roadmap. Therefore additional techniques were developed so far. The so called *reference model* based on transformation and transition systems is an example for this purpose.

As conclusion, this PhD thesis points the way to an object-oriented method for specification of control systems within manufacturing systems as an important step towards a common method overcoming heterogeneity.

9 Literatur

9.1 Monografien, Artikel und Internetquellen

[ARLT_01] **Arlt, Volker; Koniecny, Frank:**
Funktionales Engineering modularer, verteilter Automatisierungsapplikationen auf Basis der IDA-Technologieplattform, In: Tagungsband SPS/IPC/Drives Kongress 2001 in Nürnberg. Heidelberg: Hüthig, 2001, S. 83 - 91

[BALZERT_99] **Balzert, Helmut:**
Lehrbuch der Objektmodellierung. Berlin: Spectrum Akademischer Verlag, 1999

[BARNARD_01] **Barnard, Judith:**
Use-Case Tree. In: Journal of Objectoriented Programming (2002) Februar, S. 19 - 24

[BARTELS_01] **Bartels, Jan; Vogel, Birgit:**
Systementwicklung für die Automatisierung im Anlagenbau.
In: at Automatisierungstechnik 49(2001)5, S. 214 - 224

[BENDER_92] **Bender, Klaus (Hrsg).:**
Profibus – Der Feldbus für die Automation. München und Wien: Hanser Verlag, 1992

[BENDER_99] **Bender, Klaus; Albert, József:**
UML-basierte Beschreibungssprache zur hybriden Maschinenmodellierung, In: Tagungsband Entwicklung und Betrieb komplexer Automatisierungssysteme EKA 1999. Braunschweig: Institut für Regelungs- und Automatisierungstechnik, 1999, S. 341 - 350

[BENDER_00] **Bender, Klaus et al.:**
Fabrikautomation 2000. In: Werkstatt und Betrieb, 133 (2000)März, S.16 - 19

[BOOCH_99] **Booch, Grady et al.:**
Das UML-Benutzerhandbuch. Bonn: Addison-Wesley, 1999

[BOOCH_97] **Booch, Grady:**
Objektorientierte Analyse und Design. Bonn: Addison-Wesley, 1997

[BRAATZ_00] **Braatz, Arnulf:**
UML-basierte Software-Spezifikation von dezentralen Produktionssystemen. In: Konferenzband SPS/IPC/DRIVES 2000. Heidelberg: Hüthig Verlag, 2000, S. 336 - 344

[BRAATZ_01a] **Braatz, Arnulf et al.:**
UML-basierte Software-Spezifikation und Entwicklungswerkzeuge für Systeme der Automatisierungstechnik. In: Entwicklung und Betrieb komplexer Automatisierungssysteme EKA 2001. Braunschweig: Institut für Regelungs- und Automatisierungstechnik, 2001, S. 91 – 100

[BRAATZ_01b] **Braatz, Arnulf; Westkämper, Engelbert:**
Eine Methode zur objektorientierten Software-Spezifikation von dezentralen Automatisierungssystemen mit der Unified Modelling Language (UML). In: at Automatisierungstechnik 49 (2001)5, S. 225 – 233

[BRAATZ_03a] **Braatz, Arnulf et al.:**
Konzeption und Entwicklung eines UML-basierten Funktionsblockmodells für den objektorientierten Steuerungsentwurf. In: Tagungsband Entwicklung und Betrieb komplexer Automatisierungssysteme EKA 2003. Braunschweig: Institut für Regelungs- und Automatisierungstechnik, 2003, S. 341 – 355

[BRAATZ_03b] **Braatz, Arnulf:**
Vom Modell zur Steuerung – Ablaufprogramme automatisch erzeugen, In: Tagungsband SPS IPC DRIVES Nürnberg 2003. Berlin: VDI Verlag, 2003, S. 195 - 204

[BRENN_01] **Brennan, R. W.; Fletcher, M.; Norrie, D. H.:**
Design of Real-Time Distributed Manufacturing Control Systems using UML Capsules. In: Proceedings 7th International Conference on Object-Oriented Information Systems (OOIS 2001), London: Springer Verlag, 2001, S. 382 – 391

[CHEES_02] **Cheesman, John; Daniels, John:**
UML Components, Bonn: Addison-Wesley, 2001

[DFG_98] **DFG-Schwerpunktprogramm SPP 1064**:
http://www.tfs.cs.tu-berlin.de/projekte/indspec/SPP/ (Stand: Dezember 2003)

[DAMIAN_01] **Damianou, Nicodemos:**
The Ponder Policy Specification Language. In: Proceedings Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001. London: Springer-Verlag, 1995, S. 18 - 39

[DIETSCH_94] **Ditsch, H.:**
Physikalische Eigenschaften verteilter Rechensysteme: Beiwerk oder Basis?, In: Wedekind, H.: Verteilte Systeme. Mannheim, Wien, Zürich: BI-Wissenschaftsverlag, 1994, S. 3 - 16

[DOEB_00] **Doebrich, Udo et al.:**
Strukturen künftiger verteilter leittechnischer Systeme am Beispiel der Feldtechnik, In: Automatisierungstechnische Praxis atp 42 (2000)9, S. 39 - 43

[DOUG_99] **Douglass, Bruce Powel:**
Doing Hard Time – Developing Real-Time Systems with UML, Objects, Frameworks and Patterns. Bonn: Addison-Wesley, 1999

[DRÖSCH_98] **Dröschel, W. et al.:**
Inkrementelle und objektorientierte Vorgehensweisen mit dem V-Modell 97. München: Oldenbourg Verlag, 1998

[DUBBEL_97] **Beitz, Wolfgang; Grote, Karl-Heinrich (Hrsg.):**
Dubbel – Taschenbuch für den Maschinenbau. Berlin: Springer Verlag, 1997

[DUJMO_02] **Dujmovic, S.:**
Unterstützung der Anwendungsentwicklung mit Komponenten-Frameworks. In: Tagungsband: Software-Engineering in der industriellen Praxis. Düsseldorf: VDI-Verlag, 2002, VDI-Berichte 1666, S. 3 - 15

[DUMSKY_02] **Dumsky, Günther:**
Der Durchbruch. In: Computer & Automation (2002)5, S. 34 - 38

[EISEN_00] **Eisenecker, U. W.; Czarnecki, K.**
Generative Programming. Methods, Tools and Applications. Bonn: Addison Wesley, 2000

[ELTING_01a] **Elting, Andreas; Huber, Walter:**
Komponenten und Wiederverwendung – Präzise Modellierung mit der UML. In: ObjektSpektrum (2001)1, S. 40 - 44

[ELTING_01b] **Elting, Andreas; Huber, Walter:**
Immer im Plan? – Programmieren zwischen Chaos und Planwirtschaft. In: c't (2001)2, S. 184 - 189

[ELTING_99] **Elting, Andreas; Huber, Walter:**
Catalysis: ein Vorgehensmodell für die komponentenbasierte und objektorientierte Softwareentwicklung. In: ObjektSpektrum (1999)6, S. 42 - 46

[ENSLOW_78] **Enslow, P. H.:**
What is a Distributed data processing system ?. In: IEEE Computer (1978)11, S. 13 -21

[ENSTE_01] **Enste, Udo:**
Generische Entwurfsmuster in der Funktionsbausteintechnik und deren Anwendung in der operativen Prozessführung. Düsseldorf: VDI-Verlag, 2001, VDI-Fortschrittsberichte Nr.884

[FEHR_02] **Fehr, K.:**
Einführung objektorientierter Softwareentwicklungsmethoden bei der Entwicklung von Prozessautomations-Systemen. In: Tagungsband: Software-Engineering in der industriellen Praxis. Düsseldorf: VDI-Verlag, 2002, VDI-Berichte 1666, S. 77 - 88

[FISCHER_02] **Fischer, Katja; Vogel-Heuser, Birgit:**
UML in der automatisierungstechnischen Anwendung – Stärken und Schwächen. In: Automatisierungstechnische Praxis atp (2002)10, S.63 – 69

[GÖHNER_00] **Göhner, Peter et al.:**
Einsatz von Frameworks bei der Automatisierung technischer Produkte, In: Automatisierungstechnische Praxis atp 42 (2000)6, S. 50 - 57

[GRABO_02] **Grabowski, Jens; Rudolph, Ekkart; Schmitt, Michael:**
Die Spezifikationssprachen MSC und SDT – Teil 2: Specification and Description Language (SDL). In: at Automatisierungstechnik 50 (2002)2, S. A1 – A4

[GRIFFEL_98] **Griffel, Frank:**
Componentware: Konzepte und Techniken eines Software-Paradigmas. Heidelberg: dpunkt-Verlag, 1998

[GRORHO_00] **Große-Rhode, M.:**
Using a formal reference model for consistency checking and integration of UML diagrams. In: Proceedings 5th World Conference on Integrated Design and Process Technology (IDPT 2000), Dallas/Texas, USA. Grandview(USA): Society for Design and Process Science, 2000, S. 153 - 160

[GRUHN_00] **Gruhn, Volker; Thiel, Andreas:**
Komponentenmodelle – DCOM, JavaBeans, Enterprise JavaBeans, CORBA. Bonn: Addison-Wesley, 2000

[GRUND_02] **Grundmann, Uwe:**
JAVA betritt das Parkett. In: Computer & Automation (2002)1, S. 10 –12

[HAREL_87] **Harel, David:**
Statecharts: A visual formalism for complex Systems. In: Science of Computer Programming 8 (1987)3, S. 231 - 274

[HEVER_02] **Heverhagen, Thorsten:**
Kommunikation zwischen Funktionsbausteinen und UML Capsules in einer industriellen Softwareumgebung. In: Software-Engineering in der industriellen Praxis. Düsseldorf: VDI-Verlag, 2002, S. 121 - 132

[HRUBY_00] **Hruby, Pavel:**
Structuring Software Development Artifacts with UML. In: JOOP (2000)2, S. 22 - 33

[HERRT_89] **Herrtwich, R. G. et al.:**
Kooperation und Konkurrenz, Nebenläufige, verteilte und echtzeitabhängige Programmsysteme. Berlin: Springer Verlag, 1989

[HOANG_99] **Hoang, S. H.; Rieger P.:**
Komponentenbasierte Automatisierungssoftware. München und Wien: Hanser Verlag, 1999

[HOEPF_97] **Höpf, M.; Schaeffer, C.; Westkämper, E.:**
Holonic Manufacturing Systems. In: Entwicklung und Betrieb komplexer Automatisierungssysteme. Braunschweig: Institut für Regelungs- und Automatisierungstechnik, 1997, S. 127 - 137

[HÜTTE_96] **Czichos, H. (Hrsg.):**
Hütte - Die Grundlagen der Ingenieurwissenschaften. Berlin und Heidelberg: Springer Verlag, 1996

[IHNS_00] **Ihns, Oliver:**
Aspekte der Performanz und Skalierbarkeit von CORBA-Applikationen (Teil 2). In: Objectspekrum (2000)6, S. 62 - 72

[IHNS_02] **Ihns, Oliver; Heldt, Stefan M.**
„Message-driven Beans"- Einsatz und Integrationsmöglichkeiten. In: Objectspektrum (2002)3, S. 53 -60

[KAHL_98] **Kahlbrandt, B.:**
Software-Engineering. Berlin: Springer Verlag, 1998

[KIESS_95] **Kieß, Jan U.:**
Objektorientierte Modellierung von Automatisierungssystemen. Berlin: Springer Verlag, 1995. Zugl.: Dissertation Universität Stuttgart 1995

[KLEMM_02] **Klemm, Eckehard:**
Generische Maschinenrepräsentation in der Welt von PABADIS. In: Tagungsband SPS/IPC/DRIVES 2002. Heidelberg: Hüthig Verlag, S. 219 - 227

[KOWAL_01] **Kowalewski, Stefan:**
Modellierungsmethoden aus der informatik. In: at Automatisierungstechnik (2001)9, S. A1 – A5

[KRALL_02] **Krallmann, Hermann; Albayrak, Sahin:**
Holonic Manufacturing - Agentenorientierte Techniken zur Umsetzung von holonischen Strukturen. München: TCW-Transfer-Centrum GmbH, 2002 (TCW-Report)

[KRUCHT_98] **Kruchten, P.:**
The Rational Unified Process.Bonn: Addison-Wesley, 1998

[LAUBER_99] **Lauber, Rudolf; Göhner, Peter:**
Prozessautomatisierung 2. Berlin: Springer Verlag, 1999

[LAENGLE_97] **Längle, Thomas:**
Verteiltes Steuerungskonzept für komplexe inhomogene Robotersysteme. Düsseldorf: VDI-Verlag, 1997. Zugl.: Dissertation Universität Karlsruhe 1997, Fortschrittsberichte VDI Reihe 8 Nr. 614

[LUETH_98] **Lüth, T.:**
Technische Multi-Agenten-Systeme: verteilte autonome Roboter- und Fertigungssysteme. München und Wien: Carl Hanser Verlag, 1998

[MEIER_01] **Meier, Jürg; Scharf, Tanja:**
Regeln für die Anwendungsarchitektur verteilter Systeme. In: Objektspektrum (2001)2, S. 77 - 81

[METZ_97] **Metz, J.:**
Rechnergestützte Entwicklung von Automatisierungssystemen auf der Grundlage eines objektorientierten Vorgehensmodells. In: 42. Internationales Wissenschaftliches Kolloquiums, Band 3. Ilmenau: Technische Universität Ilmenau, 1997,
S. 477-481

[MIDDER_98] **Midderhoff, Rainer (Hrsg.) et al.:**
Inkrementelle und objektorientierte Vorgehensweise mit dem V-Modell 97. München: Oldenbourg Verlag, 1998

[MOIK_02] **Moik, Adam:**
Engineering-related formal methods for the development of safe industrial automation systems, In: Automatisierungstechnische Praxis atp (2002)9, S. 56 – 65

[MUEHL_92] **Mühlhäuser, Max; Schill, Alexander:**
Software Engineering für verteilte Anwendungen. Berlin und Heidelberg: Springer Verlag, 1992

[MÜNNE_01] **Münnemann, Ansgar; Ernste, Udo:**
Systemtechnische Integration gehobener Regelungsverfahren. In: at Automatisierung 43 (2001)7, S. 40 - 48

[OESTER_99a] **Oestereich, Bernd et al.:**
Erfolgreich mit Objektorientierung. München: Oldenbourg Verlag, 1999

[OESTER_99b] **Oestereich, Bernd:**
Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language. München: Oldenbourg Verlag, 1999

[PANG_01] **Pang, Chung-Yeung:**
A Usage- and Collaboration-Based Approach to Application Design. In: Journal of Objectoriented Programming (JOOP) (2001)April, S. 18 - 22

[POLKE_94] **Polke, Martin:**
Prozessleittechnik. München: Oldenbourg Verlag, 1994

[POSCH_02] **Posch, T.:**
Modell-basierter Software-Entwurf und Code-Generierung mit UML für Embedded Systeme. In: Software-Engineering in der industriellen Praxis. Düsseldorf: VDI-Verlag 2002, S. 69 - 75

[PREHN_02] **Prehn, Eckehardt:**
Realisierung des OMAC Maschinen-Modells mit Zustandsgraphen und der Komponententechnologie. In: Tagungsband SPS/IPC/DRIVES Nürnberg 2002, 13. Fachmesse und Kongress, Heidelberg: Hüthig, 2002, S. 319 - 326

[REISSEN_98] **Reißenweber, Bernd:**
Feldbussysteme. München: Oldenbourg Verlag, 1998

[REINHA_96] **Reinhardt, H.:**
Automatisierungstechnik – Theoretische und gerätetechnische Grundlagen, SPS. Berlin und Heidelberg: Springer Verlag, 1996

[REINHO_97] **Reinhold, Markus:**
Die UML und das standardisierte Prozessmodell „V-Modell `97": Warum reicht eine Modellierungssprache allein nicht aus? In: Objektspektrum (1997)5, S. 70 - 76

[REINHO_00] **Reinhold, Markus:**
Rational Unified Process 2000 und *V-Modell '97*: Synergie oder Widerspruch? In: Objektspectrum (2000)3, S. 74 – 81

[RIEDL_02] **Riedl, M. et al.:**
EDDL – Electronic Device Descripition Language. München: Oldenbourg Verlag, 2002

[RITTER_00] **Ritter, Arno; Braatz, Arnulf; Winz, G.:**
Agentensysteme in der Produktion. In: Tagungsband SP/IPC/DRIVES Nürnberg 2000, 11. Fachmesse und Kongress. Heidelberg: Hüthig, 2000, S. 208 – 216

[RITBRA_01] **Ritter, Arno; Braatz, Arnulf; Höpf, Michael:**
Models and Modules for Production Agents. In: Proceedings of the 1st International Conference on Information Technology in Mechatronics (ITM'01), Special Session, Istanbul, Türkei, 2001, S. 47-52

[RITTER_01] **Ritter, Arno et al.**:
Transportagenten in der Produktion. In: Tagungsband SPS/IPC/DRIVES Nürnberg 2001, 12. Fachmesse und Kongress. Heidelberg: Hüthig, 2001, S. 211 – 217

[RITTER_02] **Ritter, Arno et al.:**
Agentification for Production Systems. In: Tagungsband Integration of Software Specification Techniques INT '02 in Grenoble. Berlin: Institut für Softwaretechnik und Theoretische Informatik, 2002, S. 21 – 28.

[ROSTAN_02] **Rostan, M.:**
Zykluszeit ist nicht alles: zur Performance von Feldbussystemen in der Fertigung. In: Tagungsband SPS/IPC/DRIVES Nürnberg 2002, 13. Fachmesse und Kongress. Heidelberg: Hüthig, 2002, S. 99 – 108

[RUMB_99] **Rumbaugh, Jim et al:**
The Unified Modelling Language Reference Manual. Bonn: Addison-Wesley, 1999

[RÜFFER_02] **Rüffer, Martin:**
Real-Time UML in der Automatisierungstechnik. In: Tagungsband: Software-Engineering in der industriellen Praxis. Düsseldorf: VDI-Verlag, 2002, S. 53 - 67

[RUMPE_01] **Rumpe, B.; Sandner, R.:**
UML – Unified Modeling Language im Einsatz - Teil 3: UML-RT für echtzeitkritische und eingebettete Systeme. In: at - Automatisierungstechnik, 49(2001), S. A15 – A18

[SELIC_94] **Selic, Bran et al.:**
Real-Time Object-Oriented Modeling. Hoboken(USA): John Wiley & Sons, 1994

[SELIC_98] **Selic, Bran; Rumbaugh, Jim:**
Die Verwendung der UML für die Modellierung komplexer Echtzeitsysteme. In: Objektspectrum (1998)4, S. 24 - 36

[SCHERFF_99] **Scherff, Birgit et al.:**
Feldbussysteme in der Praxis - Ein Leitfaden für den Anwender. Heidelberg u. Berlin: Springer Verlag, 1999

[SCHLEIF_00] **Schleifele, Dieter:**
Offene Steuerungen im Maschinenbau. In: Automatisierungstechnik/WB (2000)3, S. 21 - 23

[SCHNIED_98] **Schnieder, Eckehard et al.:**
Klassifikation und Bewertung von Beschreibungsmitteln für die Automatisierungstechnik. In: at-Automatisierungstechnik (1998)12, S. 582 - 591

[SCHNIED_99] **Schnieder, Eckehard:**
Methoden der Automatisierung. Braunschweig: Vieweg Verlag, 1999

[SCNIED_01] **Schnieder, Eckehard; Jansen, Lars:**
Begriffsmodelle der Automatisierungstechnik – Basis effizienten Engineerings. In: Tagungsband Engineering komplexer Automatisierungssysteme EKA 2001. Braunschweig: Institut für Regelungs- und Automatisierungstechnik TU Braunschweig, 2001

[SIEG_96] **Siegert, H.-J.; Bocionek, S.:**
Robotik: Programmierung intelligenter Roboter. Heidelberg und Berlin: Springer Verlag, 1996

[SIMON_02] **Simon, R.:**
Gerätemodell zur Feldinstrumentierung von Verteilten Automatisierungssystemen. In: at-Automatisierungstechnik (2002)10, S.490 - 499

[SOLVIE_95] **Solvie, M.:**
Zeitbehandlung und Multimedia-Unterstützung in Feldkommunikationssystemen. München und Wien: Carl Hanser Verlag, 1995

[SOMMER_96] **Sommerville, Ian:**
Software Engineering. Harlow (England): Addison Wesley Longman Limited, 1996

[STETTER_00] **Stetter, R.:**
Software – das lästige Anhängsel. In: Computer & Automation (2000)9, S. 24

[STETTER_01] **Stetter, R.:**
Die Systemspezifikation – Ein Leitfaden / Teil 2. In: Computer & Automation (2001)3, 2001, S.14 - 16

[STUDIE_03] **N. N.:**
Embedded Systems und industrielle Software - Umfrage unter deutschen Unternehmen. XCC Software AG: Karlsruhe, 2003

[STURM_02] **Sturm, T. et al.:**
Creating Code from UML with Velocity Templates. In: Proceedings UML 2002 – The Unified Modeling Language 5th International Conference in Dresden. Berlin: Springer, 2002, S.150 - 161

[SPERL_97] **Sperling, Wolfgang; Lutz, Peter**:
Designing Applications For An OSACA Control. In: Proceedings of the International Mechanical Engineering Congress and Exposition, Dallas/USA, November 16-21.Dallas: IEEE, 1997, S. 243-249

[SUND_01] **Sundermeyer, Kurt; Bussmann, Stefan:**
Einführung der Agententechnologie in einem produzierenden Unternehmen – ein Erfahrungsbericht. In: Wirtschaftsinformatik (2001)43, S. 135 - 142

[TERW_01] **Terwiesch, P.:**
Internettechnologien in der industriellen Automatisierung: Anwendungsstand und Zukunftspotential. In: at-Automatisierungstechnik 49(2001), S. 30

[TRAUB_02] **Traub, Andreas:**
Eine komponentenbasierte Softwarearchitektur für mobile Serviceroboter.
Heimsheim: Jost-Jetter Verlag, 2002. Zugl.: Dissertation Universität Stuttgart 2001

[ULLMA_01] **Ullmann, Bernd:**
Eine Methode zur zielorientierten Steuerung von dezentralen Fertigungsinseln. Heimsheim: Jost Jetter Verlag, 2001. Zugl.: Zugl. Dissertation Uni Stuttgart 2001

[VOGHEU_02] **Vogel-Heuser, Birgit:**
UML/RT in der Automatisierungstechnik - Vorstellung und Bewertung.In: Tagungsband SPS/IPC/DRIVES 2002. Heidelberg: Hüthig Verlag, 2002, S. 117 - 125

[WEST_98] **Westkämper, E. et al.**
Dezentralisierung und Autonomie in der Produktion. In: ZWF 93(1998), S. 407 - 410

[West_99] **Westkämper, E.:**
Die Wandlungsfähigkeit von Unternehmen, In: wt Werkstattstechnik 89(1999)4, S. 131 - 140

[WEISS_01] **Weiss, Gerhard:**
Agentenorientiertes Software Engineering. In: Informatik-Spektrum (2001)1, S. 98 - 101

[WEISSHA_99] **Weisshaupt, Bruno:**
Neue Vorgehensweise im Entwicklungsprozess durch den interdisziplinären, prozessorientierten Entwurf intelligenter technischer Systeme. In: 6. Fachtagung: Entwicklung und Betrieb komplexer Automatisierungssysteme. Braunschweig: Institut für Regelungs- und Automatisierungstechnik der TU Braunschweig, 1999,
S. 351 - 368

[WRATIL_96] **Wratil, Peter:**
Moderne Programmiertechnik für Automatisierungssysteme. Würzburg: Vogel Fachbuchverlag, 1996

[ZERBE_99] **Zerbe, Klaus:**
Bauplan für Objekte - Eine Einführung in objektorientierte Verfahren mit der Unified Modeling Language. In: c't (1999)21, S. 338 – 355

9.2 Normen und Spezifikationen

[DIN40719] **N. N.:**
DIN 40719-6: Schaltungsunterlagen, Regeln für Funktionspläne. Berlin: Beuth Verlag, 1992

[DIN66000] **N. N.:**
DIN 66000: Informationsverarbeitung. Mathematische Zeichen und Symbole der Schaltalgebra. Berlin: Beuth Verlag, 1985

[DIN66025] **N. N.:**
DIN 66025-2: Industrielle Automation: Programmaufbau für numerisch gesteuerte Arbeitsmaschinen. Berlin: Beuth Verlag, 1988

[DIN66312] **N. N.:**
DIN 66312: Industrial Robot Language (IRL). Berlin: Beuth Verlag, 1996

[EDDL_03] **PROFIBUS Nutzerorganisation:**
Specification for PROFIBUS Device Description and Device Integration, Volume 2: EDDL Specification. http://www.profibus.com/profibus.html (Stand: Dezember 2003)

[FDCML_03] **N. N.:**
Field Device Configuration Markup Language. http://www.fdcml.org (Stand: Dezember 2003)

[FIPA_98] **N. N.:**
FIPA Specifications. http://www.fipa.org (Stand: Dezember 2003)

[IDA_01] **N. N.:**
Homepage der IDA-Group (Interface of Distributed Automation). http://www.ida-group.org, (Stand: Stand: Dezember 2003)

[IEC61499-1] **N. N.:**
International Electrotechnical Commission, Function blocks for industrial-process measurement and control systems. Ort: Verlag, 2000, Part 1: Architecture IEC 61499-1

[IEC_61804] **N. N.:**
EC SC 65C WG7: Function Blocks for Process Control. Ort: Verlag, 1999, Committee Draft IEC 61804-1

[OMG_02] **N. N.:**
OMG: Modell Driven Architecture MDA. http://www.omg.org (Stand: Dezember 2003)

[OOSE_03] **N. N:**
OOOSE. http://www.oose.de (Stand: Dezember 2003)

[OSACA_02] **N. N.:**
Open system Architecture for Controls within Automation Systems (OSACA), ESPRIT III Project 6379. http://www.osaca.org (Stand: Dezember 2003)

[PLCOPEN_03] **N. N.:**
PLCopen Association. http://www.plcopen.org (Stand: Dezember 2003)

[SDL_02] **N. N.:**
SDL Forum Society. http://sdl-forum.org (Stand: Dezember 2003)

[UML_03] **N. N.:**
OMG: Unified Modelling Language Specification 1.5. http://www.omg.org, (Stand: Dezember 2003)

[UMLprofil_03] **N. N.:**
UML Profile for Scheduability, Performance, and Time Specification. http://www.OMG.org (Stand: Dezember 2003)

[VDI_3694] **N. N.:**
VDI-Richtlinie: Lasten/Pflichtenheft für den Einsatz von Automatisierungssystemen. Berlin: Beuth Verlag, 1991

[VDI_3687] **N. N.:**
VDI/VDE-Richtlinie: Auswahl von Feldbussystemen durch Bewertung ihrer Leistungseigenschaften für industrielle Anwendungsbereiche. Berlin: Beuth-Verlag, 1999

10 Anhang: Referenzfallstudie Produktionstechnik

10.1 Einführung in das Projekt

10.1.1 Veranlassung

In einer von zwei Referenzfallstudien des DFG-Schwerpunktprogramm „*Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen*" wird ein werkstattorientiertes Fertigungssystem vorgestellt, das in der Grundstufe aus drei autonomen Transportfahrzeugen und drei Werkzeugmaschinen (WZM), einer Drei-Achsen-Fräsmaschine (3-FRÄS), einer Fünf-Achsen-Fräsmaschine (5-FRÄS) und einer Waschmaschine (WASCH) besteht [DFG_98]. Zusätzlich sind ein automatisches Hochregallager als Eingangslager (EIN) und ein Ausgangszwischenlager (AUS) vorhanden. Aufgabe des Fertigungssystems ist das Entgraten von Gussteilen (V6-Motoren in der Grundstufe und ergänzend Kurbelwellen in der Ausbaustufe).

Der Materialfluss wird in der Grundstufe von den drei autonomen Transportfahrzeugen realisiert. Da keine zentrale Fahrzeugsteuerung vorgesehen ist und sie über den gleichen Aufbau und identische Fähigkeiten verfügen, wird von holonischen Transportfahrzeugen (HTF) gesprochen. Die Gesamtheit aller HTF im Fertigungssystem wird holonisches Transportsystem (HTS) genannt. Pro Transportfahrt kann vom HTF aufgrund des Lastaufnahmemittels (LAM) und des Werkstückträgers immer nur ein einzelnes Werkstück (Gussteil) übernommen und transportiert werden.

Die hier beschriebene Referenzfallstudie umfasst das Lastenheft für eine Grund- und eine Ausbaustufe des Fertigungssystems. In der Ausbaustufe ist von jedem WZM-Typ eine zweite Maschine vorhanden, um somit einen alternativen Materialfluss zu ermöglichen. Zusätzlich umfasst sie eine Batterieladestation (BL), einen Parkbereich (PB) für die nicht aktiven HTF und ein Blockstrecken-Management-System (BMS). Die Anzahl der insgesamt verfügbaren HTF wird auf sechs erhöht.

10.1.2 Zielsetzung des Automatisierungsvorhabens

Die Erreichung eines maximalen Durchsatzes an Werkstücken unter Beibehaltung vereinfachter Konfigurierbarkeit des Fertigungssystems ist Ziel dieses Vorhabens. Hierfür müssen die Steuerungen für das HTS, die WZM und die Lager EIN und AUS entsprechend spezifiziert werden. Die Batterieladestation und der Parkbereich verfügen über keine zu spezifizierende Steuerung.

10.1.3 Technologisches Umfeld

Das Fertigungssystem ist eingebunden in ein flexibles Produktionssystem, das aus weiteren Fertigungssystemen besteht. Dieses umgebende Produktionssystem ist dafür verantwortlich, dass die Lagerplätze des EIN stets gefüllt sind, d.h. mindestens ein zu entgratendes Werkstück zur Verfügung steht. Dies gilt für Grund- und Ausbaustufe. Das AUS wird von einem nachgeschalteten Fertigungssystem zum Zweck der Montage geleert. Es kann in der Grund- und Ausbaustufe davon ausgegangen werden, dass stets ein freier Lagerplatz vorhanden ist, um ein Werkstück aufzunehmen. Zudem hat das AUS

Kenntnis über die Anzahl der Werkstücke, die vom Produktionssystem als Tages-Soll vorgegeben sind. Erreicht die Anzahl der entgrateten Werkstücke das Tages-Soll, wird die Fertigung in Grund- und Ausbaustufe gestoppt.

In der Grundstufe wird vorausgesetzt, dass die Batteriekapazität eines HTF so ausgelegt ist, dass ein Aufladen während des Betriebes nicht notwendig ist, sondern von Hand in der Stillstandzeit des Produktionssystems erfolgt. Für die WZM wird in der Grundstufe störungsfreier Betrieb zugesichert. Störungen im Antriebsmodul eines HTF sind aber möglich. In der Ausbaustufe ist eine Batterieladung vorzusehen und zusätzlich die Störung einer WZM durch Werkzeugbruch zu berücksichtigen.

Die Einbettung des Fertigungssystems in ein flexibles Produktionsumfeld bringt es mit sich, dass wechselnde Produkte das Fertigungssystem durchlaufen und somit geänderte Konfigurationen im Layout der Werkzeugmaschinen und in der Zusammensetzung der Transportfahrzeuge nötig werden. Um diese Konfigurationen mit minimalen Eingriffen in die bestehende Software zu ermöglichen, erhalten alle Teilsysteme des Fertigungssystems eine dezentrale Steuerung. Die Synchronisation dieser Teilsysteme erfolgt über ein funkgestütztes Broadcast-Kommunikationssystem, über das die Kommunikationspartner Telegramme (mit eindeutigem Zeitstempel) versenden. In der Grundstufe wird von verlustfreier Kommunikation ausgegangen. In der Ausbaustufe sind Störungen des Kommunikationssystems möglich. Es ist dabei der Spezifikation der Steuerungen überlassen, welche Telegramme von einem Teilsystem ausgewertet werden, da wesentlich von der Strategie der Teilsysteme abhängt, welche Informationen benötigt und welche ignoriert werden können.

Es ist zu beachten, dass einzig die dezentralen Steuerungen der HTF als holonisch, d.h. autonom, ausgeführt werden.

Das Layout des Fertigungssystems ist in Bild 10.1 dargestellt. In der Ausbaustufe erhält das Fertigungssystem ein Gitternetz mit Blockstrecken (grau markiert), da nicht mehr von Kollisionsfreiheit der HTF ausgegangen werden kann. Um Kollisionen zu vermeiden, müssen die HTF vor einer Transportfahrt einen Streckenabschnitt oder die gesamte Strecke beim BMS reservieren. Fest definiert sind lediglich die Übergabepunkte vor den Eingangs- und Ausgangspuffern der WZM und Lager (schwarz markiert). Dies gilt auch für die Ladeposition an der BL.

Die Wartepositionen für HTF ohne einen Auftrag müssen von der Spezifikation festgelegt werden.

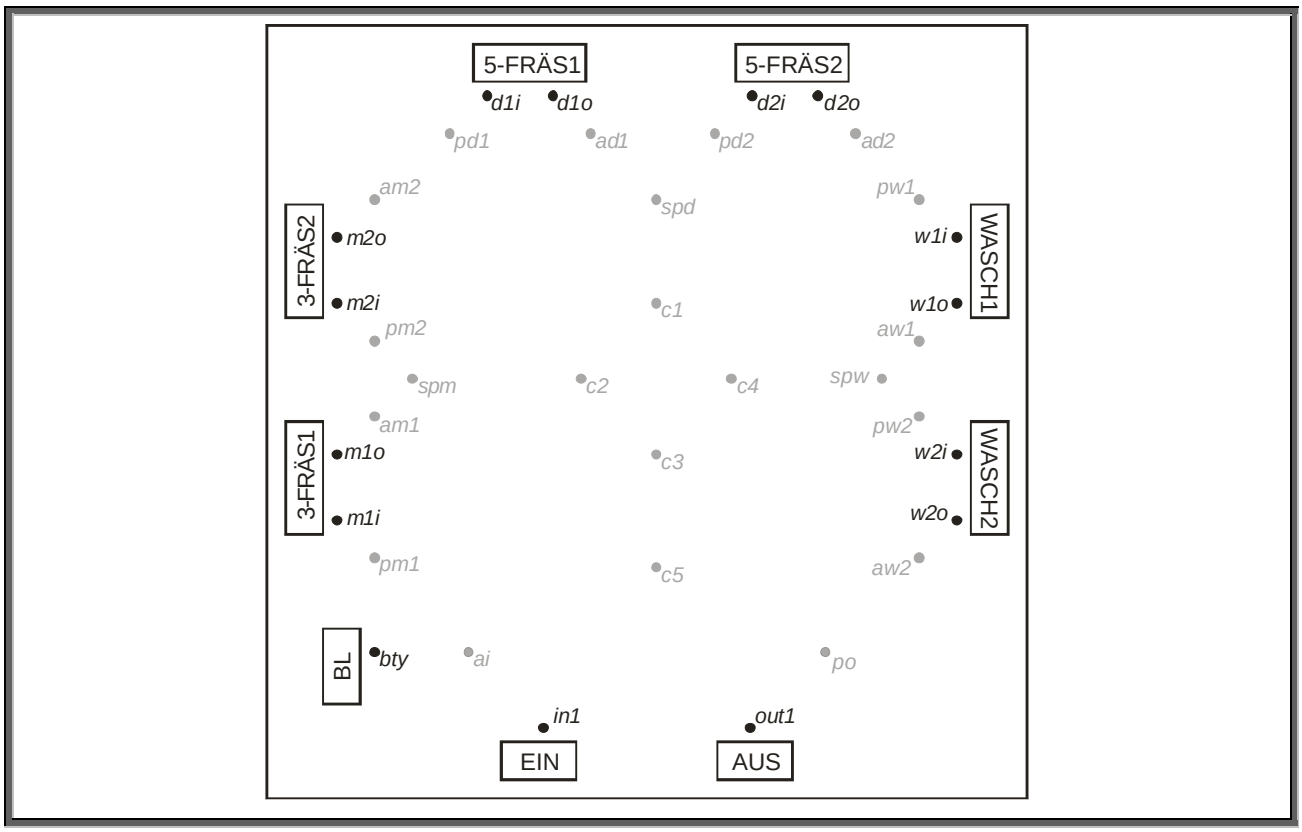

Bild 10.1: Layout des Fertigungssystems (Grund- und Ausbaustufe)

Maschinentyp		Abmessung (Länge x Tiefe x Höhe)
WZM	Drei-Achsen Fräsmaschine	4 m x 1.5 m x 2 m
	Fünf-Achsen-Fräsmaschine	5 m x 2 m x 2 m
	Waschmaschine	4 m x 2.5 m x 2 m
HTF		1.8 m x 1 m x 1.5 m
EIN, AUS		3 m x 4 m x 6 m
BL		1.2 m x 2m x 1.5 m
PB (für 6 HTF dimensioniert)		6 m x 1m

Tabelle 10.1: Abmessungen der Maschinen

10.1.4 Wesentliche Aufgaben des Fertigungssystems:

- Entgraten der Werkstücke, d.h. Fräsen der Gussgrate, Glätten der Bohrungen und Auswaschen der Metallspäne
- Verhandlung und Ausführung der Transportaufträge, d.h. die Werkstücke aus dem automatischen Hochregallager (EIN) zu entnehmen und in der richtigen Ablauffolge den jeweiligen Bearbeitungsmaschinen und schließlich dem Ausgangszwischenlager AUS zuzuführen.
- Vermeidung bzw. Auflösung von Dead-Lock-Situationen der HTF **(Ausbaustufe)**
- Vermeidung von Kollisionen zwischen den HTF **(Ausbaustufe)**

10.2 Aufgabenstellung (Sollzustand)

10.2.1 Aufgabenbeschreibung

Durch die geforderte dezentrale Steuerungsarchitektur des Fertigungssystems wird die detaillierte Aufgabenbeschreibung anhand der Aufgaben der einzelnen dezentralen Maschinen dargestellt.

10.2.2 Aufgabenbeschreibung der dezentralen Werkzeugmaschinen (WZM)

Eine WZM (Drei-Achsen-, Fünf-Achsen-Fräsmaschine oder Waschmaschine) besteht in der Grundstufe aus einem zweistufigen Eingangspuffer für Werkstücke, einer Bearbeitungseinheit und einem einstufigen Ausgangspuffer. In der Ausbaustufe enthält jede WZM einen zweistufigen Ausgangspuffer.
Um die einfache Konfigurierbarkeit des Systems aufrecht zu halten, verfügen die WZM über kein Wissen bezüglich des Prozesses des Fertigungssystems. Sie beschränken sich auf das Senden (Broadcast) ihres Maschinenstatus und die Ausführung ihres jeweiligen Bearbeitungsschrittes. Ein Status wird immer dann gesendet, sobald sich die Belegung des Eingangs- oder Ausgangspuffers ändert oder ein leerer Eingangspuffer bzw. voller Ausgangspuffer die WZM blockiert.

Aufgaben	Eingabe	Verarbeitung	Ergebnis
Bearbeitung (Bohren, Fräsen, Waschen)	Unfertiges Werkstück	Bearbeitungsschritt ausführen	Bearbeitetes Werkstück
Status der Maschine senden	Maschinendaten	Ermittlung und Versendung des Maschinenstatus	Gesendeter Maschinenstatus
Verbringung der Werkstücke	Werkstück	Transport vom Eingangspuffer auf den Bearbeitungsplatz der WZM oder von diesem auf den Ausgangspuffer	Anzahl der Werkstücke im Puffer sind um 1 erhöht bzw. verringert

Tabelle 10.2: Aufgabenbeschreibung der WZM

Maschinenfunktion		Dauer [s]
Verbringen des Werkstückes vom Eingangspuffer auf den Bearbeitungsplatz bzw. vom Bearbeitungsplatz auf den Ausgangspuffer		2
Ausführung des Bearbeitungsschrittes	Fräsen der Gussgrate (Fünf-Achsen-Fräsmaschine)	30
	Glätten der Bohrungen (Drei-Achsen-Fräsmaschine)	20
	Auswaschen der Metallspäne	35

Tabelle 10.3: Zeitliche Spezifikationen der WZM

10.2.3 Aufgabenbeschreibung der Batterieladestation BL (Ausbaustufe*)*

Die Batterieladestation verfügt über keine zu spezifizierende Steuerungssoftware. Der Ladevorgang wird vom HTF über seine Kommunikationseinrichtung gesteuert. Die Batterieladestation ist nur in der Ausbaustufe der Referenzfallstudie Teil des Fertigungssystems.

Aufgaben	Eingabe	Verarbeitung	Ergebnis
Aufladen der Batterie eines HTF	HTF in Ladeposition	Laden der Batterie bis zur Maximalkapazität	Vollständig geladene Batterien des HTF

Tabelle 10.4: Aufgabenbeschreibung der BL

Maschinenfunktion	Dauer [min]
Maximale Ladedauer an der Batterieladestation bei entleerter Batterie	20

Tabelle 10.5: Zeitliche Spezifikation der BL

10.2.4 Aufgabenbeschreibung des Blockstrecken-Management-Systems BMS

Das BMS ist ein zentraler Prozess, der lediglich über eine Kommunikationseinrichtung verfügt. Er wird nur in der Ausbaustufe der Referenzfallstudie eingesetzt.

Aufgaben	Eingabe	Verarbeitung	Ergebnis
Reservierung und Freigabe von Blockstrecken für HTF	Reservierungswunsch eines HTF	Prüfen des Reservierungswunsches, Reservieren der Strecke oder Zurückweisung des Reservierungswunsches	Reservierte Blockstrecke oder zurückgewiesener Reservierungswunsch

Tabelle 10.6: Aufgabenbeschreibung des BMS

10.2.5 Aufgabenbeschreibung des Lagers EIN

Das Lager verfügt über eine unbegrenste Anzahl von Lagerplätzen für Werkstücke (inklusive der Werkstückträger). Zudem ist ein einstufiger Ausgangspuffer vorhanden. Werkstücke werden entsprechend dem Werkstücktyp nach Anforderung durch ein HTF aus einem Lagerplatz in den Ausgangspuffer verbracht, wo sie vom HTF entnommen werden.

Aufgaben	Eingabe	Verarbeitung	Ergebnis
Positionierung des Werkstückes	Werkstück und Anforderung des Werkstücktyps durch HTF	Verbringen eines Werkstückes auf den Ausgangspuffer des Lagers	Anzahl der Werkstücke im Lager um 1 verringert

Tabelle 10.7: Aufgabenbeschreibung des EIN

Aufgabe/Maschinenfunktion	Dauer [s]
Positionierung des Werkstückes im Ausgangspuffer (abhängig vom Lagerplatz)	5 - 10

Tabelle 10.8: Zeitliche Spezifikationen des EIN

10.2.6 Aufgabenbeschreibung des Lagers AUS

Das Lager AUS verfügt über eine unbestimmte Anzahl von Lagerplätzen für Werkstücke (inklusive der Werkstückträger). Zudem ist ein einstufiger Eingangspuffer vorhanden. Die HTF transportieren die fertig entgrateten Werkstücke in das AUS, von wo sie zur Montage in nachgeschaltete Fertigungssysteme transportiert werden. Darüber hinaus ist die Verwaltung der aktiven HTF des Fertigungssystems und die Überwachung des Tages-Solls im AUS integriert.

Aufgaben	Eingabe	Verarbeitung	Ergebnis
Verbringung des Werkstückes	Werkstück	Verbringen eines Werkstückes vom Eingangspuffer in die Lagerposition	Anzahl der Werkstücke erhöht sich um 1
Beendigung der Fertigung	Anzahl der entgraten Werkstücke, Tages-Soll	Prüfen der Anzahl der entgraten Werkstücke ≥ Tages-Soll	Beendigung der Fertigung
An- und Abmelden eines HTF	Telegramm, Liste der aktiven HTF	Registrierung oder Austragung des HTF	HTF an- oder abgemeldet und Vorgang gesendet

Tabelle 10.9: Aufgabenbeschreibung des AUS

Aufgabe/Maschinenfunktion	Dauer [s]
Verbringung des Werkstückes auf einen Lagerplatz (abhängig vom Lagerplatz)	5 - 10
An- und Abmelden eines HTF	max. 1

Tabelle 10.10: Zeitliche Spezifikationen des AUS

10.2.7 Aufgabenbeschreibung der HTF

Ein HTF besteht aus einer Bewegungskinematik (Antrieb und Lenkung), einem aktiven LAM (Roboterarm mit einer Arretierung für den Werkstückträger) und einer sensorischen Einrichtung zur Lokalisierung und Hinderniserkennung. Die Zusammensetzung der Sensoren und Aktoren sowie ihre Ansteuerung ist für die Steuerung des HTF transparent (siehe 3.1). Die maximale Fahrgeschwindigkeit des HTF beträgt 1.5 m/s und die Beschleunigung beim Anfahren und Bremsen wird mit 0.7 m/s^2 angenommen. Zusätzlich verfügt ein HTF über die Fähigkeit, in den Energiesparmodus zu schalten. In diesem Modus sind alle Geräte des HTF auf minimalen Energieverbrauch geschaltet und nicht betriebsbereit. Lediglich die Kommunikationseinrichtung ist hierbei betriebsbereit.

Ein HTF verfügt über Prozesswissen, da ihm die Richtung und Reihenfolge des Materialflusses innerhalb des Fertigungssystems bezogen auf einen Werkstücktyp bekannt ist (Produktionsdatenbank). Auch die aktuellen Zustände der Maschinen und ihre Positionen im Fertigungssystem werden in der Produktionsdatenbank abgelegt.

In der Grundstufe der Referenzfallstudie ist die Anzahl der HTF auf drei begrenzt. In der Ausbaustufe ist die Anzahl der aktiven HTF dynamisch, d.h. HTF aus dem Parkbereich melden sich am AUS an.

Ebenfalls gilt für die Ausbaustufe, dass ein HTF nach Beendigung eines Transportauftrages die Position zur Werkstückübergabe bzw. –entnahme verlässt, um eine bestimmte, nächstliegende Warteposition anzufahren.

Aufgaben	Eingabe	Verarbeitung	Ergebnis
Werkstückentnahme	Werkstück im Ausgangspuffer der WZM oder des EIN	Anforderung an EIN oder WZM, Transport des Werkstückes vom Ausgangspuffer der WZM oder des EIN auf HTF	Werkstück im LAM
Werkstückübergabe	Werkstück	Anforderung an WZM oder AUS, Transport des Werkstückes vom LAM des HTF auf Eingangspuffer der WZM oder des AUS	Werkstück im Eingangspuffer der WZM oder des AUS
Senden eines Angebotes	Position des HTF	Berechnung des Angebotes	Telegramm mit Angebot gesendet
Moderation eines Transportauftrages	Produktionsdatenbank	Berechnung des Transportauftrages (Holauftrag), Angebot senden, Verhandlung beenden und Zuschlag geben	Verhandelter Transportauftrag
Herunterfahren des HTF	-	Auftrag zu Ende ausführen und in Energiesparmodus gehen.	HTF im Energiesparmodus
Neustart der Sensorik/Aktorik	Störung im Antrieb	Die Sensorik und Aktorik wird nach einer Fehlfunktion im Antrieb neu gestartet	HTF betriebsbereit

Tabelle 10.11: Aufgabenbeschreibung eines HTF

Aufgabe/Maschinenfunktion	Dauer [s]
Werkstückentnahme oder Werkstückübergabe	3
Verhandlungsdauer (durch den Moderator bestimmt)	durchschnittlich 1.5
Neustart der Sensorik/Aktorik	30

Tabelle 10.12: Zeitliche Spezifikationen eines HTF

10.3 Ablaufbeschreibung (Fertigungssystem)

10.3.1 Regulärer Betrieb

Nach der Initialisierung des Fertigungssystems und Anmeldung der HTF beginnen die Maschinen ihren Status zu senden. Ein HTF (Rolle des Moderators) erkennt den Transportauftrag (Hol-Auftrag) und sendet seine Auftragsausschreibung. Nach einer festgelegten Zeitspanne beendet der Moderator die Verhandlung und erteilt den Zuschlag. Die Ausführung eines Transportauftrages beinhaltet eine Hol- und eine Bring-Fahrt. Während der Transportfahrt nimmt ein HTF nicht an der Verhandlung teil. Die Fertigung endet bei Erreichung des Tages-Solls.

Die Detaillierung der Ablaufbeschreibung des Fertigungssystems ist in der folgenden Tabelle 9 beschrieben:

Szenarien	Vorbedingungen	Schritte	Nachbedingungen
Initialisierung des Fertigungssystems **(1)**	HTS ist im Energiesparmodus, HTF haben keine Werkstücke geladen, WZM, AUS und EIN sind gestoppt.	Ein externes Leitsystem sendet ein Telegramm (Aktion: START) zum Start der Fertigung in dem Fertigungssystem. HTS, d.h. die einzelnen HTF, fährt aus dem Energiesparmodus in die Betriebsbereitschaft hoch. WZM, AUS und EIN werden gestartet.	HTS und WZM sind betriebsbereit
Anmeldung eines HTF an AUS **(2)**	WZM, HTS, EIN und AUS sind betriebsbereit, mindestens ein HTF ist noch nicht angemeldet und verfügt über Produktionsdatenbank des Fertigungssystems	1. Ein HTF meldet sich per Telegramm (Aktion: ANMELDUNG) bei AUS an. 2. Nach der Registrierung wird dem HTF die Meldung per Telegramm (Aktion: NMELDUNG_OK) bestätigt. 3. Antwortet das AUS nicht, wird die Anmeldung vom HTF wiederholt → Schritt 1. 4. (Es kann sich stets nur ein HTF zur selben Zeit anmelden)	HTF ist angemeldet
Abmeldung eines HTF beim AUS **(3)**	-	1. HTF entscheidet aufgrund seiner Produktionsdatenbank und seiner implementierten Regeln, dass zu viele HTF im Fertigungssystem angemeldet sind. 2. HTF sendet ein Telegramm zur Abmeldung (Aktion: ABMELDUNG). Das AUS bestätigt dies mit einem entsprechenden Telegramm (Aktion: ABMELDUNG_OK). 3. Das HTF fährt in den Parkbereich (PB) und geht dort in den Energiesparmodus.	Anzahl der HTF im PB erhöht sich um 1
Auftragsverhandlung **(4)**	Mindestens ein HTF ist angemeldet und betriebsbereit (Gilt für die Grundstufe immer)	1. Senden des Maschinenstatus per Telegramm (Aktion: WZM_STATUS), dieses wird nur vom HTS ausgewertet. 2. Ein HTF erkennt aus dem Maschinenstatus einen Transportauftrag. 3. Das HTF übernimmt die Rolle des Moderators und sendet den erkannten Transportauftrag verbunden mit dem eigenen Positions-gebundenen Kostenfaktor per Telegramm (Aktion: AUFTRAG) 4. Moderator wartet auf bessere Angebote, d.h. Kostenfaktoren der anderen HTF per Telegramm (Aktion: ANGEBOT) und beendet die Verhandlung nach ca. 1.5 Sekunden mit dem expliziten Versenden des Zuschlages. Andere HTF antworten nur bei besserem Angebot.	Transportauftrag verhandelt und einem HTF zugeteilt

HTF fährt **(5)**	HTF hat einen Transportauftrag oder einen Fahrauftrag (z.B. zur BL in der Ausbaustufe)	1. Übergeben des Fahrzieles (Maschinenkennung) an Steuerung des HTF. In der Ausbaustufe muss vorher am BMS die geplante Strecke reserviert werden und ggf. eine Alternative berechnet und reserviert werden. 2. Steuerung signalisiert das Erreichen des Zieles.	HTF ist an Zielposition
Werkstückübergabe oder –entnahme **(6)**	Werkstückentnahme: HTF steht an Werkstückentnahmeeinheit einer WZM/EIN, mindestens ein Werkstück ist im Ausgangspuffer	Werkstückentnahme: 1. HTF sendet an WZM/EIN Telegramm (Aktion: WS_ENT), dies enthält auch die Information zum Werkstücktyp. 2. WZM/EIN sendet ein Telegramm zur Bestätigung (Aktion: WS_OK). EIN verbringt davor das entsprechende Werkstück auf seinen Ausgangspuffer. 3. LAM des HTF verbringt das Werkstück auf seine Halterung. Danach sendet das HTF ein Telegramm (Aktion: WS_ENDE) zum Abschluss der Entnahme.	Werkstückentnahme: Im Ausgangspuffer der WZM oder des EIN ist die Anzahl der Werkstücke um 1 reduziert
(7)	Werkstückübergabe: HTF steht an der Werkstückübergabeeinheit einer WZM oder AUS, mindestens ein Platz im Eingangspuffer ist frei	Werkstückübergabe: 1. HTF sendet an WZM/AUS ein Telegramm (Aktion: WS_ÜB). 2. WZM/AUS sendet ein Telegramm (Aktion: WS_OK). 3. LAM des HTF verbringt das Werkstück in den Eingangspuffer von WZM/AUS. 4. HTF sendet ein Telegramm (Aktion: WS_ENDE) zum Abschluss der Übergabe.	Werkstückübergabe: Im Eingangspuffer der WZM oder des AUS ist die Anzahl der Werkstücke um 1 erhöht
Eine WZM bearbeitet Werkstück **(8)**	Werkstück im Eingangspuffer einer WZM, ein Platz im Ausgangspuffer frei	1. WZM holt Werkstück aus dem Eingangspuffer auf den Bearbeitungsplatz. Ist kein Werkstück vorhanden, wird der Maschinenstatus gesendet. Dieser wird im 3 Sekunden Abstand wiederholt gesendet, bis ein Werkstück im Eingangspuffer ist. 2. Ausführen des entsprechenden Arbeitsschrittes am Werkstück. 3. Verbringen des Werkstückes auf den Ausgangspuffer. Ist dieser blockiert, so wird ebenfalls alle 3 Sekunden der Maschinenstatus gesendet.	Anzahl der Werkstücke im Ausgangspuffer um 1 erhöht
Beendigung der Fertigung **(9)**	Tages-Soll erreicht	1. AUS sendet Telegramm zur Beendigung der Fertigung (Aktion: FERT_ENDE) 2. HTS empfängt Telegramm und jedes HTF meldet sich beim AUS per Telegramm (Aktion: ABMELDUNG) ab. AUS sendet die Bestätigung (Aktion: ABMELDUNG_OK). HTF, die noch ein Werkstück transportieren, führen den Transportauftrag zu Ende aus und melden sich dann ab. WZM führen den aktuellen Bearbeitungsschritt zu Ende aus. 3. HTS geht in den Energiesparmodus. WZM stoppen.	Fertigung beendet

Tabelle 10.13: Beschreibung der Szenarien des Fertigungssystems im regulärem Betrieb (*Aktion: XYZ* bezieht sich auf die Struktur der Broadcast-Telegramme, siehe Tabelle 9.15)

10.3.2 Irregulärer Betrieb

Irregulärer Betrieb definiert sich durch die Störung einer Maschine, worunter auch das Laden der Batterie einer HTF fällt.

Szenario	Vorbedingungen	Schritte	Nachbedingungen
Einfacher Störfall **(1)**	HTF hat eine Störung im Antrieb und kein Werkstück geladen	1. Neustart der Sensorik und Aktorik. Während dieser Zeit wird vom HTF an keiner Verhandlung teilgenommen und ein aktuell vorhandener Transportauftrag gelöscht. Es wird in der Grundstufe davon ausgegangen, dass das HTF an keiner kritischen Position steht (Werkstückübergabe- oder –entnahmeposition). 2. Nach ca. 30 Sekunden melden die HTF ihre Betriebsbereitschaft und HTF kann wieder an den Verhandlungen teilnehmen.	HTF ist betriebsbereit und angemeldet
Erweiterter Störfall (Ausbaustufe) **(2)**	HTF hat eine Störung im Antrieb, HTF kann an einer kritischen Position stehen	1. Gestörtes HTF sendet ein Telegramm (Aktion: STÖRUNG), welches die Information der Störung des HTF und die aktuelle Position enthält. HTS wertet dieses Telegramm aus und berücksichtigt es bei den Verhandlungen. War das HTF auf einer Holfahrt eines Transportauftrages, wird der Auftrag neu ausgeschrieben mit dem gestörten HTF als Moderator. 2. Neustart der Sensorik und Aktorik des HTF. Während dieser Zeit wird an keiner Verhandlung teilgenommen. 3. Nach ca. 30 Sekunden meldet das HTF seine Betriebsbereitschaft per Telegramm (Aktion: BEREIT) und HTF kann wieder an den Verhandlungen teilnehmen bzw. seinen Transportauftrag zu Ende ausführen.	HTF ist betriebsbereit und angemeldet
Störung einer WZM – Werkzeugbruch (Ausbaustufe) **(3)**	Eine WZM hat einen Werkzeugbruch	1. WZM sendet Telegramm mit Störungsmeldung (Aktion: STÖRUNG) und HTS wertet es aus. 2. WZM wird nicht freigeräumt. Werkstücke verbleiben in den Puffern und am Bearbeitungsplatz. 3. Servicetechniker beheben den Fehler durch Werkzeugwechsel. 4. WZM sendet Telegramm mit Betriebsbereitschaft (Aktion: BEREIT) und danach den Maschinenstatus (Aktion, STATUS).	WZM betriebsbereit
Laden der Batterie (Ausbaustufe) **(4)**	Batterieladestand eines HTF kritisch	1. HTF prüft Batterieladestand. 2. Batterieladestand ist kritisch und HTF sendet Anmeldetelegramm an BL. 3. BL bestätigt mit Telegramm und ist damit für andere HTF verriegelt. 4. Weist BL die Anmeldung zurück, geht HTF für 10 min in den Energiesparmodus. Danach wird eine erneute Anmeldung gestartet →Schritt 2. 5. HTF fährt zur BL. 6. HTF startet Ladevorgang per Telegramm (Aktion: LADE). 7. HTF prüft die Ladung und beendet den Ladevorgang bei maximaler Ladung (Aktion: LADE_OK).	HTF ist vollständig geladen

Tabelle 10.14: Beschreibung der Szenarien des Fertigungssystems im irregulärem Betrieb (Aktion: XYZ bezieht sich auf die Struktur der Broadcast-Telegramme, siehe Tabelle 9.15)

10.4 Datendarstellung und Mengengerüst

10.4.1 Kommunikationsdaten

Datentyp	Struktur	Einheit
Maschinen-kennung	Kennung (Grundstufe)	[EIN, AUS, 3-FRÄS, 5-FRÄS, WASCH, HTF1..3]
	Kennung (Ausbaustufe)	[EIN, AUS, 3-FRÄS1..2, 5-FRÄS1..2, WASCH1..2, HTF1..6]
Maschinenstatus	Belegung des Eingangspuffers	[BELEGT, FREI]
	Belegung des Bearbeitungsplatzes	[BELEGT, FREI]
	Belegung des Ausgangspuffers	[BELEGT, FREI]
Broadcast-Telegramm	Absenderkennung	Maschinenkennung
	Aktion	[START, ANMELDUNG, ANMELDUNG_OK, WS_ÜB, WS_ENT, WS_OK, LADE, LADE_OK, WS_ENDE, AUFTRAG, WZM_STATUS, FERT_ ENDE, ABMELDUNG, ABMELDUNG_OK, ANGEBOT, STÖRUNG, BEREIT]
	Empfänger	[*Maschinenkennung*, ALLE_HTF, ALLE]
Transportauftrag (Hol-Auftrag)	Transportauftrags-Identifizierung	[ohne Einheit]
	Quelle	Maschinenkennung
	Werkstück	(Siehe Prozessdaten)
	Senke	Maschinenkennung
Angebot	Transportauftrag	[ohne Einheit]
	Kostenfaktor	[ohne Einheit]
Auftrag	Moderator	Maschinenkennung
	Transportauftrag	[ohne Einheit]
	Kostenfaktor (Moderator)	[ohne Einheit]

Tabelle 10.15: Beschreibung der Kommunikationsdaten des Fertigungssystems (kursive Bezeichnungen sind Referenzen auf bereits definierte Datentypen)

10.4.2 Prozessdaten

Datentyp	Struktur	Einheit
Werkstück	Kennung	[ohne Einheit]
	Bearbeitungsstatus	[unbearbeitet, gebohrt, gefräst, gewaschen]
	Reihenfolge der Bearbeitung: V6-Motor Kurbelwelle	konstant [IN-3-FRÄS-5-FRÄS-WASCH-AUS] [IN-5-FRÄS-WASCH-AUS]
Tages-Soll	Maximale Anzahl der zu entgratenden Werkstücke pro Tag	[ohne Einheit]
Kostenfaktor	Wert	[ohne Einheit]
Batterieladung	Ladestand	[%]
Position	x-Koordinate	[ohne Einheit]
	y-Koordinate	[ohne Einheit]
Produktionsdaten-bank	Enthält alle Positionen von Maschinen und sonstigen Einrichtungen des Fertigungssystems, die Richtung und Reihenfolge des Materialflusses	-
Liste der aktiven HTF	Listennummer	[ohne Einheit]
	Kennung	*[Maschinenkennung*]

Tabelle 10.16: Beschreibung der Prozessdaten des Fertigungssystems (kursive Bezeichnungen sind Referenzen auf bereits definierte Datentypen)

10.5 Schnittstellen

Alle Maschinen (WZM, HTS, EIN, AUS) verfügen über eine Steuerung, die es zu spezifizieren gilt, und eine weitgehend transparente Systemtechnik, bestehend aus Sensoren zur Navigation und Hinderniserkennung und Aktoren, zum Antrieb, zur Lenkung und zum Werkstück-Handling. Die Schnittstelle einer Steuerung zur Systemtechnik wird Systemschnittstelle genannt.

Darüber hinaus sind die Maschinen in der Lage, miteinander Kommunikationstelegramme auszutauschen. Dies geschieht über eine Kommunikationsschnittstelle. In der für die Referenzfallstudie verfügbare*n 3-D Animation* ist es möglich, mittels einer definierten Animationsschnittstelle die Animation des Fertigungssystems zu steuern [DFG_98]. Die Animationsschnittstelle umfasst aber nur die Summe der Systemschnittstellen der Maschinen. Die Tabellen 9.17 bis 9.20 beschreiben die Systemschnittstellen der Maschinen. Die Batterieladestation und der Parkbereich enthalten keine zu spezifizierende Steuerungssoftware und weisen somit auch keine Systemschnittstelle auf.

10.5.1 Systemschnittstelle des HTF

Systemfunktionen	Parameter	Rückgabewert	Animationsschnittstelle
Fahren in absoluter Richtung	Relativer Winkel [Grad] Fahrweg [cm] Geschwindigkeit [km/h]	Bestätigung der Funktion [OK, FEHLER]	*AgvStartMoveAbs* (asynchron) *agvPerformMoveAbs*
Fahrt in relativer Richtung	Relativer Winkel [Grad] Fahrweg [cm] Geschwindigkeit [km/h]	Bestätigung der Funktion [OK, FEHLER]	*AgvStartMoveRel* (asynchron) *agvPerformMoveRel*
Anhalten	-	Bestätigung der Funktion [OK, FEHLER]	agvStopMoving
Werkstückentnahme oder –übergabe starten	Richtung [Übergabe, Entnahme]	Bestätigung der Funktion [OK, FEHLER]	AgvStartHandOver (asynchron) agvPerformHandOver
Aufladen der Batterie	-	Bestätigung der Funktion [OK, FEHLER]	AgvStartRechargeBattery (asynchron) agvPerformRechargeBattery
In den Energiesparmodus schalten	-	-	*Kein Kommando notwendig (es wird nur Energie bei Bewegungen verbraucht)*
Neustart der Sensrik / Aktorik	-	Bestätigung der Funktion [OK, FEHLER]	*agvRestart*
Status lesen	-	Zustand des HTF [WARTEN, FAHREN, WS_ÜBER, WS_ENT, LADEN, FEHLER]	agvGetState
Batterieladestand lesen	-	*Batterieladung* [%]	*agvGetBatteryLoad*
Messen der Entfernung zum nächsten Hindernis	-	Entfernung [cm]	agvGetNextObstacle
Messen der aktuellen Position	-	x-Position [cm] y-Position [cm]	agvGetPosition
Messen der aktuellen Ausrichtung	-	Absolute Ausrichtung [Grad]	agvGetDirection

Tabelle 10.17: System- und Animationsschnittstelle des HTF

10.5.2 Systemschnittstelle der WZM

Systemfunktionen	Parameter	Rückgabewert	Funktion der Animations-schnittstelle
Starten der WZM	-	Bestätigung der Funktion [OK, FEHLER]	*mtStart*
Werkstück aus dem Eingangspuffer holen und bearbeiten	-	Bestätigung der Funktion [OK, FEHLER]	*mtStartWork* *mtPerformWork*
Anhalten der WZM	-	Bestätigung der Funktion [OK, FEHLER]	*mtStopWorking*
Wiederanlaufen nach Maschinenstopp	-	Bestätigung der Funktion [OK, FEHLER]	*mtRestart*
Lesen der Anzahl der freien Plätze im Eingangspuffer	-	Anzahl [0..2]	*mtGetIInputBufferNumber*
Lesen der Anzahl der belegten Plätze im Ausgangspuffer	-	Anzahl [0..2]	*mtGetOutputBufferNumber*

Tabelle 10.18: System- und Animationsschnittstelle der WZM

10.5.3 Systemschnittstelle des Lagers EIN

Systemfunktionen	Parameter	Rückgabewert	Funktion der Animationsschnittstelle
Start des Eingangslagers	-	Bestätigung der Funktion [OK, FEHLER]	*inStart*
Ein Werkstück auf die Entnahmeposition verbringen	Werkstücktyp	Bestätigung der Funktion [OK, FEHLER]	*inStartSupplyItem* *inPerformSupplyItem*
Anhalten des Eingangslagers	-	Bestätigung der Funktion [OK, FEHLER]	*inStopWorking*
Wiederanlaufen des Eingangslagers	-	Bestätigung der Funktion [OK, FEHLER]	*inRestart*
Lesen des Status	-	Maschinenstatus [WARTEN, ARBEITEN, ANGEHALTEN, FEHLER]	*inGetState*

Tabelle 10.19: System- und Animationsschnittstelle des EIN

10.5.4 Systemschnittstelle des Lagers AUS

Systemfunktionen	Parameter	Rückgabewert	Funktion der Animations-schnittstelle
Start des Ausgangslager	-	Bestätigung der Funktion [OK, FEHLER]	*outStart*
Anhalten des Ausgangslagers	-	Bestätigung der Funktion [OK, FEHLER]	*outStopWorking*
Wiederanlaufen des Ausgangslagers	-	Bestätigung der Funktion [OK, FEHLER]	*outStart*
Lesen des Status	-	Maschinenstatus [WARTEN, ARBEITEN, ANGEHALTEN, FEHLER]	*outGetState*

Tabelle 10.20: System- und Animationsschnittstelle des AUS