Lehrstuhl für
Werkzeugmaschinen und Fertigungstechnik
der Technischen Universität München

Modellgetriebene Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme

Clemens Pörnbacher

Vollständiger Abdruck der von der Fakultät für Maschinenwesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. G. Reinhart

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. M. Zäh
2. Univ.-Prof. Dr.-Ing. B. Vogel-Heuser

Die Dissertation wurde am 01.03.2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 17.06.2010 angenommen.

Clemens Pörnbacher

Modellgetriebene Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme

Herbert Utz Verlag · München

Forschungsberichte IWB

Band 248

Zugl.: Diss., München, Techn. Univ., 2010

Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN 978-3-8316-4108-6

Printed in Germany
Herbert Utz Verlag GmbH, München
089-277791-00 · www.utzverlag.de

Geleitwort der Herausgeber

Die Produktionstechnik ist für die Weiterentwicklung unserer Industriegesellschaft von zentraler Bedeutung, denn die Leistungsfähigkeit eines Industriebetriebes hängt entscheidend von den eingesetzten Produktionsmitteln, den angewandten Produktionsverfahren und der eingeführten Produktionsorganisation ab. Erst das optimale Zusammenspiel von Mensch, Organisation und Technik erlaubt es, alle Potentiale für den Unternehmenserfolg auszuschöpfen.

Um in dem Spannungsfeld Komplexität, Kosten, Zeit und Qualität bestehen zu können, müssen Produktionsstrukturen ständig neu überdacht und weiterentwickelt werden. Dabei ist es notwendig, die Komplexität von Produkten, Produktionsabläufen und -systemen einerseits zu verringern und andererseits besser zu beherrschen.

Ziel der Forschungsarbeiten des *iwb* ist die ständige Verbesserung von Produktentwicklungs- und Planungssystemen, von Herstellverfahren sowie von Produktionsanlagen. Betriebsorganisation, Produktions- und Arbeitsstrukturen sowie Systeme zur Auftragsabwicklung werden unter besonderer Berücksichtigung mitarbeiterorientierter Anforderungen entwickelt. Die dabei notwendige Steigerung des Automatisierungsgrades darf jedoch nicht zu einer Verfestigung arbeitsteiliger Strukturen führen. Fragen der optimalen Einbindung des Menschen in den Produktentstehungsprozess spielen deshalb eine sehr wichtige Rolle.

Die im Rahmen dieser Buchreihe erscheinenden Bände stammen thematisch aus den Forschungsbereichen des *iwb*. Diese reichen von der Entwicklung von Produktionssystemen über deren Planung bis hin zu den eingesetzten Technologien in den Bereichen Fertigung und Montage. Steuerung und Betrieb von Produktionssystemen, Qualitätssicherung, Verfügbarkeit und Autonomie sind Querschnittsthemen hierfür. In den *iwb* Forschungsberichten werden neue Ergebnisse und Erkenntnisse aus der praxisnahen Forschung des *iwb* veröffentlicht. Diese Buchreihe soll dazu beitragen, den Wissenstransfer zwischen dem Hochschulbereich und dem Anwender in der Praxis zu verbessern.

Gunther Reinhart *Michael Zäh*

Meinen Eltern Cecilia und Franz Pörnbacher für ihre stete und uneingeschränkte Unterstützung gewidmet

Inhaltsverzeichnis

Symbolverzeichnis

Große lateinische Buchstaben

Symbol	*Einheit*	*Bedeutung*
A	-	*Menge der Ausgangssignale eines Petri-Netzes*
A_i	m^2	*Fläche i*
AK	-	*Menge der Aktionen eines Petri-Netzes*
C	-	*Kapazität einer Stelle eines Petri-Netzes*
C, C_i	F	*Elektrische Kapazität, Elektrische Kapazität i*
E	-	*Menge der Eingangssignale eines Petri-Netzes*
$F(t)$	N	*Kraft*
K	-	*Menge der Kanten eines Petri-Netzes*
K_i	-	*Kante i eines Petri-Netzes*
L, L_i	H	*Induktivität, Induktivität i*
M_0^d	-	*Anfangsmarkierung eines Petri-Netzes*
$\dot{Q}_i$	m^3/s	*Volumenstrom i*
R, R_i	Ω	*Elektrischer Widerstand, Elektrischer Widerstand i*
S	-	*Menge der Stellen eines Petri-Netzes*
S_i	-	*Stelle i eines Petri-Netzes*
SB	-	*Menge der Schaltbedingungen eines Petri-Netzes*
$SB(T_i)$	-	*Zur Transition T_i gehörige Schaltbedingung eines Petri-Netzes*
SV	-	*Systemfunktion*
T	-	*Menge der Transitionen eines Petri-Netzes*
T_i	-	*Transition i eines Petri-Netzes*
U, U_i	V	*Elektrische Spannung, Elektrische Spannung i*
W	-	*Gewicht einer Kante eines Petri-Netzes*
X^d	-	*Menge der diskreten Eingangssignale (Eingabemenge)*
Y^d	-	*Menge der diskreten Ausgangssignale (Ausgabemenge)*
Z	-	*Zustand*
Z_0^d	-	*Anfangszustand*
Z^d	-	*Diskrete Zustandsmenge*
$Z_k^d(n)$	-	*Diskreter Zustand k*
Z^k	-	*Kontinuierliche Zustandsmenge*
$Z_k^k(t)$	-	*Kontinuierlicher Zustand k*
$\dot{Z}_k^k(t)$	-	*Differentialquotient des kontinuierlichen Zustands k*

Kleine lateinische Buchstaben

Symbol	*Einheit*	*Bedeutung*
a	*m/s²*	*Beschleunigung*
a_i	*-*	*Ausgangssignal i eines Signalinterpretierten Petri-Netzes*
c	*N/m*	*Steifigkeit*
d	*s^{-1}*	*Dämpfungskonstante*
e_i	*-*	*Eingangssignal i eines Signalinterpretierten Petri-Netzes*
f	*-*	*Zustandsübergangsfunktion*
$f(t)$	*-*	*Zeitabhängige Funktion*
$f(n)$	*-*	*Diskrete Funktion*
g	*-*	*Ausgabefunktion*
m	*kg*	*Masse*
n	*-*	*Diskreter Zeitpunkt n*
$p_i(t)$	*N/m²*	*Druck i*
$s(t)$	*m*	*Weg allgemein*
$s(t)$	*-*	*Zeitabhängiges Signal*
$s(n)$	*-*	*Diskretes Signal*
$\dot{s}$	*m/s*	*Geschwindigkeit*
$\ddot{s}$	*m/s²*	*Beschleunigung*
s_{max}	*m*	*Maximale Position*
s_{ist}	*m*	*Aktuelle Position*
t	*s*	*Zeit*
v	*m/s*	*Geschwindigkeit*
x_i	*-*	*Eingangsgröße i*
$x_i^k(t)$	*-*	*Kontinuierliche Eingangsgröße i*
$x_i^d(n)$	*-*	*Diskrete Eingangsgröße i*
$\vec{x}^k$	*-*	*Eingangsvektor*
y_j	*-*	*Ausgangsgröße j*
$y_j^d(n)$	*-*	*Diskrete Ausgangsgröße j*
$y_j^k(t)$	*-*	*Kontinuierliche Ausgangsgröße j*
$\vec{y}^k$	*-*	*Ausgangsvektor*
z_k	*-*	*Zustandsgröße k*
$z_k^k(t)$	*-*	*Kontinuierliche Zustandsgröße k*
$z_k^d(n)$	*-*	*Diskrete Zustandsgröße k*
$\vec{z}^k$	*-*	*Zustandsvektor*
$\dot{z}_k^k$	*-*	*Ableitung der Zustandsgröße k*

Große griechische Buchstaben

Symbol	*Einheit*	*Bedeutung*
Δt	*s*	*Schrittweite eines Zeitinkrements*
Σ	-	*Summe*

Allgemeine Symbole

Symbol	*Einheit*	*Bedeutung*
$\mathbb{N}$	-	*Menge der natürlichen Zahlen*
$\mathbb{R}$	-	*Menge der reellen Zahlen*
$\mathbb{Z}$	-	*Menge der ganzen Zahlen*
$A \cup B$	-	*Vereinigungsmenge*
$A \cap B$	-	*Schnittmenge*
$A \subseteq B$	-	*Teilmenge*
$A \times B$	-	*Kartesisches Produkt*
$\setminus$	-	*Differenz zweier Mengen*
$\in$	-	*Element einer Menge*
$f: A \to B$	-	*Abbildung*
$=$	-	*Gleichheitsoperator*
$:=$	-	*Zuweisungsoperator*
$\wedge$	-	*Logischer Operator "und"*
$\vee$	-	*Logischer Operator "oder"*
$\neg$	-	*Negation einer Aussage*
$\overline{x_i}$	-	*Negation der Variablen* x_i

Abkürzungsverzeichnis

Symbol	*Bedeutung*
AC-MDSD	Architecture Centric Model Driven Software Development
AR	Augmented Reality
ACSL	Advanced Continuous Simulation Language
AS	Ablaufsprache
AS-I	Actuator Sensor Interface
AWL	Anweisungsliste
BDI	Bundesverband der Deutschen Industrie
BMBF	Bundesministerium für Bildung und Forschung
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CPU	Central Processing Unit
DAE	Differential-Algebraic Equation
DSL	Domain Specific Language
DIN	Deutsches Institut für Normung
FC	Funktion
FBS	Funktionsbausteinsprache
FEM	Finite-Elemente-Methode
FWF	Forschungsvereinigung Werkzeugmaschinen und Fertigungstechnik
GMA	Gesellschaft für Mess- und Automatisierungstechnik
GRAFCET	Graphe Fonctionnel de Commande Etapes-Transitions
GSD	Gerätestammdatei
HDN	Hybrides Dynamisches Netz
IEC	International Electrotechnical Commission
IFTMM	International Federation for the Theory of Machines and Mechanism
INCOSE	International Council On Systems Engineering
IPC	Industrie-PC
IRDAC	Industrial Research and Development Advisory Committee of the European Union
IRL	Industrial Robot Language
ISO	International Organization for Standardization
IT	Information Technology
iwb	Institut für Werkzeugmaschinen und Betriebswissenschaften

JSPMI	Japan Society for the Promotion of Machine Industry
KOP	Kontaktplan
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MDSD	Model Driven Software Development
MKS	Mehrkörpersimulation
MSC	Message Sequence Chart
NC	Numerical Control
OCL	Object Constraint Language
ODE	Ordinary Differential Equation
OMG	Object Management Group
OMT	Object Modeling Technique
PC	Personal Computer
PDM	Produktdatenmanagement
PIM	Platform Independent Model
PLC	Programmable Logic Control
PM	Physikalisches Modell
PSM	Platform Specific Model
ROOM	Real-Time Object Oriented Modeling
SDL	System Description Language
SIPN	Signalinterpretiertes Petri-Netz
SM	Softwaremodell
SPS	Speicherprogrammierbare Steuerung
ST	Strukturierter Text
STEP	Standard for the Exchange of Product Model Data
SysML	Systems Modeling Language
UML	Unified Modeling Language
UML-PA	Unified Modeling Language for Process Automation
VDI	Verein Deutscher Ingenieure
VDE	Verband der Elektrotechnik, Elektronik und Informationstechnik
VR	Virtual Reality
XML	Extensible Markup Language
ZVEI	Zentralverband Elektrotechnik- und Elektronikindustrie

1 Einleitung und Zielsetzung

1.1 Ausgangssituation

Hersteller hochproduktiver Fertigungsanlagen und Werkzeugmaschinen können nur durch die Bereitstellung innovativer Produkte erfolgreich am Markt bestehen *(Gausemeier u. a. 2004; Milberg 2003, Weck 2005)*. Konkret bedeutet dies, dass, neben der Entwicklung und Integration neuer und der Optimierung bestehender technologischer Prozesse, die Leistungsfähigkeit der Maschinen und Anlagen verbessert werden muss *(BMBF 2002)*. Aus Anwendersicht wird darüber hinaus die Anpassung der Systeme an individuelle Anforderungen sowie die Sicherstellung einer hohen Produktqualität als selbstverständlich betrachtet, wobei resultierende Kostensteigerungen nicht über steigende Preise kompensiert werden können.

Die Forderung nach Produktivität und Qualität einerseits und Individualität andererseits hat wesentlich zum erfolgreichen Einsatz von Automatisierungstechnik in Produktionssystemen beigetragen *(Milberg 1997)*. Waren früher Steuerungsfunktionen noch ganz oder teilweise an die Mechanik gebunden, so wird die Funktionalität heute durch die stetig steigende Rechenleistung kostengünstiger Mikroprozessoren softwaretechnisch realisiert. Dem entsprechend verschiebt sich die Wertschöpfung immer mehr in die Steuerungssoftware *(Schweiker 2003)*. Lag der relative Anteil der Software an den Entwicklungskosten einer Produktionsanlage zu Beginn der siebziger Jahre bei etwa 20 Prozent, so waren es am Ende des Jahrtausends bereits fast 40 Prozent *(Bender u. a. 2000)*.

Der steigende Funktionsumfang und die Variation bestehender Funktionalität aufgrund kundenspezifischer Anpassungen resultieren in einer zunehmenden Komplexität der Steuerungssoftware *(Reinhart u. a. 1998)*. Voraussetzung für deren Beherrschung ist die abteilungsübergreifende Kooperation der an der Entwicklung beteiligten Fachdisziplinen *(Weck 2001)*. Deshalb und im Hinblick auf die Forderung nach einer weiteren Reduktion der Time to Market[1] sowie der Entwicklungskosten, stellt eine Parallelisierung und Synchronisierung der Engineeringtätigkeiten (engl. Concurrent Engineering) *(Lu 2004; Winner u. a. 1988)* eine fundamentale Prämisse für die erfolgsorientierte Realisierung zukünftiger Produktionssysteme dar *(Gausemeier u. a. 2000a)*.

Basis für die praktische Umsetzung interdisziplinärer Entwicklungsprozesse ist der konsequente und durchgängige Einsatz innovativer Simulations- und Berechnungsmethoden *(Großmann 2002, Neugebauer u. a. 2003)*. Die systematische Nutzung moderner Entwicklungswerkzeuge erlaubt die Parallelisierung und Abstimmung von Abläu-

[1] Die Time to Market ist die Zeit, die ein Produkt benötigt, um von der Idee bis zur Markteinführung zu gelangen.

fen und die sichere Vorhersage des Verhaltens von Fertigungsanlagen und deren Baugruppen *(Weck u. a. 2003; Walker 2004)*. Zudem kann durch die Abbildung der Produkteigenschaften in Form digitaler Modelle (virtuelle Prototypen) der kostenintensive Einsatz physikalischer Prototypen reduziert und das Entwicklungsrisiko gesenkt werden *(Bley 2003; Schenk 2003)*.

Moderne Produktionssysteme sind als mechatronische Systeme zu betrachten *(Heimann u. a. 2001; Isermann 2002)*. Elektrische, elektromechanische, hydraulische und pneumatische Komponenten werden zur Erfüllung spezifischer Produktfunktionalität mit leistungsfähigen Steuerungs- und Regelungsbaugruppen integriert. Dieser Aspekt muss beim Entwurf und bei der Optimierung der Fertigungssysteme beachtet werden *(Brecher u. a. 2002; Van Brussel 2005)*. Entsprechende wissenschaftliche Ansätze zielen daher auf die Integration fachspezifischer Entwicklungsmethoden und Rechnerwerkzeuge ab. Als Beispiele genannt seien die Co-Simulation *(Ören 2002)* von Zerspanprozessen und der Systemdynamik von Werkzeugmaschinen *(Großmann & Mühl 2004; Zäh & Oertli 2004)* zum Zwecke des Reglerentwurfs sowie die Koppelung einer realen Numerischen Steuerung an ein Mehrkörper-Simulationsmodell (MKS-Modell) mit dem Schwerpunkt Prozessoptimierung *(Denkena u. a. 2004)*.

Die Entwicklung der Steuerungssoftware als wesentlicher Bestandteil derartiger Systeme ist diesem Paradigma ebenso unterworfen. Folglich darf die Spezifikation, Konzeption, Umsetzung und Verifikation der Softwarefunktionalität nicht isoliert betrachtet werden. Vielmehr ist es notwendig, die Eigenschaften und den strukturellen Aufbau der Fertigungssysteme im Rahmen der Softwareentwicklung durchgängig zu berücksichtigen *(Anderl u. a. 2005; Meier u. a. 2004)*. Während entsprechende Methoden in manchen Bereichen der Entwicklung eingebetteter Systeme (Embedded Systems) bereits praktisch umgesetzt werden *(Gonzalez & Parkin 2003; Krauss 2004)*, sind auf dem Gebiet der Softwareentwicklung für automatisierte Fertigungssysteme nur wenige Ansätze bekannt *(z. B. Albert u. a. 1999; Osmers 1998; Storr u. a. 1999)*.

Trotz der Potentiale des Einsatzes virtueller Prototypen bei der Entwicklung von Fertigungsanlagen halten entsprechende Techniken nur zögerlich Einzug im industriellen Umfeld. Die Ursache hierfür liegt neben der mangelnden Spezialisierung der kommerziellen Software-Pakete bezüglich der besonderen Anforderungen des Produktionssystembaus (*Pritschow & Croon 2002)* auch in der mangelnden Effizienz der Engineeringprozesse begründet *(Hardebusch 2002)*. Insbesondere durch die Defizite bei der Erstellung der Modelle wird der Nutzen virtueller Prototypen relativiert. Die derzeit gelebte, vertikale Integration der Modellbildungsschritte führt zu redundanter Datenhaltung, Brüchen in der Durchgängigkeit der Daten und Mehrfacheingaben. Folgen davon sind Inkonsistenzen und eine Verminderung der Datenqualität *(Baudisch 2003)*.

Einen visionären Ansatz zur angestrebten horizontal[2] und vertikal durchgängigen Modellbildung und damit eine effiziente Entwicklung auf der Grundlage virtueller Prototypen bildet die modellgetriebene Entwicklung (engl. MDE - Model Driven Engineering) *(Alanen u. a. 2004; Fondement & Silaghi 2004; Kent 2002)*. Im Gegensatz zur auf fachspezifischen Dokumenten basierenden Entwicklung werden beim MDE für eine disziplinenübergreifende Betrachtung komplexer Systeme abstrakte Modelle verwendet, um die vielschichtigen funktionalen Zusammenhänge abzubilden (vgl. Abbildung 1.1). Diese werden im Verlauf des Entwicklungsprozesses detailliert und mit Informationen angereichert. Die notwendigen Projektunterlagen werden anschließend durch spezielle Generatoren direkt aus den Modellen erzeugt *(Litto u. a. 2004; Rugaber & Stirewalt 2004)*. Damit dieser Ansatz durchgängig verfolgt werden kann, sind für die einzelnen Entwicklungsphasen entsprechende Modellkonzepte notwendig, die anhand geeigneter Formalismen rechnerinterpretierbar abgebildet werden müssen.

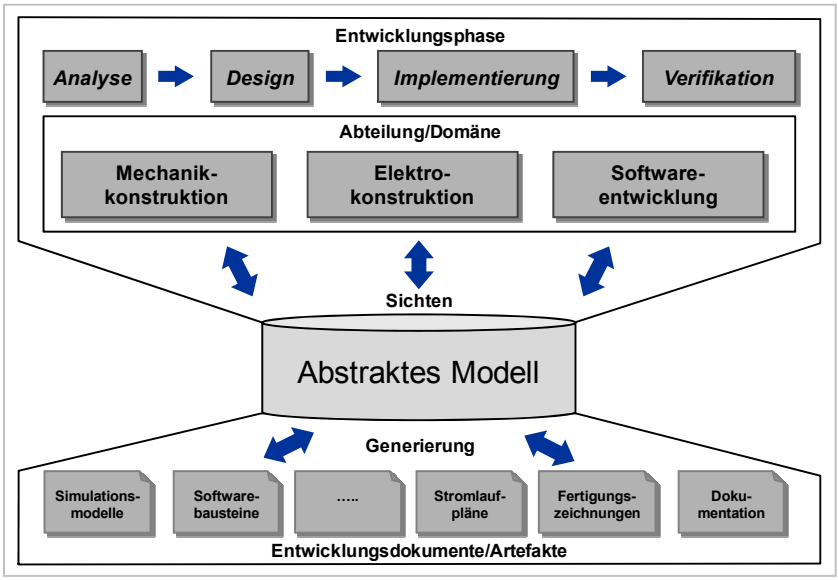


Abbildung 1.1: Model Driven Engineering (Schematische Darstellung)

Gerade im Bereich der Softwareentwicklung ist eine modellbasierte Funktionsentwicklung Grundvoraussetzung, um die Potentiale mechatronischer Systeme optimal zu nutzen und die Komplexität zukünftiger Produkte beherrschbar zu machen *(Otterbach & Schütte 2004; Zentner & Bertram 2005)*. Ein durchgängig modellgetriebener Entwurf sowie eine darauf aufbauende simulationsgestützte Verifikation und Optimierung der

[2] Die horizontale Integration fokussiert sich auf eine auch abteilungsübergreifend konsistente Modellbildung.

implementierten Funktionen ist bei der Entwicklung von Steuerungssoftware für Produktions- und Fertigungssysteme jedoch bislang nicht Stand der Technik. Es fehlen geeignete, interdisziplinäre Beschreibungsmittel, Modellierungsmethoden und Entwicklungswerkzeuge zur Spezifikation der Systemfunktionalität und eine durchgängige Vorgehensweise zur Entwicklung der Steuerungssoftware *(Gewald & Mikk 2003)*.

1.2 Zielsetzung und Fokus

Ziel dieser Arbeit ist es daher, Grundlagen für die modellgetriebene Entwicklung und ingenieurmäßige Erstellung der Steuerungssoftware automatisierter Fertigungssysteme zu schaffen. Wesentliche Aufgaben dabei sind:

- Die Ausarbeitung eines domänenspezifischen Ansatzes zur modellgetriebenen Softwareentwicklung für automatisierte Fertigungssysteme;
- die Spezifikation der notwendigen Beschreibungstechniken und Modellbildungsmethoden; dabei muss insbesondere der interdisziplinäre Charakter der Konstruktion und Inbetriebnahme von Fertigungssystemen berücksichtigt werden;
- die Integration von Simulationsmethoden in den ausgearbeiteten Ansatz mit dem Zweck der Verifikation und Validierung der Steuerungssoftware;
- die Entwicklung einer geeigneten Vorgehensweise zur durchgängigen Anwendung der modellgetriebenen Softwareentwicklung beim Entwurf, der Implementierung und der Inbetriebnahme der notwendigen Steuerungsfunktionalität.

Wesentliche durch die modellgetriebene Entwicklung zu erwartende Vorteile sind:

- Eine verbesserte Integration der Erstellung der Steuerungssoftware in den gesamten Entwicklungsprozess von Fertigungssystemen;
- die Reduktion der Entwicklungszeiten durch eine Parallelisierung und Synchronisierung von Engineeringtätigkeiten;
- eine Steigerung der Qualität durch eine durchgängige und konsistente Modellbildung und simulationsgestützte Absicherung und Optimierung der Steuerungssoftware;
- die langfristige Sicherung von implizit vorhandenem Wissen durch Modellierung und Formalisierung;
- eine Entlastung der Produktentwicklung von zeit- und kostenintensiven Routinetätigkeiten durch die automatisierte Generierung von Entwicklungsdokumenten aus den Modellen;
- die durchgängige Rechnerunterstützung von der Anforderungsanalyse bis hin zur Inbetriebnahme, Wartung und Pflege der Software.

Bei der Entwicklung des Ansatzes sind Randbedingungen bezüglich Standards und Normen zu Vorgehensmodellen, Entwurfsmethoden, Beschreibungsmitteln und Modellierungskonzepten zu beachten. Des Weiteren wird der Fokus auf die Entwicklung der Steuerungssoftware für Fertigungssysteme gelegt. Prinzipiell sollten die Ergebnisse jedoch auch auf andere automatisierte Systeme, wie beispielsweise Montagesyste-

me, übertragbar sein. Detaillierte Untersuchungen hierzu sind jedoch nicht Bestandteil der vorliegenden Arbeit.

1.3 Aufbau und Inhalt der Arbeit

Um die formulierte Zielsetzung zu erreichen, wurde eine Vorgehensweise gewählt, die sich in der in Abbildung 1.2 dargestellten Gliederung der Arbeit widerspiegelt.

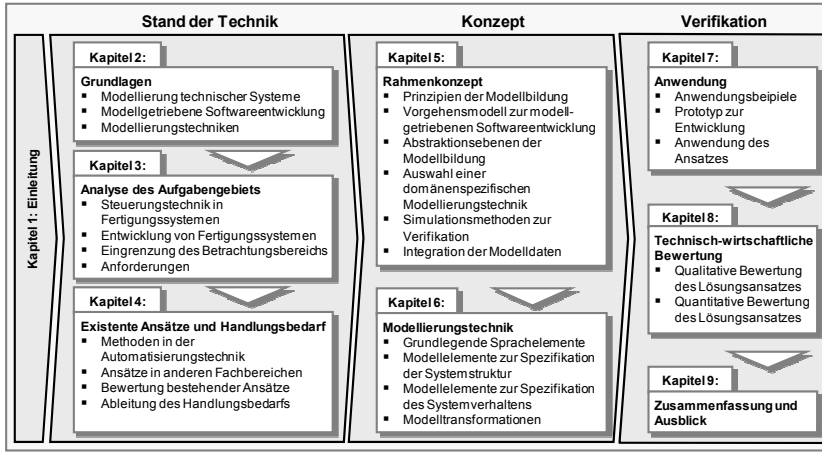


Abbildung 1.2: Vorgehensweise und Struktur der Arbeit

Kapitel 2 bietet eine Beschreibung der zum Verständnis der Arbeit notwendigen theoretischen Grundlagen. In einem ersten Schritt wird eine Einführung in die Basiskonzepte der modellgetriebenen Softwareentwicklung gegeben. Ebenso erfolgt eine Darstellung der fundamentalen Aspekte der Modellierung technischer Systeme sowie deren Klassifikation. Erläuterungen zu prinzipiellen formalen Beschreibungsformen schließen die theoretische Betrachtung ab. Entsprechend den Eigenschaften eines Systems und dem Fokus der Modellbildung werden unterschiedliche Beschreibungsmittel und Notationen eingesetzt und teilweise auch kombiniert. Ausführungen zu ausgewählten Ansätzen im Bereich der Softwareentwicklung für automatisierte Systeme und zur Abbildung physikalischer Gesetzmäßigkeiten bilden die Basis für die Entwicklung eines für die Domäne der Automatisierungstechnik geeigneten Ansatzes.

Kapitel 3 dient einer eingehenden Analyse des gewählten Aufgabengebietes und unterstreicht die Motivation zur modellgetriebenen Entwicklung der Steuerungssoftware. Basierend auf einer Beschreibung des strukturellen Aufbaus automatisierter Fertigungseinrichtungen erfolgt die Darstellung typischer Steuerungsarchitekturen und der verwendeten automatisierungstechnischen Komponenten. Im Anschluss daran wird die Vorgehensweise bei der Entwicklung der Maschinen und Anlagen einer kritischen Betrachtung unterzogen. Im Fokus der Untersuchung liegt die Umsetzung der geforderten

Steuerungsfunktionalität im industriellen Umfeld. Identifizierte Verbesserungspotentiale, eine Erörterung der zukünftigen Trends der Automatisierungstechnik sowie die Präzisierung des im Rahmen der Arbeit betrachteten Aufgabengebietes bilden die Grundlage zur abschließenden Ableitung von detaillierten, domänenspezifischen Anforderungen an einen modellgetriebenen Entwicklungsansatz.

Aufbauend auf den in Kapitel 3 beschriebenen Rahmenbedingungen beinhaltet *Kapitel 4* Ausführungen zum aktuellen Stand der Technik bei der Entwicklung von Steuerungssoftware für automatisierte Fertigungseinrichtungen. Betrachtet werden Ansätze im Umfeld der modellgestützten Softwareerstellung und Verifikation der implementierten Funktionen, aber auch Vorgehensweisen zur Gestaltung der Entwicklungsprozesse. Das Kapitel schließt mit einer Zusammenfassung der aufgeführten Arbeiten.

In *Kapitel 5* wird ein Rahmenkonzept für die modellgetriebene Softwareentwicklung beschrieben. Dieses umfasst sowohl grundlegende Prinzipien zur Modellierung automatisierter Fertigungssysteme als auch ein mehrstufiges Vorgehensmodell zur Entwicklung der Steuerungssoftware im interdisziplinären Team. Ergänzend werden mehrere Abstraktionsebenen für die Modellierung eingeführt und hierzu grundsätzlich geeignete Techniken ausgewählt. Die Vorstellung adäquater Simulationsmethoden zur Verifikation der Entwicklung sowie die Diskussion prinzipieller Möglichkeiten zur Nutzung der Modelldaten für weitere Geschäftsprozesse der Entwicklung von Fertigungssystemen schließen das Kapitel ab.

In *Kapitel 6* wird als wesentlicher Bestandteil der Arbeit eine interdisziplinäre Modellierungstechnik zur Entwicklung der Steuerungsfunktionalität beschrieben und detailliert spezifiziert. Diese umfasst sowohl grundlegende Sprachelemente als auch Konstrukte zur Modellierung der Struktur und des Verhaltens von Fertigungssystemen. Die Weiterverwendung bereits generierter Entwicklungsinformationen in nachfolgenden Prozessschritten wird ebenfalls diskutiert.

Kapitel 7 dokumentiert die praktische Umsetzung und Verwendung der vorgestellten Konzepte anhand ausgewählter Beispiele. Anhand der Nutzung des modellgetriebenen Ansatzes bei der Konzeption, Umsetzung und Inbetriebnahme einer automatisierten Mess- und Justageeinrichtung sowie bei einer Änderungskonstruktion an einem Fräsbearbeitungszentrum wird die Anwendung der vorgestellten Methodik erläutert.

Im Rahmen der in *Kapitel 8* vorgenommenen Diskussion der erzielten Ergebnisse wird der vorgestellte Lösungsansatz aus technischer und ökonomischer Sicht bewertet. Im Vordergrund steht die Abwägung der wesentlichen Vor- und Nachteile des vorgeschlagenen Konzeptes, um daraus eine Aussage über die praktische Umsetzung des Ansatzes abzuleiten.

Kapitel 9 schließt die Arbeit mit einer Zusammenfassung und einem Ausblick darauf, wie die vorgestellten Ergebnisse langfristig in die Entwicklung der Steuerungssoftware für automatisierte Fertigungssysteme einfließen können.

2 Grundlagen

2.1 Grundlagen der modellgetriebenen Softwareentwicklung

2.1.1 Modellgetriebene Entwicklung im Fertigungssystembau

Die in Abschnitt 1.1 beschriebenen Randbedingungen bei der Entwicklung von Fertigungssystemen bedingen prinzipiell neue Verfahren und Vorgehensweisen zu deren Planung, Konzeption, Umsetzung, Inbetriebnahme und Wartung. Aufgrund der zunehmenden äußeren und inneren Komplexität der Anlagen sowie der wachsenden Anforderungen in Bezug auf die Leistungsfähigkeit, die Qualität, die Entwicklungszeit und die Ressourcen ist ein effizientes und effektives Engineering mit konventionellen Vorgehensweisen nicht mehr realisierbar. Nicht ohne Grund haben sich daher auf Computermodellen basierende Methoden bei der Entwicklung technischer Systeme auf ganzer Linie durchgesetzt. So erlauben beispielsweise moderne 3D-CAD-Systeme eine zielgerichtete Umsetzung innovativer Ideen von den Anforderungen über das Konzept bis hin zum fertig entwickelten Produkt. Auf der Basis eines rudimentären Modells und einer wohldefinierten Produktstruktur werden erste Abschätzungen und Auslegungsrechnungen realisiert, bevor eine Überführung des Entwurfs in detaillierte Modelle der Einzelbauteile und Baugruppen erfolgt. Die schrittweise Verfeinerung der Modelle wird durch die Verwendung vordefinierter Bauteile und Baugruppen in Form von Normteilbibliotheken oder die Wiederverwendung von Elementen aus anderen Projekten unterstützt. Ebenso sind durch deren parametrischen Aufbau schnelle Anpassungen möglich. Die zur Fertigung des Produktes notwendigen NC-Programme können durch den Einsatz von CAM-Systemen[1] direkt aus dem dreidimensionalen Modell abgeleitet und in Bezug auf fertigungstechnische Randbedingungen (z. B. Kollisionen) überprüft werden. Darüber hinaus ist die unmittelbare Generierung der Produktdokumentation (z. B. Montagezeichnungen oder Stücklisten) aus dem Modell realisierbar. Auch die Bereitstellung der im Computermodell abgebildeten Informationen für andere Entwicklungsaufgaben wird mittlerweile im praktischen Umfeld erfolgreich umgesetzt. Als Beispiel genannt sei die Nutzung der dreidimensionalen Modelle für Berechnungen mit der FEM[2] oder für Visualisierungs- und Marketingzwecke.

Anhand des beschriebenen Beispiels ist ersichtlich, dass der modellgetriebene[3] Entwicklungsansatz im Rahmen der mechanischen Konstruktion bereits umgesetzt wird. Bei der Softwareentwicklung hingegen hat die Verwendung von Modellen zwar ebenfalls eine gewisse Tradition und findet auch eine zunehmende Verbreitung, doch handelt es sich dabei in der Regel nur um eine Dokumentation der implementierten Sys-

[1] CAM = Computer Aided Manufacturing.

[2] FEM = Finite-Elemente-Methode.

[3] Der Begriff "modellgetrieben" fokussiert die zentrale, treibende Rolle des Modells bei der Entwicklung.

teme. Es gibt lediglich eine gedankliche Verbindung zwischen den Modellen und deren Umsetzung in Form von Software *(Stahl & Voelter 2005)*. Des Weiteren werden meist nur frühe Entwicklungsphasen (Analyse- und Designmodelle) betrachtet, eine vollständige Ausdetaillierung findet nicht statt.

Diese Vorgehensweise bringt zwei fundamentale Nachteile mit sich. Zum einen tragen reine Dokumentationsmodelle nicht unmittelbar zum Projektfortschritt bei, zumal sie erst durch die Softwareentwicklerinnen und Softwareentwickler analysiert, interpretiert und implementiert werden müssen. Zum anderen ist Software als ein dynamisches Produkt zu betrachten, das im Laufe des Entwicklungsprozesses zum Teil starken Änderungen unterworfen ist. Dies resultiert in aufwendigen Anpassungen der Dokumentation. Die Modelle werden daher häufig als unbedeutend betrachtet und nicht aktualisiert, was in der Folge zu Inkonsistenzen und erhöhtem Entwicklungsaufwand führt *(Beckering 2003)*.

2.1.2 Zielsetzung der modellgetriebenen Softwareentwicklung

Die modellgetriebene Softwareentwicklung (MDSD[4]) stellt ein neues Entwicklungsparadigma dar, mit dem Ziel, die genannten Unzulänglichkeiten zu beseitigen. Die Grundidee besteht darin, bereits frühzeitig im Entwicklungsprozess auf der Basis eines Pflichtenheftes ein *ausführbares*[5] *Funktionsmodell* der Software zu erstellen, welches als zentraler Ausgangspunkt für alle nachfolgenden analytischen und konstruktiven Entwicklungsschritte dient *(Schlingloff u. a. 2004)*. Im weiteren Verlauf wird darauf aufbauend ein Implementierungsmodell und letztendlich ausführbarer Code erzeugt. Ebenso bildet die funktionale Modellierung die Grundlage für die Ableitung von Testspezifikationen. Anhand dieser werden im Rahmen der Inbetriebnahme die notwendigen Softwaretests durchgeführt. Dieser neue Entwicklungsansatz schlägt somit die Brücke zwischen den unterschiedlichen Engineeringphasen und erlaubt die frühzeitige Validierung der Softwarefunktionalität (siehe Abbildung 2.1). Modelle sind somit als zum erzeugten Quellcode gleichwertige Entwicklungsartefakte zu betrachten.

Ein besonderes Kennzeichen der MDSD ist, dass die erstellten Modelle der Systemfunktionen zunächst Plattform-unabhängig sind (PIM[6]). Das Systemverhalten wird zunächst auf einer sehr abstrakten Ebene, losgelöst von der späteren physikalischen und softwaretechnischen Umsetzung, beschrieben. Sukzessive erfolgt eine Verfeinerung der Modelle zu Implementierungsmodellen (PSM[7]), die auf die intendierte Ziel-

[4] MDSD = Model Driven Software Development.

[5] Ausführbar bedeutet im beschriebenen Zusammenhang, dass die Modelle durch Rechnerwerkzeuge interpretierbar sind und gegenüber statischen Modellen auch ein dynamisches Verhalten aufweisen.

[6] PIM = Platform Independent Model

[7] PSM = Platform Specific Model.

plattform zugeschnitten sind. Der zusätzliche Aufwand zur Erstellung der einzelnen Modelle wird minimiert, indem notwendige Modelltransformationen und die abschließende, spezifische Generierung der Softwareprogramme soweit möglich und gewünscht, automatisiert werden.

In Analogie zu den Bauteilbibliotheken in CAD-Systemen liegt ein weiterer Schwerpunkt auf der Verwendung standardisierter Softwarebausteine, Frameworks und Vorlagen (Templates). Genauso wie bei der mechanischen Konstruktion wird damit das Ziel verfolgt, die Effizienz und Effektivität der Entwicklung zu steigern. Die Wiederverwendung bereits erstellter Softwarebausteine und von Zulieferern angebotenen, getesteten Bibliotheken erhöht die Softwarequalität und vermeidet redundante Arbeitsschritte. Zusätzlich kann durch die Bereitstellung von Vorlagen, die allgemein notwendige Funktionen sowie eine vorbereitete, klare Basisdokumentation beinhalten und nach einem einheitlichen Konzept strukturiert sind, die Transparenz verbessert und unnötige Routinearbeit vermieden werden. Außerdem tragen diese zur schnellen Einarbeitung neuer Projektmitarbeiterinnen und -mitarbeiter bei und erleichtern die Wartung der Software durch das Servicepersonal.

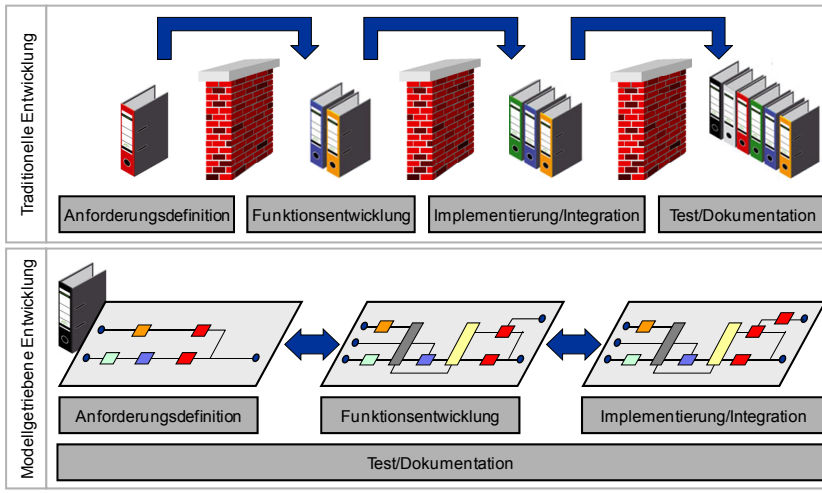


Abbildung 2.1: Traditioneller und modellgetriebener Entwicklungsansatz

Neben den genannten nützlichen Effekten bietet ein auf Modellen basierender Entwicklungsprozess noch eine Vielzahl weiterer Vorteile:

Die Abstraktion von der eigentlichen Zielsetzung (z. B. Erstellung von bestimmten Codefragmenten) hin zu einem leichter verständlichen Modell erleichtert die Diskussion der Entwicklungsaufgabe im interdisziplinären Umfeld und unterstützt die Vermei-

dung kosten- und zeitintensiver Entwicklungsfehler. Dies ist insbesondere bei der Konzeption und Umsetzung mechatronischer Systeme von wesentlichem Interesse, da sowohl der Ausbildungshintergrund wie auch das Begriffsverständnis der beteiligten Entwicklungsdomänen sehr unterschiedlich ist und ein Großteil des Änderungs- und Überarbeitungsaufwandes diesen Gegebenheiten zugrunde liegt *(BMBF 2005)*.

Ein weiteres signifikantes, abstraktionsbasiertes Potential ist die bessere Handhabbarkeit der zunehmenden Komplexität der Software. Modellierungssprachen ermöglichen eine frühzeitige Fokussierung auf die wesentlichen Entwicklungsaufgaben, ohne dass bereits die Feinheiten der technischen Umsetzung betrachtet werden müssen. Durch geeignete Modellelemente zum Aufbau von domänenspezifischen Softwarearchitekturen steigt auch der Grad der Wiederverwendbarkeit. Zudem wird durch die erstellten Modelle Expertenwissen in formaler und damit verständlicher Form in der Breite verfügbar gemacht.

Der Einsatz von Modellierungssprachen trägt zur Steigerung der Softwarequalität bei, da sich die anhand der Modelle definierte Architektur der Software gleichförmig in der Implementierung wiederfindet. Dies hat nicht nur kurzfristig eine Verkürzung der Entwicklungs- und Inbetriebnahmephase zur Folge, sondern auch langfristige Auswirkungen, wie beispielsweise geringere Wartungskosten und die Verbesserung des Images beim Kunden. Auch die Entlastung des Entwicklungspersonals von fehleranfälligen Routinetätigkeiten durch die automatisierte Generierung von Dokumenten aus den Modellen ist als Vorteil zu betrachten *(Bettin 2004)*.

Die MDSD erlaubt aufgrund der ausführbaren Modelle bereits in frühen Entwicklungsphasen eine simulationsgestützte Absicherung der umzusetzenden Funktionen. Durch die Überprüfung auf der Modellebene können konzeptionelle Fehler frühzeitig erkannt und durch entsprechende Änderungen direkt behoben werden. Dieser praktische Nutzen der Modellierung trägt nicht nur zur Einsparung von Ressourcen bei, sondern resultiert auch in einer signifikanten Erhöhung der Akzeptanz bei den Entwicklerinnen und Entwicklern. Die Erstellung der Modelle wird nicht mehr nur als unnötiger Mehraufwand verstanden, der reinen Dokumentationszwecken dient, sondern trägt unmittelbar zur Realisierung der Entwicklungsaufgabe bei.

Zusammenfassend kann festgestellt werden, dass die genannten Vorteile aus der Erhöhung des Abstraktionsniveaus durch den Einsatz formalisierter und somit rechnerinterpretierbarer Modellierungssprachen resultieren. Dies stellt auch einen wesentlichen Unterschied zu traditionellen Ansätzen der Softwareentwicklung dar, bei denen auf das Zielsystem abgestimmte Programmiersprachen eingesetzt werden. Zur erfolgreichen Umsetzung einer derartigen Methodik sind jedoch eine Reihe von Randbedingungen zu beachten.

2.1.3 Anforderungen an einen modellgetriebenen Ansatz

Die Abstraktion der erforderlichen Softwarefunktionalität von der proprietären Implementierung hin zu generischen Modellen ist nicht für die gesamte Domäne der Softwareentwicklung durch einen einheitlichen Ansatz abbildbar. Dies ist dadurch bedingt, dass bestimmte Subdomänen typische Hardwarearchitekturen, Betriebssysteme und teilweise auch determinierte Zielsprachen voraussetzen. Darüber hinaus werden an die Eigenschaften der implementierten Funktionen sehr unterschiedliche Anforderungen gestellt. Während beispielsweise bei der Erstellung von Anwendungen für das World Wide Web Interoperabilität auf unterschiedlichen Systemen eine wesentliche Rolle spielt, sind dort Restriktionen bezüglich der Ausführungszeit einzelner Funktionen und der Schlankheit der Applikationen in der Regel von untergeordnetem Interesse. Bei der Entwicklung von Anwendungen für Steuergeräte, wie sie üblicherweise in mechatronischen Systemen eingesetzt werden, sind hingegen gerade diese Eigenschaften für das sichere Erfüllen der Aufgabenstellung als Prämisse zu betrachten. Folglich bedarf die Spezifikation von Software durch formale Modelle einer domänenspezifischen Modellierungssprache (DSL[8]), welche die Anforderungen des jeweiligen Anwendungsgebietes mit hinreichender Genauigkeit erfüllt und die notwendigen Beschreibungskonstrukte (Modellelemente) bereitstellt *(Fondement & Silaghi 2004)*. So ist beispielsweise der Einsatz von Timern[9] bei der Entwicklung der Software für Speicherprogrammierbare Steuerungen (SPS) weit verbreitet. Eine für diese Domäne geeignete Sprache muss folglich deren anschauliche und vollständige Modellierung unterstützen.

Trotz der Abstraktion vom Detaillierungsgrad der Programmiersprachen muss eine DSL also die Möglichkeit bieten, Modelle kompakt und auf das Wesentliche reduziert darzustellen, ohne dabei syntaktische und semantische Lücken zu tolerieren. Gleichzeitig ist in Bezug auf die Zielsetzung der MDSD die Lesbarkeit durch den Menschen sowie die Interpretation durch den Rechner sicherzustellen. Diese Anforderungen lassen sich nur erfüllen, wenn sich die in der eingesetzten Beschreibungstechnik verwendeten Notationen an der Begriffswelt der jeweiligen Domäne orientieren und somit eine problemorientierte Lösung der Aufgabenstellung gestatten.

Die sukzessive Verfeinerung und Detaillierung der im Verlauf der Entwicklung aufgebauten Modelle stellt ebenfalls Bedingungen an die eingesetzten Modellkonstrukte. Während in frühen Engineeringphasen eine informale bzw. semiformale Modellierung hinreichend ist, setzt der letzte Schritt der modellgetriebenen Entwicklung, die Plattform-spezifische Generierung des Zielcodes, die Vollständigkeit und Präzision der

[8] DSL = Domain Specific Language (Synonym wird auch der Begriff Modellierungssprache verwendet).

[9] Timer sind Steuerbausteine zur Realisierung unterschiedlicher, zeitbezogener Funktionen.

Modelle voraus. Genauso ist für die simulationsgestützte Absicherung der Softwarefunktionalität eine eindeutige Interpretierbarkeit unabdingbar *(Frankel 2003, S. 32)*.

Die Entwicklung bzw. der Einsatz einer domänenspezifischen Modellierungssprache ist nur dann effizient, wenn die im Verlauf des Entwicklungsprozesses mit Informationen angereicherten Modelle durchgängig weiterverwendet werden können. Wie dies zu geschehen hat, inwieweit dieser Vorgang automatisierbar ist und welche Abstraktionsstufen benötigt werden, ist nicht allgemein definierbar. Hierbei sind neben den fachlichen Rahmenbedingungen der jeweiligen Domäne auch organisatorische Aspekte zu beachten. Ein modellgetriebener Entwicklungsansatz muss daher auch eine Vorgehensweise umfassen, die festlegt, welche Modellkonzepte wann und zu welchem Zweck eingesetzt werden und wie Modellverfeinerungen durchzuführen sind *(Stahl & Voelter 2005, S. 13; Kent 2002)*.

Zusammenfassend sind für die Entwicklung eines domänenspezifischen Ansatzes folgende Aufgaben zu erfüllen:

- Es muss definiert werden, welche Abstraktionsebenen im Rahmen der Entwicklung erforderlich sind. Dabei ist auch zu berücksichtigen, welche Plattformen (Hardware, Betriebssysteme, usw.) integriert werden müssen und welche Vorgehensweisen und Methoden für die jeweilige Domäne geeignet sind.
- Des Weiteren ist festzulegen, welche Modellnotationen notwendig sind, welche Syntax auf der jeweiligen Detaillierungsebene eingesetzt werden soll und welche Eigenschaften des entwickelten Systems bei der Modellbildung zu beachten sind.
- Die Umsetzung von Verfeinerungen ist ebenfalls zu untersuchen. Dabei gilt es insbesondere festzulegen, welche Informationen auf den niedrigeren Abstraktionsebenen integriert werden müssen und wie dies zu realisieren ist.
- Neben den Möglichkeiten zur Übertragung der Modellinformationen zwischen den verschiedenen Detaillierungsebenen gilt es auch, die abschließende Umsetzung der Modelle in Plattform-spezifischen Zielcode zu betrachten.
- Die Validierung der abgebildeten Funktionen auf den unterschiedlichen Abstraktionsniveaus ist genauso Teil eines modellgetriebenen Entwicklungsansatzes. Daher ist die Untersuchung entsprechender simulationsgestützter Ansätze von primärem Interesse.

Zur Beantwortung der genannten Fragestellungen wurden mehrere prinzipielle und teilweise aufeinander aufbauende Lösungen entwickelt, die durch verschiedene Schwerpunktsetzungen gekennzeichnet sind. Gemeinsam ist allen Vorgehensweisen, dass sie Methoden aufzeigen, die soweit generalisierbar sind, dass sie durch Kombination und entsprechende Anpassungen die Erarbeitung eines für eine bestimmte Domäne geeigneten Entwicklungsansatzes, im Sinne der Zielsetzung der Arbeit, erlauben.

2.1.4 Model Driven Architecture

Model Driven Architecture (MDA) stellt den allgemeingültigsten Ansatz zur modellgetriebenen Softwareentwicklung dar. Der von der Object Management Group (OMG[10]) erarbeitete Vorschlag beschreibt eine durchgängige Methode zur Anwendung von Modellen im Rahmen der Softwareentwicklung *(OMG 2003)*. Besonderer Wert wird dabei auf die klare Trennung zwischen Plattform-unabhängigen und zielsystemspezifischen Modellen gelegt. Dadurch soll es möglich sein, Softwaresysteme zu entwerfen, die autark von deren technischen Realisierung sind. Durch die Auflösung der Abhängigkeiten zwischen den Systemfunktionen und deren Umsetzung wird die Lauffähigkeit der Software auf unterschiedlichen Plattformen sichergestellt. Als weitere Schwerpunkte des MDA-Konzeptes werden von *(OMG 2003)* Interoperabilität und Wiederverwendbarkeit genannt.

MDA hat die Intention, präzise zu definieren, welche Sprachen bzw. Beschreibungsmittel zur Modellierung verwendet werden sollen, wie die Regeln für Modelltransformationen festzulegen sind und wie Transformationen realisiert und Code generiert werden soll *(Fondement & Silaghi 2004)*. Während für die Modellbildung Sprachen, die auf der Unified Modeling Language (UML) basieren, vorausgesetzt werden, schlägt der Ansatz zur Festlegung der Transformationen und zu deren Umsetzung mehrere Möglichkeiten vor (siehe Abbildung 2.2). Beim Konzept der Markierung erfolgt eine Auszeichnung der Plattform-unabhängigen Modellkonstrukte durch zielsystemspezifische Marken. Die Marken selbst sind zwar nicht Teil dieser Modellbausteine, sie dienen aber entsprechenden Transformatoren als Eingangsinformation. Dadurch können diese entscheiden, welche Transformationsregeln (Mapping) auf die jeweiligen Bausteine angewendet werden müssen. So kann beispielsweise die Markierung einer bestimmten Softwarefunktion die Einbindung eines Plattform-spezifischen Softwarebausteins signalisieren, der diese Funktion unter Berücksichtigung der Eigenschaften des Zielsystems realisiert.

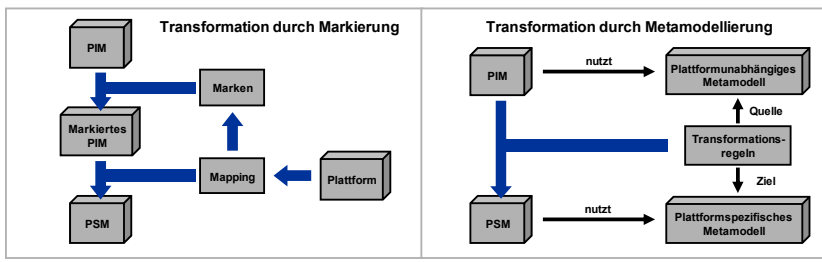


Abbildung 2.2: Beispiele für Transformationen im Rahmen des MDA (OMG 2003)

[10] Die OMG ist ein internationales Konsortium, das Standards für die Programmierung entwickelt.

Ein weiteres Beispiel zur Spezifikation von Modelltransformationen ist die Nutzung von Metamodellen[11]. Diese beschreiben sowohl die Regeln zum Aufbau der unabhängigen wie auch der Plattform-spezifischen Modelle anhand einer formalen Spezifikation. Für entsprechende Transformatoren muss lediglich festgelegt werden, wie die einzelnen Konstrukte der jeweiligen Metamodelle zusammenhängen. Auf der Basis dieser Informationen kann unter Nutzung des Ausgangsmodells die Umwandlung in die zielsystemspezifischen Artefakte durchgeführt werden.

Trotz der relativ präzisen Definition weist der MDA-Ansatz einige fundamentale Schwächen auf. Zum einen existiert zurzeit noch keine einheitliche Sprache, um die notwendigen Modelltransformationen zu beschreiben. Zum anderen liegt der Schwerpunkt auf der Standardisierung von Modellen für bestimmte, weit verbreitete Anwendungsbereiche (z. B. E-Business-Systeme). Auch eine praktische Methodenunterstützung ist nicht Bestandteil der MDA. MDA kann also als Standardisierungsinitiative zur modellgetriebenen Softwareentwicklung betrachtet werden *(Stahl & Voelter 2005, S. 5)*. Um trotz der unvollständigen Reife den MDA-Ansatz zielführend anwenden zu können, ist es notwendig, diesen an den jeweiligen Anwendungsbereich anzupassen.

2.1.5 Architekturzentrierter Entwicklungsansatz

Bei der architekturzentrierten MDSD (AC-MDSD) liegt der Fokus im Gegensatz zur MDA nicht auf der Interoperabilität und Probabilität von Software, sondern auf einer Steigerung der Entwicklungseffizienz sowie der Verbesserung der Qualität und Wiederverwendbarkeit. Dies bedeutet, insbesondere die Entwicklerinnen und Entwickler von fehleranfälliger Routinearbeit zu befreien *(Stahl & Voelter 2005, S. 24)*. Gleichzeitig steht die praktische Handhabbarkeit im Mittelpunkt dieses Ansatzes.

Schwerpunktmäßig eignet sich AC-MDSD zur Erstellung von Software-Systemfamilien, also architektonisch gleichartig gelagerten Produkten. Basierend auf einer Referenzimplementierung wird unter Nutzung einer auf die entsprechende Domäne zugeschnittenen Modellierungssprache eine generative Architektur der Software aufgebaut (siehe Abbildung 2.3). Diese beinhaltet alle notwendigen und allgemeingültigen Basiskonstrukte und definiert gleichzeitig, wie diese unter Nutzung zusätzlicher, Plattform-spezifischer Bausteine durch entsprechende Transformationen in Quellcode überführt werden können. Die Entwicklung dezidierter Applikationen geschieht unter Nutzung dieser generativen Architekturkomponenten. In einem ersten Schritt wird ein Designmodell der Software erstellt. Ein Generator kann dieses nun automatisiert in eine Basisapplikation transferieren. In einem zweiten Schritt erfolgt direkt auf der Quellcode-Ebene die Integration der anwendungsspezifischen Funktionalität.

[11] Ein Metamodell legt die Regeln (Syntax und Semantik) fest, die bei der Modellerstellung anzuwenden sind.

Architekturzentrierte MDSD verzichtet im Gegensatz zur MDA-Vision auf die Transformation von Plattform-unabhängigen in zielsystemspezifische Modelle. Da die Basis für den generierten Code anhand einer verifizierten Referenzimplementierung erstellt wird, kann eine gute Lesbarkeit und Qualität des Generats sichergestellt werden. Der wesentliche Nachteil besteht darin, dass die individuelle Anwendungslogik manuell integriert werden muss. Daher ist hiermit in der Regel nur 60 bis 80% eines Softwaresystems generierbar *(Stahl & Voelter 2005, S. 32)*.

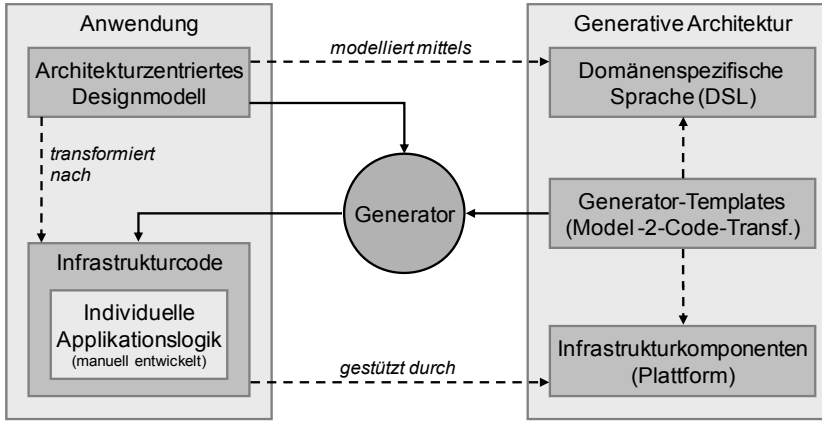


Abbildung 2.3: Architekturzentrierter Entwicklungsansatz (Stahl & Voelter 2005)

2.2 Grundlagen der Modellierung technischer Systeme

2.2.1 Allgemeines

Ein modellgetriebener Entwicklungsansatz erfordert die Bereitstellung geeigneter Techniken zur Beschreibung des zu entwickelnden Systems. Für das Erreichen einer entsprechend dem Einsatzzweck (Modellzweck) hinreichend hohen Modellgüte muss sich der Charakter des abzubildenden Systems in geeigneter Form im Modell widerspiegeln. Zur Auswahl der für die Modellbildung herangezogen Beschreibungsmittel sind daher Kenntnisse bezüglich des Aufbaus und der globalen Eigenschaften von Systemen von wesentlicher Bedeutung. Die Darstellung der notwendigen Grundlagen technischer Systeme sowie deren Modellierung ist somit eine entscheidende Prämisse zum Verständnis der vorliegenden Arbeit.

Zur Beschreibung der vielschichtigen Zusammenhänge in technischen Systemen wurden im Laufe der letzten Jahre und Jahrzehnte eine Reihe domänenspezifischer und domänenübergreifender Ansätze entwickelt. Diese lassen sich jedoch auf allgemein anwendbare Methoden zur Beschreibung von Systemen in ingenieurmäßiger Form

zurückführen, die unter dem Begriff der Systemtheorie[12] *(Girod u. a. 2005; Frey & Bossert 2004)* bzw. Systemtechnik *(Ropohl 1999; Unbehauen 2002)* zusammengefasst werden *(Bruns 1991)*.

2.2.2 Grundlegende Konzepte der Systemtheorie

Im Rahmen der Systemtheorie wird ein System als eine Einheit betrachtet, die über eine bestimmte *Struktur* (einen inneren Aufbau) sowie ein bestimmtes *Verhalten* (Auswirkungen auf seine Umgebung) verfügt *(Zeigler u. a. 2005)*. Gegenüber ihrer Umwelt wird diese Einheit durch die *Systemgrenze* (eine definierte Schnittstelle) abgegrenzt. Die Verbindung zwischen dem System selbst und seinem Umfeld erfolgt durch die Festlegung von *Eingangsgrößen* und *Ausgangsgrößen* (Verhaltensgrößen), die die Systemgrenze überschreiten *(Pahl u. a. 2005, S. 38)* und in Bezug zu einer unabhängigen Variablen definiert werden.

Wesentliches Merkmal der Struktur eines Systems (siehe Abbildung 2.4) ist seine Zerlegbarkeit in kleinere Einheiten durch *Dekomposition*. Diese können wiederum als Systeme (*Teilsystem/Subsystem*) betrachtet werden. Ebenso kann ein System durch *Komposition* (Aggregation) zu einem größeren Gesamtsystem verbunden werden. Dadurch ist es möglich, ein System zur Erfüllung bestimmter Funktionen aus *modularen* Teilsystemen zusammenzusetzen und *hierarchische Strukturen* zu bilden.

Die Komposition erfordert es, dass die Teile eines Systems zueinander in eine konkrete Relation gebracht werden. Dies geschieht durch das definierte Verknüpfen der Eingangs- und Ausgangsgrößen der einzelnen Subsysteme. Der dadurch entstehende strukturelle Zusammenhang wird als *Wirkstruktur* oder Wirkzusammenhang bezeichnet.

Das Verhalten eines Systems bzw. Subsystems wird durch die Relation zwischen den Eingangs- und Ausgangsgrößen sowie seinen *Zustand* charakterisiert. Das Systemverhalten kann deshalb als mathematischer Operator *SV* aufgefasst werden, der die Eingangsgrößen in die Ausgangsgrößen transformiert. Bei einem System mit *i* Eingangsgrößen *x* und *j* Ausgangsgrößen *y* sowie einem aktuellen Zustand *Z* kann dieses durch die Gleichung

$$y_j = SV(x_i, Z) \quad Z = \{z_1, \dots, z_n\} \; n \in \mathbb{N}$$

beschrieben werden. *SV* wird auch als *Systemfunktion*, Systemoperator oder Eingangs-Ausgangs-Relation bezeichnet *(Lunze 2003)*.

[12] Der Begründer der Systemtheorie ist der Biologe Ludwig von Bertalanffy (1901-1972), dessen Ziel es war, gemeinsame Gesetzmäßigkeiten aus verschiedenen Wissensgebieten herauszuarbeiten, indem er deren allgemeine Prinzipien beobachtete. Die Systemtheorie ist also als eine Metatheorie zu betrachten, die eine Integration von unterschiedlichem Wissen erlaubt und in verschiedenen Bereichen anwendbar ist.

Der Zustand *Z* eines Systems wird durch die Menge aller voneinander unabhängigen *Zustandsgrößen* z_k definiert, die dieses zu jeder Zeit vollständig beschreiben. Folglich können diese als die Menge aller konstanten und variablen Attribute eines Systems aufgefasst werden, einschließlich der daraus ableitbaren Ausgangsgrößen *(Bossel 1994, S. 19)*. Der Übergang zwischen den Zuständen eines Systems (*Zustandsübergang*) wird durch die kontinuierliche oder diskrete Änderung mindestens einer Zustandsgröße aufgrund einer oder mehrerer Änderungen der Eingangsgrößen oder weiterer Zustandsgrößen des Systems realisiert.

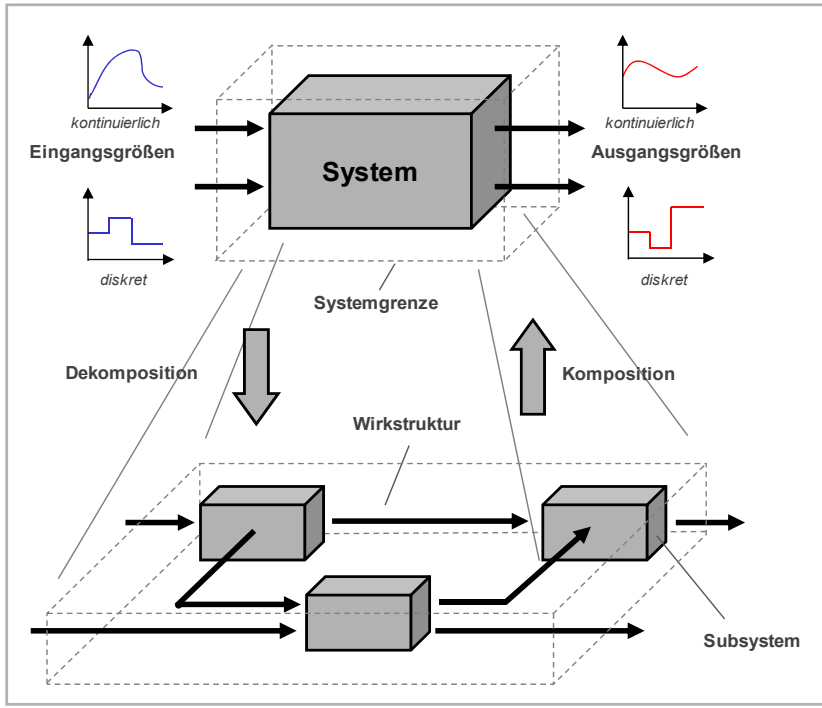


Abbildung 2.4: Struktureller Aufbau eines Systems

Der Wirkzusammenhang bestimmt durch die Verschaltung der Ein- und Ausgangsgrößen maßgeblich das Verhalten des Systems. Dadurch wird die Funktionalität eines Gesamtsystems gegenüber der Summe der Funktionen seiner Subsysteme erweitert. Die Beziehungen zwischen den Einzelteilen können aber auch zu nicht gewollten Effekten (*Störwirkungen*) führen. Dem entsprechend werden Eingangsgrößen in gewollte (*Stellgrößen*) und nicht gewollte (*Störgrößen*) Eingänge unterteilt. Durch Rückwirkung der Systemausgänge auf die -eingänge (*Rückkoppelung*) kann ein System auch

Auswirkungen auf sich selbst haben. Bei technischen Systemen sind beispielsweise Regelungen als Form gewollter Rückkoppelungen zu nennen.

2.2.3 Technische Systeme

Technische Systeme haben, abstrakt betrachtet, die *Funktion*, Energie, Stoff und Daten in technischen Gebilden zu speichern, zu teilen, zu sammeln, zu leiten, zu skalieren, zu wandeln oder zu verknüpfen *(Koller 1998, S. 27)*. Folglich korrespondieren die Ein- und Ausgangsgrößen mit dem Stoff-, Energie- und Datenfluss in und aus dem System. In der Praxis haben sich dem entsprechend die Begriffe *Apparat*, *Gerät* und *Maschine* gemäß der dominierenden Umsatzart als Klassifikationsmerkmal herausgebildet. Eine weitere Differenzierung wird nach der primär zugrunde liegenden physikalischen Realisierung vorgenommen. Bei den im Maschinenbau vorwiegend angewandten technischen Gebilden werden mechanische, hydraulische, pneumatische, elektrische, magnetische, optische, akustische und thermische Systeme unterschieden (siehe Abbildung 2.5).

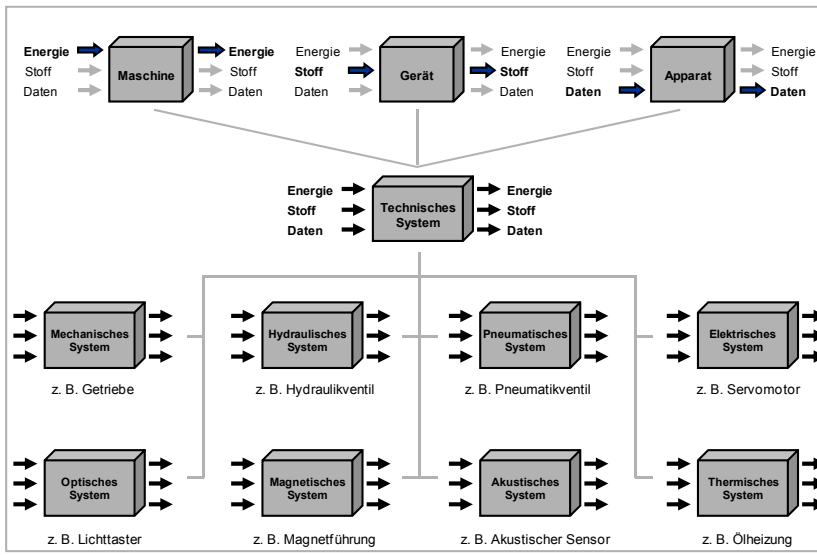


Abbildung 2.5: Einteilung technischer Systeme

Typische technische Systeme, wie Werkzeugmaschinen und Fertigungsanlagen, bestehen in der Regel aus einer Vielzahl von Subsystemen, die jeweils eine Hauptumsatzart haben[13]. So hat die Hauptspindel einer Werkzeugmaschine die primäre Funktion,

[13] Integrierte Systeme mit unterschiedlicher Hauptumsatzart werden auch als „hybride technische Systeme“ bezeichnet. Sie stellen die Mehrzahl der entwickelten technischen Produkte dar.

elektrische Energie in mechanische Energie umzuwandeln. Die Numerische Steuerung hingegen dient im Wesentlichen zur Verarbeitung von Daten, indem Benutzereingaben oder Steuerungsprogramme interpretiert und entsprechende Stellgrößen abgeleitet werden. Als weitere Differenzierungsmerkmale benennt die Literatur unter anderem die Komplexität, die Bau- und Flussstruktur, die Funktion sowie produktspezifische Gestaltungsparameter *(Ehrlenspiel 2003)*.

Die genannten Unterscheidungsmerkmale sind wesentlich für die Strukturierung eines Systems. Diese geschieht entsprechend dem Zweck der Betrachtung nach Funktionen, Montagegesichtspunkten oder Fertigungsmodulen usw. *(Pahl u. a. 2005, S. 39)*. So ist es beispielsweise zweckmäßig, ein zu entwickelndes Produkt im Rahmen der Lösungssuche entsprechend den umzusetzenden Funktionen (*Funktionsstruktur*) zu gliedern. In späteren Entwicklungsschritten, wenn der geometrische und bauliche Zusammenhang des Systems im Fokus liegt, ist eine Gliederung analog zur *Baustruktur* (Bauteile, Baugruppen, usw.) sinnvoll. Häufig ist auch eine kombinierte Strukturierung angebracht, bei der einzelne Teilsysteme funktional gegliedert werden, deren Integration aber entsprechend der Baustruktur stattfindet. Der Aufbau des Wirkzusammenhangs wird meist in enger Anlehnung an den Daten-, Energie- und Stofffluss im System (*Flussstruktur*) realisiert.

Inwieweit die Dekomposition eines Systems notwendig ist, hängt ebenfalls vom Fokus der Betrachtung ab. Prinzipiell kann die Aufgliederung soweit fortgesetzt werden, bis nur noch elementare Strukturen existieren, die als *Konstruktionselemente* bezeichnet und zur Durchführung bestimmter *Grundoperationen* verwendet werden. Gerade für die Aufgabe der Modellbildung und Simulation sind diese von enormer Bedeutung, da sie im Allgemeinen durch physikalische oder logische Gesetzmäßigkeiten beschreibbar und folglich durch mathematische Modelle abbildbar sind.

Bei der Betrachtung technischer Systeme ist es im Sinne der Systemtheorie zweckmäßig, von den Ein- und Ausgangsgrößen zu abstrahieren. Die Energie-, Stoff- und Datenflüsse werden daher einheitlich als *Signale* betrachtet, welche die entsprechenden Zusammenhänge in generalisierter, allgemeingültiger Form beschreiben. Deren Darstellung als direkte oder indirekte Funktion der Zeit ist aufgrund des vorwiegend dynamischen Charakters technischer Systeme adäquat und üblich *(Lunze 2003, S. 39-40)*.

Entsprechend den Eigenschaften der Größe, die durch ein Signal abgebildet wird, können diese klassifiziert werden. Ein *deterministisches* Signal *s* ist dadurch gekennzeichnet, dass sein zeitlicher Verlauf bekannt und somit mathematisch beschreibbar ist:

$$s(t) = f(t)$$

Stochastische Signale hingegen können nicht in ihrem funktionalen, zeitabhängigen Verlauf angegeben werden, sondern sind nur durch Erfahrungswerte und wahrscheinlichkeitstheoretische Charakteristiken (Mittelwert, Varianz, usw.) beschreibbar *(Unbehauen 2002)*. Derartige Signale spielen bei der Betrachtung dynamischer Systeme in

der Regel nur eine untergeordnete Rolle. Im Rahmen der Modellbildung und Simulation werden sie üblicherweise nicht (z. B. Vernachlässigung von Störgrößen) oder nur in vereinfachter Form berücksichtigt.

Ein weiteres Unterscheidungsmerkmal resultiert aus dem möglichen Wertebereich eines Signals (siehe Abbildung 2.6). Während *wertkontinuierliche* Signale innerhalb ihres Definitionsbereichs jeden beliebigen Wert annehmen können, sind *wertdiskrete* Signale auf eine endliche Menge von Werten beschränkt. In Bezug auf die genannten Signalarten sind auch die Begriffe Analogsignal und Digitalsignal gebräuchlich. Die in technischen Systemen häufig vorkommende Wandlung analoger in digitale Größen wird als Quantisierung bezeichnet.

Bezüglich ihrer zeitlichen Abhängigkeit werden *zeitkontinuierliche* und *zeitdiskrete* Signale unterschieden. Erstere sind dadurch gekennzeichnet, dass jedem reellwertigen Zeitpunkt t ein definierter Wert $s(t)$ zuordenbar ist. Viele zeitkontinuierliche Signale können daher durch analytische Zusammenhänge beschrieben werden. Liegt der Wert eines Signals nur zu bestimmten, diskreten Zeitpunkten n vor, so gibt es häufig keine geschlossene analytische Beschreibung. Daher wird der Signalverlauf meist durch eine Folge von Werten $s(n)$ dargestellt. Die einzelnen Zeitpunkte n müssen dabei nicht notwendigerweise äquidistant sein. Sich asynchron ändernde Signale werden auch als ereignisdiskret bezeichnet.

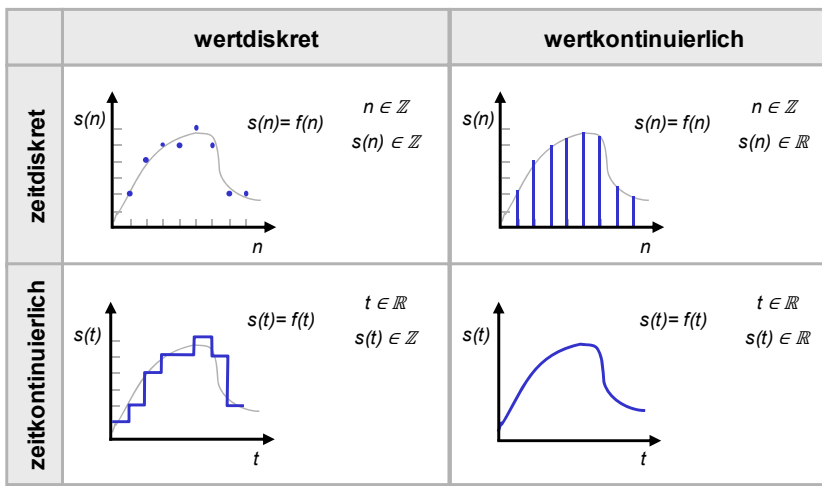


Abbildung 2.6: Signalarten

Als weitere Klassifizierungsmerkmale für Signale sind in der Fachliteratur u. a. der annehmbare Wertebereich oder die Dimensionalität angegeben. Da diese im Rahmen

der Arbeit von untergeordneter Bedeutung sind, wird an dieser Stelle auf entsprechende Quellen *(Schrüfer 2004; Girod u. a. 2005)* verwiesen.

2.2.4 Klassifizierung technischer Systeme

Die Einordnung technischer Systeme lehnt sich weitestgehend an die Gliederung der Ein- und Ausgangsgrößen derselben an. Nach *(Girod u. a. 2005, S. 12*) sind unter digitalen, analogen, zeitkontinuierlichen oder zeitdiskreten Systemen solche zu verstehen, die Beziehungen zwischen den entsprechenden Signalen herstellen. Die Klassifizierung eines dezidierten technischen Gebildes hängt jedoch nicht nur von dessen Eigenschaften, sondern auch wesentlich vom Zweck der Betrachtung ab *(Lunze 2003, S. 43)*. Daher wird in Bezug auf den Hintergrund der vorliegenden Ausführungen eine Unterteilung in *ereignisdiskrete, kontinuierliche und hybride*[14] Systeme *(Litz 2005, S. 385)* als geeignet befunden.

Kontinuierliche Systeme sind dadurch gekennzeichnet, dass sich das Systemverhalten *SV* in seinem zeitlichen Verlauf stetig ändert. Entsprechend dessen Definition als Relation der Eingangs-, Ausgangs- und Zustandsgrößen[15] verfügt ein kontinuierliches System über eine unendliche Menge von Systemzuständen Z^k.

$$y_j^k(t) = SV\left(x_i^k(t), Z^k(t)\right) \quad i,j \in \mathbb{N}$$

Dabei ist es unerheblich, ob die Anzahl der möglichen Zustände aufgrund der Reellwertigkeit der Eingangssignale bzw. Zustandsgrößen $x_i^k(t),\ z_k^k(t) \in \mathbb{R}$ oder aus der Anzahl der betrachteten Zeitpunkte $t \in \mathbb{R}$ resultiert. Folglich können kontinuierliche Systeme sowohl zeitkontinuierliche wie auch wertkontinuierliche Signale verarbeiten. Wesentlich ist, dass der Informationsgehalt im Wert liegt, den ein Signal zu einem definierten Zeitpunkt aufweist.

Viele technische Systeme, wie beispielsweise Regler, arbeiten zeitdiskret, d. h. sie ändern Zustände in diskreten Zeitschritten. Zudem ist der Wertebereich der verarbeiteten Signale und Zustandsgrößen in der Regel auf eine endliche Menge (z. B. 32-Bit) beschränkt. Diese notwendige Diskretisierung bzw. Quantisierung kann jedoch in sehr kleine Schritte aufgeteilt werden, sodass die Eigenschaften analoger bzw. zeitkontinuierlicher Signale nahezu erreicht werden. Daher werden technische Systeme mit einer äquidistanten Abtastung und quasikontinuierlichem Signalwert häufig als kontinuierliche Systeme betrachtet *(Bossel 2004, S. 22)*.

Ereignisdiskrete Systeme können als eine spezielle Variante zeit- und wertdiskreter Systeme aufgefasst werden. Der wesentliche Unterschied zu kontinuierlichen Systemen besteht zum einen darin, dass der Informationsgehalt der verarbeiteten Signale

[14] Hybride Systeme werden in der Fachliteratur häufig auch als diskret-kontinuierliche Systeme bezeichnet.

[15] Zur Unterscheidung zwischen kontinuierlichen und diskreten Größen werden im Weiteren die hochgestellten Indizes k = kontinuierlich und d = (ereignis-)diskret verwendet.

und Zustandsgrößen nicht durch deren aktuellen Wert beschrieben wird. Vielmehr liegt dieser in der Änderung der Werte durch das Unterschreiten bzw. Überschreiten definierter Grenzwerte (Ereignisse). In Bezug auf technische Systeme erfolgt also eine Abstraktion von den beschreibenden physikalischen Zusammenhängen und Gesetzmäßigkeiten hin zur Definition einer endlichen Menge von Zuständen, die durch das System eingenommen werden können und dieses in Bezug auf den Zweck der Betrachtung hinreichend genau abbilden. Zum anderen treten Ereignisse zu vorab nicht bekannten Zeitpunkten auf. Deren absolute Lage auf der physikalischen Zeitachse ist meist nicht wichtig, lediglich die Reihenfolge des Auftretens ist von Bedeutung.

Die Behandlung eines technischen Produktes als kontinuierliches oder ereignisdiskretes System hängt von den Systemeigenschaften und dem Betrachtungszweck ab. Bei der Auslegung einer hydraulisch gesteuerten Positionierachse mit Endlagenüberwachung (siehe Abbildung 2.7 a), wie sie in Werkzeugmaschinen häufig eingesetzt wird, ist es sinnvoll, diese als kontinuierliches System zu betrachten. Die physikalischen Zusammenhänge erlauben es, den notwendigen Systemdruck in Abhängigkeit von den benötigten Kräften und der Kolbenfläche zu bestimmen. Gleichzeitig kann die Geschwindigkeit des Zylinders, und damit die Verfahrzeit, aus dessen geometrischen Abmessungen und dem zu- bzw. abgeführten Ölvolumenstrom bestimmt werden (siehe Abbildung 2.7 b). Bei einer Analyse des Positioniersystems aus der Sicht der Softwareentwicklung sind lediglich die diskreten Zustände, die die Achse einnehmen kann (z. B. „Positionierachse eingefahren“, „Positionierachse ausgefahren“) sowie die Zustandsübergänge relevant. Für den genannten Zweck ist die abstraktere Betrachtung als ereignisdiskretes System sinnvoll und hinreichend (siehe Abbildung 2.7 c).

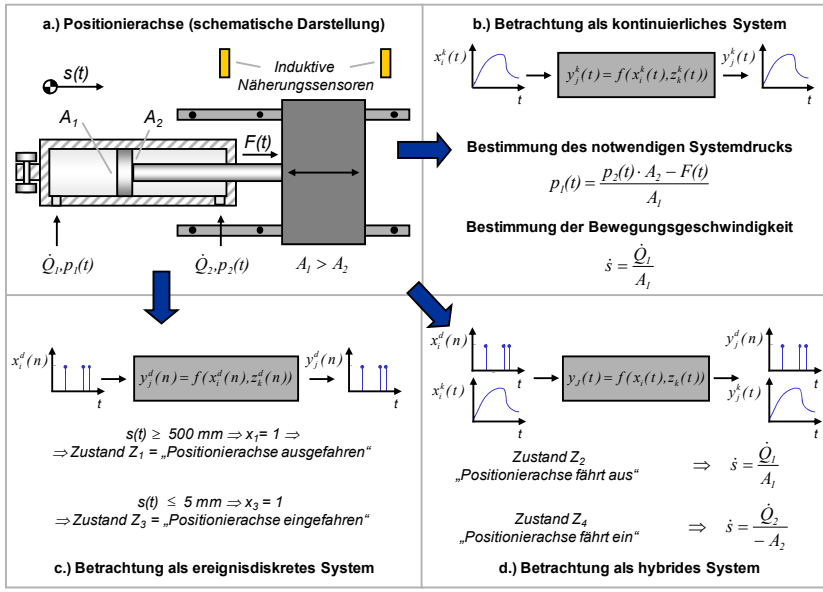


Abbildung 2.7: Systemklassifikation nach dem Betrachtungszweck

Für die Analyse und Synthese vieler technischer Systeme ist jedoch keine der beiden Klassifikationsarten ausreichend, da ihnen sowohl eine kontinuierliche wie auch eine ereignisdiskrete Dynamik zugrunde liegen. Beide Dynamikarten vereinigen sich zu einem neuen Ganzen mit einer anderen Qualität und neuen Eigenschaften. Die Beschäftigung mit diesen sogenannten *hybriden Systemen* stellt ein noch relativ junges Wissensgebiet dar. Eine Übersicht hierzu findet sich beispielsweise in *(Engell u. a 2002; Krebs & Schnieder 2000; Krebs & Schnieder 2001)*.

Hybride Systeme sind folglich solche, die sowohl kontinuierliche wie auch (ereignis-) diskrete Eingangssignale verarbeiten und dem entsprechende Ausgangsgrößen erzeugen. Wesentlich dabei ist, dass sich ein hybrides System nicht in zwei parallel arbeitende Subsysteme, nämlich ein kontinuierliches und ein ereignisdiskretes, unterteilen lässt. Vielmehr findet eine Verkoppelung zwischen ereignisdiskretem und kontinuierlichem Teil statt. Dadurch ist es möglich, dass beide Arten von Eingangsgrößen einen Einfluss auf den jeweils anderen Typ von Systemausgang haben. Wird beispielsweise das Bewegungsverhalten der erläuterten Positionierachse analysiert, so lässt sich diese als kontinuierliches System betrachten, dessen Geschwindigkeit bzw. Position in Abhängigkeit von den geometrischen Größen und dem Ölvolumenstrom vereinfacht dargestellt werden kann. Für die beiden Bewegungsrichtungen des Systems gelten jedoch aufgrund der sich ändernden geometrischen Bedingungen unterschiedliche Parameter,

sodass bei der Beschreibung des Systemverhaltens auch die diskreten Systemzustände „Positionierachse fährt aus“ und „Positionierachse fährt ein“ eine wesentliche Rolle spielen (siehe Abbildung 2.7 d).

Gerade für die Simulation des Verhaltens technischer Systeme ist die beschriebene Einordnung von entscheidender Bedeutung. Zur Erlangung aussagekräftiger und in Bezug auf die Untersuchungsziele relevanter Ergebnisse ist es erforderlich, die der Simulation vorausgehende Modellbildung richtig zu gestalten. Die zugrunde liegenden physikalischen und logischen Zusammenhänge sind in geeigneter Form anhand formaler, mathematischer und somit durch Simulationswerkzeuge interpretierbarer Modelle zu beschreiben. Basis zur Auswahl adäquater Modellierungstechniken ist folglich die Kenntnis der zur Modellbildung und Simulation der jeweiligen Systemklassen zweckmäßigen Methoden und Beschreibungsmittel.

2.2.5 Modellbildung technischer Systeme

Die Modellbildung technischer Systeme ist prinzipiell auf zwei verschiedene Weisen möglich. Zum einen kann zur Herleitung des formal-mathematischen Modells von grundlegenden physikalischen Gesetzmäßigkeiten sowie der konstruktiven Auslegung des betrachteten Produktes ausgegangen werden. Diese Vorgehensweise wird als theoretische Systemanalyse bezeichnet. Zum anderen kann versucht werden, durch Beobachtungen des Systems bei unterschiedlichen Bedingungen zu einer umfassenden Verhaltensbeschreibung zu gelangen. Bei dieser als Identifikation oder experimentelle Systemanalyse bezeichneten Methode gilt es zu ermitteln, welches Systemverhalten (Output) sich als Reaktion auf gewisse äußere Einflüsse (Input) ergibt. Dieser Zusammenhang wird mit passenden mathematischen Funktionen – in der Regel mit differential-algebraischen Gleichungen – dargestellt, die aber meist mit den konkreten Funktionen und der inneren Struktur des Systems nichts gemein haben. Das System wird folglich aus einer „Black-Box“-Sicht behandelt. Seine realen, tatsächlich verhaltensbestimmenden Eigenschaften und Funktionen werden nicht ermittelt *(Bossel 1994, S. 29)*. Grundvoraussetzung für die Identifikation ist die Möglichkeit zur Durchführung von Experimenten. Auf der Basis der messtechnisch ermittelten Ein- und Ausgangsgrößen werden die Koeffizienten der systembeschreibenden Gleichungen abgeleitet *(Lunze 2003, S. 24)*.

Die physikalischen Zusammenhänge von technischen Systemen, wie elektromechanischen und hydraulischen Antrieben oder auch mechanischen Strukturen, sind meist sehr genau bekannt. Der Aufbau von Prototypen oder Versuchsaufbauten zur Erarbeitung von Messreihen dagegen ist außerordentlich kosten- und zeitintensiv. Es ist somit sinnvoll, das mathematische Modell weitest möglich aus den physikalischen Zusammenhängen herzuleiten *(Zirn 2002, S. 7)*.

2.3 Beschreibungsmittel für technische Systeme

2.3.1 Übersicht

Die Modellierung technischer Systeme erfordert den Einsatz von, in Bezug auf den Modellzweck, geeigneten Notationen und Beschreibungsmitteln. Aus der Vielzahl der in den letzten Jahren erarbeiteten Ansätze werden im Folgenden die für den Kontext der Arbeit relevanten, praxisnahen Entwicklungen (bzw. darauf aufbauende Rechnerwerkzeuge) beschrieben. Teilweise wird dabei ein gewisses Verständnis bezüglich prinzipieller Basiskonzepte zur Modellierung technischer Systeme vorausgesetzt. Dies betreffend sei auf den Anhang der Arbeit verwiesen. Dieser beinhaltet eine komprimierte Erläuterung der theoretischen Grundlagen sowie eine Vorstellung fundamentaler Beschreibungsmittel zur Spezifikation der in Abschnitt 2.2.4 beschriebenen Klassen von technischen Systemen.

2.3.2 Unified Modeling Language

Die Unified Modeling Language (UML) in der Version 2.0 ist eine von der OMG entwickelte Modellierungssprache für Softwaresysteme und stellt in diesem Bereich einen Quasi-Standard dar. Durch die Nutzung eindeutig festgelegter grafischer Notationselemente und deren Kombination erlaubt die UML eine transparente und durchgängige Spezifikation softwaretechnischer Systemstrukturen und ihres Verhaltens. Gegenüber den Vorgängerversionen zielen die Änderungen in der aktuellen Version auf die Qualifikation für die Entwicklung technischer Systeme ab. Daher wurden wesentliche Elemente zur Modellierung von Software für eingebettete Systeme in den Standard integriert. So sind beispielsweise die Konzepte der ROOM-Methode *(Selic u. a. 1994; Selic 1998)* sowie der „UML for Real-Time“ *(Douglas 2003)* mittlerweile Teil des Sprachumfangs. Ein weiterer Schwerpunkt wurde auf die Formalisierung der Sprache gelegt, um die notwendige Eindeutigkeit und Präzision der erstellten Modelle zu ermöglichen.

In der UML 2.0 sind insgesamt dreizehn Diagrammtypen definiert, von denen sechs für die Abbildung struktureller Aspekte und sieben für die Verhaltensmodellierung vorgesehen sind. Entsprechend der intendierten Zielsetzung sowie den Eigenschaften der zu entwickelnden Software kann die Entwicklerin bzw. der Entwickler die geeigneten Modellierungsmöglichkeiten auswählen. Die folgende Darstellung umfasst eine knappe Abhandlung einiger wesentlicher und vor allem aus der Sicht der Automatisierungstechnik relevanter Diagramme. Für eine ausführliche Beschreibung der einzelnen Diagrammtypen sei auf die entsprechenden Spezifikationen des Standards *(OMG 2005a; OMG 2005b)* oder äquivalente Fachliteratur *(Booch u. a. 2006; Hitz u. a. 2005; Jeckle u. a. 2004)* verwiesen.

Komponentendiagramm

Im Komponentendiagramm (siehe Abbildung 2.8) werden die Organisation und die Abhängigkeiten zwischen einzelnen Systemkomponenten (Wirkstruktur) abgebildet. Eine Komponente ist im Kontext des Diagramms als ein modularer Teil des Systems zu betrachten. Sie bietet eine klar definierte Funktionalität an, indem sie das Verhalten und die Struktur partikulärer Teilsysteme kapselt und abstrahiert und zur Kommunikation mit ihrer Umgebung wohldefinierte Schnittstellen zur Verfügung stellt. Diese werden durch Ports repräsentiert.

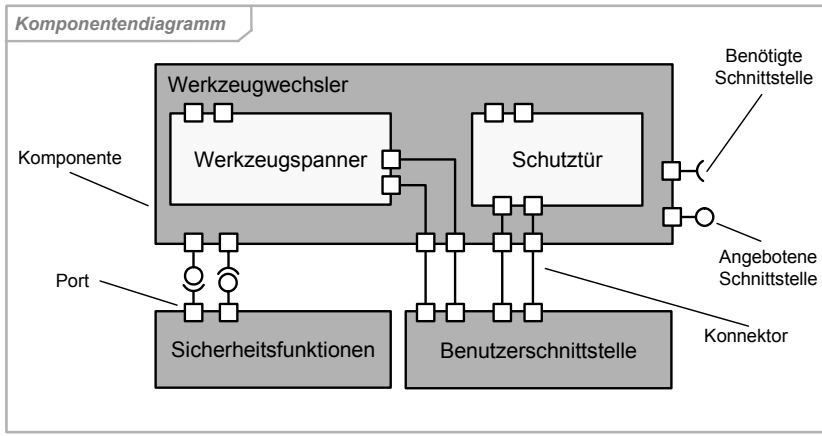


Abbildung 2.8: Komponentendiagramm

Während Ports lediglich beschreiben, welche Informationen bzw. Funktionen von einer Komponente benötigt oder zur Verfügung gestellt werden, wird der Informationsfluss im System durch Konnektoren abgebildet. Diese verbinden einzelne Ports miteinander und realisieren so die Beziehungen im modellierten System. Eine detailliertere Spezifikation der Schnittstellen einer Komponente ist möglich, indem unterschieden wird, ob ein Port eine angebotene (Systemausgang) oder benötigte (Systemeingang) Schnittstelle repräsentiert. Hierzu wird die sogenannte Stecker-Buchse-Notation[16] verwendet. Die Modularisierung erlaubt es auch, Komponenten aus einzelnen Subkomponenten zusammenzusetzen. Damit wird die Grundlage zur Bildung hierarchischer Strukturen geschaffen.

Sequenzdiagramm

Das Sequenzdiagramm (siehe Abbildung 2.9) ist ein auf den Message Sequence Charts (MSC[17]) basierender Diagrammtyp. Es stellt den Informationsaustausch zwischen ver-

[16] Wird auch als "Lollipop-Notation" bezeichnet.

[17] Message Sequence Charts sind ein ISO-Standard aus dem Telekommunikationsbereich.

schiedenen Komponenten zur Erfüllung definierter Funktionen und somit deren Interaktion dar. Entlang der sogenannten Lebenslinie können die beteiligten Komponenten zum Zwecke der Funktionserfüllung Nachrichten austauschen. Dies geschieht in Form von Signalen oder Operationsaufrufen. Die Reihenfolge der Kommunikation zwischen den Komponenten wird als „Trace“ bezeichnet und beschreibt die Funktionslogik.

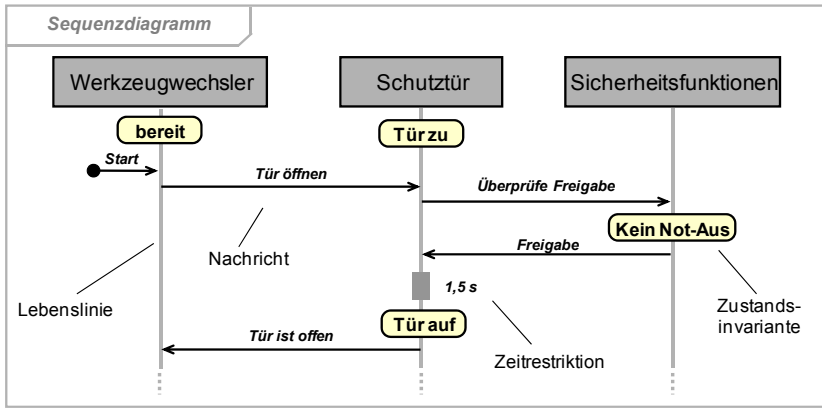


Abbildung 2.9: Sequenzdiagramm

Zusätzlich bieten Sequenzdiagramme eine Reihe von Konstrukten zur detaillierten Modellierung des Systemverhaltens. So können beispielsweise Zeitrestriktionen abgebildet werden, indem das Senden bzw. Empfangen von Nachrichten an eine definierte Zeitdauer oder an konkrete Zeitpunkte gebunden wird. Ebenso sind durch sogenannte Zustandsinvarianten Vorbedingungen definierbar, die erfüllt sein müssen, um das weitere Ausführen einer Interaktion zu erlauben. Darüber hinaus wurden in der aktuellen Version auch Notationselemente zur Darstellung von Kontrollstrukturen sowie zur Modularisierung integriert.

Zustandsdiagramm

Im Gegensatz zum Sequenzdiagramm liegt der Fokus des Zustandsdiagramms (siehe Abbildung 2.10) auf der Beschreibung des Verhaltens einzelner Softwarekomponenten durch Zustandsautomaten. Die auf Harel-Automaten (siehe Abschnitt 11.2.2) basierenden Diagramme erlauben die Definition einzelner Zustände, der Zustandsübergänge und der damit verbundenen Übergangsbedingungen sowie die Zuordnung von bestimmten Aktivitäten zu den Zuständen bzw. Transitionen im Diagramm.

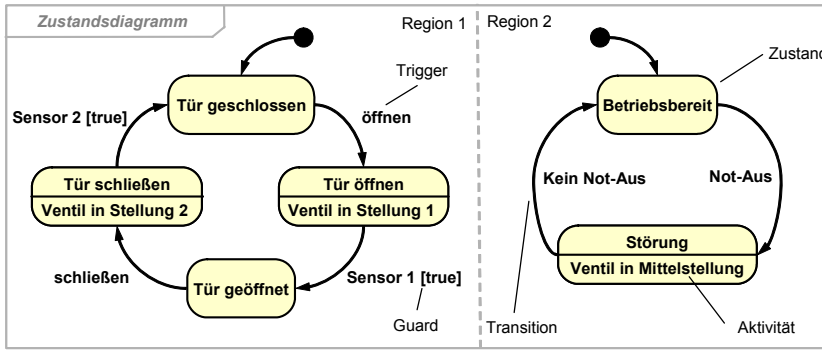


Abbildung 2.10: Zustandsdiagramm

Diese fundamentale Basis zur Verhaltensmodellierung wird durch Konzepte zur Zustandsverfeinerung, zur Klassifizierung von Aktivitäten und Zustandsübergängen und durch diverse Pseudozustände erweitert. Aufgrund der Möglichkeit zur Definition von Regionen ist auch die Spezifikation von nebenläufigem[18] Systemverhalten realisierbar.

Aktivitätsdiagramm

Auch bei den Aktivitätsdiagrammen (siehe Abbildung 2.11) steht die Modellierung des Verhaltens im Mittelpunkt. Eine besondere Eignung ist in Bezug auf die Spezifikation von komplexen Funktionen, Algorithmen und Datenflüssen festzustellen. Grundlegende Bausteine dieses Diagrammtyps sind elementare Aktionen, Kontrollelemente und Bedingungen zur Steuerung sowie Kanten, die einzelne Aktionen und Kontrollelemente miteinander verbinden. Im Gegensatz zu Zustandsdiagrammen implementieren Aktivitätsdiagramme jedoch ein Markenkonzept und sind damit semantisch stark mit Petri-Netzen (siehe Abschnitt 11.2.2) verwandt.

Das Prinzip der Markierung setzt für die Durchführung einer Aktion voraus, dass die vorstehende Aktion ausgeführt wird und vorhandene Kontrollbedingungen erfüllt sind. Die Modellierung von nebenläufigem Systemverhalten ist durch spezielle Kontrollelemente zur Synchronisation und Parallelisierung von Aktionen realisierbar.

[18] Unter Nebenläufigkeit ist die Unabhängigkeit und damit die Parallelisierbarkeit der einzelnen Verhaltensbestandteile des Systems zu verstehen.

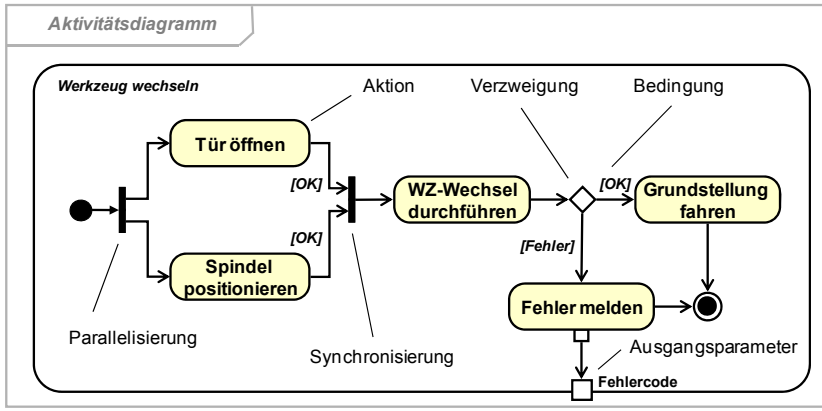


Abbildung 2.11: Aktivitätsdiagramm

Der UML liegt ein sogenanntes *Metamodell* zu Grunde, in dem die einzelnen Modellierungselemente und Diagrammtypen definiert sind. Das Metamodell stellt den Sprachkern dar, der die Zusammenhänge zwischen den einzelnen Modellelementen formal festlegt. Diagramme in der UML sind daher nicht als vollständig voneinander losgelöst zu betrachten. Vielmehr wird ein flexibler, sichtenorientierter Diagrammbegriff in den Vordergrund gestellt. Dies erlaubt es, die UML gezielt an verschiedene Anwendungsbereiche anzupassen, indem Teile des Sprachumfanges erweitert bzw. neu definiert werden. Diese Änderungen werden in sogenannten *UML-Profilen*[19] zusammengefasst und beschrieben. So existieren beispielsweise Profile zur Modellierung von Echtzeitaspekten in technischen Systemen oder die Entwicklung verteilter Anwendungen.

Die Bildung von Profilen basiert auf drei wesentlichen Mechanismen: *Stereotypen* sind eine Möglichkeit, um Sprachelemente innerhalb der UML in ihrer Semantik einzuschränken. Durch die Festlegung von speziellen Bezeichnungen und graphischen Darstellungen für bestimmte Objekte wird deren Bedeutung innerhalb eines Modells eindeutig definiert. Ergänzend dazu wird mit sogenannten *Tagged Values* (Schlüsselwort-Wert-Paare) ein bestehendes Sprachelement um definierte Eigenschaften ergänzt. Dies geschieht, indem Attribute vorgegeben werden, denen ein bestimmter Wert zugewiesen werden kann. Die *Object Constraint Language* (OCL) schließlich erlaubt es, einzelne Modellelemente sowie die Beziehungen zwischen denselben einzuschränken und damit die Aussagekraft zu präzisieren. Neben der in *(OMG 2006)* definierten formaldeklarativen Spezifikation ist auch eine Beschreibung der Randbedingungen in natürlicher Sprache möglich.

[19] Ein Profil ist eine speziell an ein Anwendungsgebiet adaptierte Variante der UML.

2.3.3 Signalinterpretierte Petri-Netze

Die Signalinterpretierten Petri-Netze (SIPN) stellen eine Verfeinerung der B/E-Netzklasse (siehe Abschnitt 11.2.2) dar und eignen sich somit zur Spezifikation des Verhaltens ereignisdiskreter Systeme. Im Gegensatz zur allgemeinen Definition von Petri-Netzen verfügen sie über explizite Eingaben und Ausgaben, die den Netzelementen (Stellen und Transitionen) zugeordnet werden. Darüber hinaus ist aufgrund der vollständigen formalen Spezifikation eine direkte Überführung in Steuerungscode möglich *(Litz 2005, S. 353-364)*.

Formal lässt sich ein Signalinterpretiertes Petri-Netz nach *(Litz 2005, S. 245)* als 8-Tupel der Form

$$SIPN(S, T, K, M_0, E, A, SB, AK)$$

definieren. Dabei bezeichnet:

$S = \{S_1, \dots, S_n\} \quad n \in \mathbb{N}$	die Menge aller Stellen (Plätze),
$T = \{T_1, \dots, T_k\} \quad k \in \mathbb{N}$	die Menge aller Transitionen mit
$T \cap S = \emptyset$	
$K \subseteq T \times S \cup S \times T$	die Menge aller Kanten,
$M_0^d \colon S \to \{0,1\}$	die Anfangsmarkierung,
$E = \{e_1, \dots, e_p\} \quad p \in \mathbb{N}$	die Menge aller Eingangssignale,
$A = \{a_1, \dots, a_q\} \quad q \in \mathbb{N}$	die Menge aller Ausgangssignale,
$SB = \{SB(T_1), \dots, SB(T_k)\}$	die Menge aller Schaltbedingungen mit
$SB(T_i) = b_i(e_1, \dots, e_p), \; i = 1, \dots, k$	und
$AK = \{a_1, \dots, a_n\}$	die Menge aller Aktionen mit
$a_i \in \{0,1,-\}, \quad i = 1, \dots, q.$	

Aus obiger Darstellung wird ersichtlich, dass der Informationsfluss in das Netz immer an die Transitionen gebunden ist und sich dabei sogenannter Schaltregeln bedient. Nur wenn die der Transition vorgeschalteten Stellen des Netzes markiert und die nachgeschalteten Stellen frei sind, gleichzeitig aber auch die der Transition zugeordnete Schaltbedingung erfüllt wird, ist der Markenfluss zu den nachfolgenden Stellen möglich. Der Informationsfluss aus dem Netz hingegen ist mit den Stellen des Netzes verknüpft (siehe Abbildung 2.12). Sobald die entsprechende Stelle aktiviert ist, werden die damit verknüpften Aktionen auch ausgeführt und damit die Ausgangssignale gebildet.

Eine Erweiterung der beschriebenen Netzart bilden die zeitbewerteten SIPN. Hierbei wird die Schaltregel um einen zeitabhängigen Gültigkeitsbereich ergänzt. Demnach wird ein Markenfluss im Netz nur möglich, wenn die oben genannten Schaltbedingungen erfüllt sind, gleichzeitig aber eine bestimmte Zeitdauer τ nicht über- bzw. unter-

schritten wird. Die Zeit beginnt per Definition zu laufen, wenn die der Transition vorgeschaltete Stelle markiert wird. Bei fehlenden Angaben gilt $0 \leq \tau \leq \infty$.

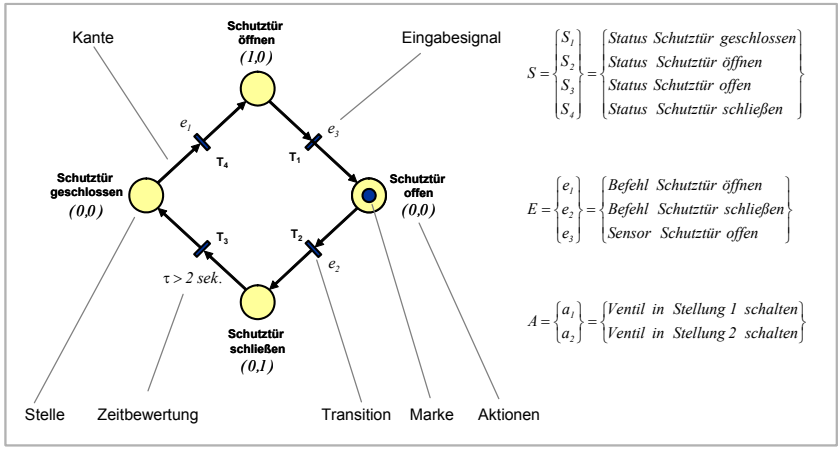


Abbildung 2.12: Zeitbewertetes, Signalinterpretiertes Petri-Netz

Ähnliche Ansätze zu den Signalinterpretierten Petri-Netzen stellen die Standards GRAFCET bzw. die Sequential Function Charts (Ablaufsprache) der IEC 61131-3 dar. Bezüglich einer detaillierten Gegenüberstellung der genannten Beschreibungsmittel wird an dieser Stelle auf *(Frey 2002)* verwiesen.

2.3.4 Sprachen der IEC 61131-3

Die IEC 61131-3 ist ein internationaler Standard zur Programmierung Speicherprogrammierbarer Steuerungen (SPS) und deren Peripherie, der mittlerweile eine breite Zustimmung gefunden hat und von den meisten kommerziell verfügbaren Steuerungen und Entwicklungsumgebungen unterstützt wird. Er umfasst die Festlegung der Syntax und Semantik von zwei textbasierten (AWL, ST[20]) und zwei graphischen (FBS, KOP[21]) Programmiersprachen (siehe Abbildung 2.13). Darüber hinaus ist die Definition struktureller Beziehungen in Form eines Software-, Kommunikations- und Programmiermodells und die Spezifikation elementarer und abgeleiteter Datentypen Teil des Standards. Mit der Ablaufsprache (AS) stehen als Ergänzung zu den Programmiersprachen Elemente zur Strukturierung der internen Organisation von SPS-Programmen zur Verfügung. Im Folgenden werden die Programmiersprachen und Programm-Organisationseinheiten des Standards in kompakter Form erläutert. Bezüglich detail-

[20] AWL = Anweisungsliste, ST = Strukturierter Text.

[21] FBS = Funktionsbausteinsprache, KOP = Kontaktplan.

lierter Betrachtungen wird auf die entsprechende Norm *(DIN EN 61131-3)* oder geeignete Fachliteratur *(Lepers 2005; Wellenreuther & Zastrow 2001)* verwiesen.

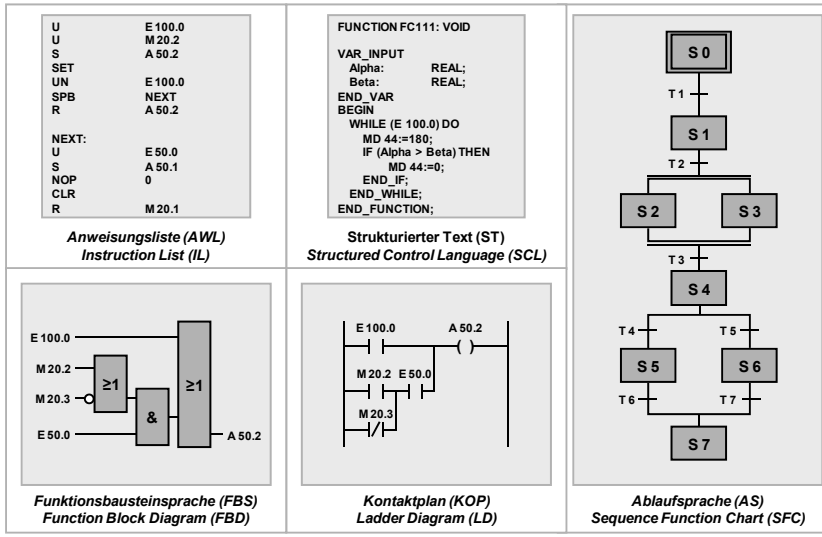


Abbildung 2.13: Programmiersprachen der IEC 61131-3

Anweisungsliste

Die Anweisungsliste ist eng an die Assemblersprache angelehnt und dient hauptsächlich zur Definition logischer Verknüpfungen. Sie ist dadurch gekennzeichnet, dass innerhalb jeder Anweisung nur ein einzelner Operand verarbeitet werden kann und die Strukturierungsmöglichkeiten auf die Festlegung von Sprungfunktionen begrenzt sind. Daher sind komplexere Aufgaben nur durch Experten mit entsprechender Erfahrung umsetzbar. Das Verständnis bereits bestehender Programmlogik erfordert trotzdem einen hohen Einarbeitungsaufwand. Dennoch werden mit höheren Sprachen erstellte Programme aufgrund der inhärenten Leistungsfähigkeit meist in AWL transformiert.

Strukturierter Text

Der strukturierte Text ist die einzige innerhalb der Norm definierte, höhere Programmiersprache. Die Pascal-ähnliche Sprache verfügt über die gängigen Strukturierungsmittel von Hochsprachen (Schleifen, bedingte Anweisungen, usw.), besitzt aber darüber hinaus noch speziell auf die Steuerungstechnik zugeschnittene Möglichkeiten zur Implementierung der Programmlogik.

Kontaktplan

Diese graphische Programmiersprache hat ihren Ursprung im Bereich der elektromechanischen Schützsteuerungen und ist den entsprechenden Stromlaufplänen (um 90

Grad gedreht) nachempfunden. Das Systemverhalten wird durch die Darstellung des Stromflusses in einzelnen Netzwerken beschrieben. Die linke und rechte Seite des Diagramms definieren Stromschienen (linke Seite potentialbehaftet, rechte Seite potentialfrei). Die Verbindungen zwischen den Schienen führen Strom zu Kontakten (Boolesche Variablen oder Eingänge), die abhängig von ihrem logischen Zustand die Weiterführung des Stroms zu Spulen (Boolesche Variablen, Ausgänge, usw.) erlauben oder unterbinden.

Funktionsbausteinsprache

Der Ursprung der Funktionsbausteinsprache liegt im Bereich der Signalverarbeitung durch funktionsorientierte, logische Ablaufketten *(Pickhardt 2000)*. Analog zum Kontaktplan werden die Programmanweisungen in Netzwerke gegliedert. Diese beinhalten entsprechend den implementierten Funktionen eine Reihe von elementaren Funktionsblöcken (z. B. Logikbausteine, Vergleichsbausteine oder mathematische Funktionen). Der Signalfluss innerhalb der Netzwerke wird durch die Verbindung der Ein- und Ausgänge der elementaren Bausteine realisiert und verläuft analog zum Kontaktplan von links nach rechts.

Ablaufsprache

Die Ablaufsprache existiert in einer textuellen und einer graphischen Variante und entspricht im Wesentlichen einer programmtechnischen Umsetzung eines Petri-Netzes. Sie ist keine Programmiersprache im eigentlichen Sinne, sondern ein den dargestellten Sprachen übergeordnetes Hilfsmittel zur Strukturierung und Steuerung definierter, sequentieller, aber auch parallel laufender Prozesse. Während die bislang dargestellten Sprachen zur Implementierung von Verknüpfungsteuerungen[22] zweckmäßig sind, ist die Ablaufsprache zur anschaulichen Umsetzung von Ablaufsteuerungen[23] geeignet. Die Aufgabe des Programmierers besteht darin, die einzelnen Schritte und Transitionen des zu automatisierenden Prozesses festzulegen. Jedem Schritt wird eine Menge von Aktionen, jeder Transition eine Übergangsbedingung zugeordnet. Zur Festlegung derselben können die bereits beschriebenen Programmiersprachen verwendet werden. Die Ablaufreihenfolge wird durch die Verknüpfung der einzelnen Schritte mittels Transitionen bestimmt. Die Ausführung der Aktionen erfolgt nur dann, wenn der zugehörige Schritt aktiv ist. Sobald die nachgeschaltete Transitionsbedingung erfüllt ist, wird der Schritt deaktiviert und die nachfolgenden Schritte werden aktiv.

[22] Verknüpfungsteuerungen ordnen in Abhängigkeit von Eingangssignalen und internen Systemzuständen den Ausgangssignalen im Sinne Boolescher Verknüpfungen bestimmte Werte zu.

[23] Ablaufsteuerungen sind Steuerungen mit einem zwangsläufig schrittweisen Ablauf. Dem entsprechend erfolgt das Weiterschalten in den programmgemäß nächsten Schritt in Abhängigkeit definierter Weiterschaltbedingungen. Es werden zeitgeführte und prozessgeführte Ablaufsteuerungen unterschieden, je nachdem, ob die Erfüllung der Weiterschaltbedingungen durch den gesteuerten Prozess oder zeitabhängig erfolgt.

Die Ablaufsprache ist im Wesentlichen eine Weiterentwicklung der Entwurfssprache GRAFCET *(DIN EN 60848)*. Die semantischen Unterschiede zwischen den beiden Spezifikationen werden trotz der syntaktischen Ähnlichkeit durch das jeweilige Anwendungsgebiet bestimmt. Während GRAFCET einer semiformalen Spezifikation des Verhaltens, unabhängig von der Implementierung, entspricht, stellt die Ablaufsprache eine Umsetzung für Speicherprogrammierbare Steuerungen dar. So bietet GRAFCET beispielsweise eine Reihe von Möglichkeiten zur Strukturierung und Hierarchisierung der Beschreibung, eine entsprechende Umsetzung bzw. Transformation in die Ablaufsprache existiert jedoch nicht. Eine detaillierte Gegenüberstellung zwischen GRAFCET und der Ablaufsprache findet sich in *(DIN EN 60848)*.

Programm-Organisationseinheiten
Zur Strukturierung der Steuerungsprogramme unterscheidet die IEC 61131-3 Funktionen, Funktionsbausteine und Programme. Im Gegensatz zu Funktionen, die bei gleichen Eingangsgrößen immer dasselbe Ergebnis liefern, arbeiten Funktionsbausteine mit einem eigenen Datensatz. Sie besitzen also ein Gedächtnis, das im Sinne der Systemtheorie den Zustandsvariablen entspricht. Sie können außerdem mehr als eine Ausgangsgröße berechnen. Gemeinsam ist beiden Konstrukten, dass sie zur Aufgabenerfüllung selbst wieder auf andere Funktionen bzw. Funktionsbausteine zurückgreifen können. Programme bilden die oberste Ebene eines SPS-Programms. Sie verwenden zur Problemlösung die beiden untergeordneten Programm-Organisationseinheiten, unterscheiden sich ansonsten jedoch nur wenig von den Funktionsbausteinen. Die wesentliche Differenz besteht darin, dass sie nicht von anderen Programmen oder Funktionsbausteinen aufrufbar sind. Gemeinsam ist allen drei Konstrukten die Forderung nach Rekursivitätsfreiheit, um die zyklische Abarbeitung durch die Steuerung zu gewährleisten.

2.3.5 MATLAB/Simulink

Während sich die genannten Verfahren zur Modellierung bzw. Programmierung ereignisdiskreter Systeme eigen, ist MATLAB *(Grupp & Grupp 2006, Denier & Otto 2005)* ein weit verbreitetes und sehr mächtiges Modellierungs- und Simulationswerkzeug für kontinuierliche bzw. hybride Systeme. Das Softwaresystem besteht im Wesentlichen aus einem Kern zur numerischen Lösung des systembeschreibenden Gleichungssystems sowie einer graphischen Benutzeroberfläche. Das darauf aufbauende Modul Simulink stellt einen graphischen Editor dar, mit dem Ziel, die Eingabe der Systemgleichungen zu vereinfachen. Dies geschieht, indem elementare mathematische Blöcke (algebraische Funktionen, Integralglieder, Differentialglieder, usw.) graphisch entsprechend den mathematischen Zusammenhängen im modellierten System miteinander verknüpft werden (Blockschaltbilder). Neben den zur Verfügung stehenden Standardblöcken erlaubt Simulink auch die Entwicklung eigener Blöcke *(Angermann*

u. a. 2002, S. 231). Durch Zusammenfassung einzelner Modellbestandteile zu Subsystemen ist der Aufbau einer hierarchischen Systemstruktur möglich.

Bei der Modellbildung werden zwei grundlegende Typen von Blöcken unterschieden. Nichtvirtuelle Bausteine sind Teil der Struktur und des Verhaltens des modellierten Systems. Das Ändern, Entfernen oder Hinzufügen hat direkte Auswirkungen auf das Modell. Virtuelle Bausteine hingegen dienen lediglich der graphischen Organisation, um zum einen das Modell übersichtlich zu gestalten und zum anderen Ergebnisse oder Signale zu visualisieren.

Neben den zur Verfügung stehenden Standardbibliotheken für Simulink existieren eine Reihe von kommerziellen Zusatzpaketen (Toolboxen), die speziell auf bestimmte Anwendungsgebiete zugeschnitten sind. So gibt es beispielsweise Erweiterungen für die Bild- und Mustererkennung oder die Entwicklung von Software für Echtzeitsysteme. Die Toolbox „Stateflow“ ist speziell für den Entwurf endlicher Automaten entwickelt worden, um die Integration ereignisdiskreter Systembestandteile in die Modelle zu ermöglichen. Abbildung 2.14 zeigt exemplarisch einen Ausschnitt aus dem Modell eines PKW-Automatikgetriebes.

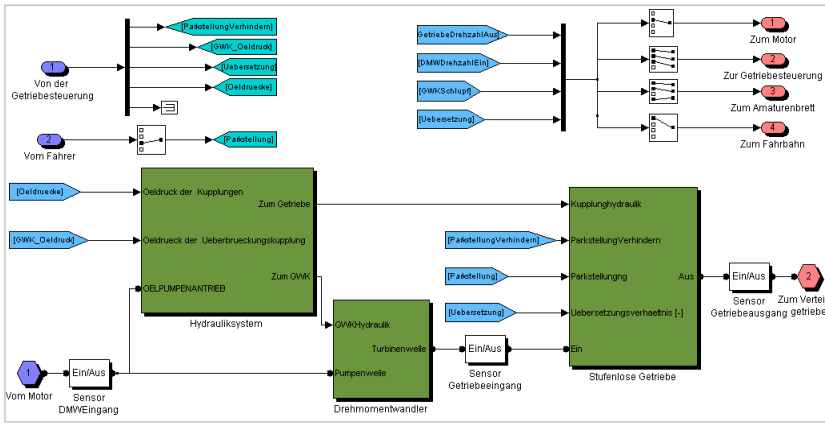


Abbildung 2.14: Ausschnitt eines Modells in MATLAB/Simulink

2.3.6 Modelica

Modelica ist eine objektorientierte Modellierungssprache zur multidisziplinären Spezifikation mathematischer Modelle dynamischer physikalischer Systeme *(Fritzson 2004)*. Der von einer europäischen Forschergruppe (Modelica Association) entwickelte, offene Standard *(Modelica 2005)* ist durch folgende Eigenschaften gekennzeichnet:

- Modelle können in Modelica mittels differential-algebraischer und diskreter Gleichungen beschrieben werden. Aufgrund der damit fehlenden gerichteten Abbildung von Eingangsgrößen auf Ausgangsgrößen ist eine akausale Modellbildung (siehe Abschnitt 11.2.1) möglich, wodurch sich die Wiederverwendbarkeit signifikant verbessert. Alternativ ist aber auch eine kausale Modellbildung mittels Zuweisungen[24] realisierbar.
- Die Sprache umfasst analog zu objektorientierten Programmiersprachen ein allgemeines Klassenkonzept. Neue Komponenten können deshalb effizient durch die Verfeinerung bestehender Komponenten gebildet werden.
- Modelica basiert auf einem stringenten Komponentenkonzept. Damit verbunden ist die eindeutige Abgrenzung der Modellbausteine gegenüber ihrer Umwelt (Systemgrenze) und die Spezifikation definierter Schnittstellen. Somit besteht die Möglichkeit der Hierarchisierung und der Verknüpfung einzelner Komponenten zu komplexen Modellen.

Analog zu Simulink existieren diverse graphische Entwicklungsumgebungen (MathModelica, Dymola, OpenModelica), die einen Aufbau komplexer Modelle ermöglichen. Die unter Nutzung des Standards formulierte Spezifikation wird anschließend von einem Compiler automatisiert in eine Zustandsraumdarstellung überführt und mittels geeigneter numerischer Algorithmen gelöst. Einen wesentlichen Vorteil stellt auch die Bereitstellung einer umfangreichen Komponentenbibliothek für verschiedene Anwendungsfelder dar, die einen effizienten Modellaufbau erlaubt. Abbildung 2.15 zeigt exemplarisch das Modell eines einfachen Feder-Masse-Dämpfer-Systems in Modelica.

[24] Eine Zuweisung ist eine Abbildungsvorschrift, die eine oder mehrere Eingangsgrößen auf eine einzelne Ausgangsgröße abbildet. Der Wert der linken Seite wird durch die rechte Seite definiert (z. B. y := 5 + 1).

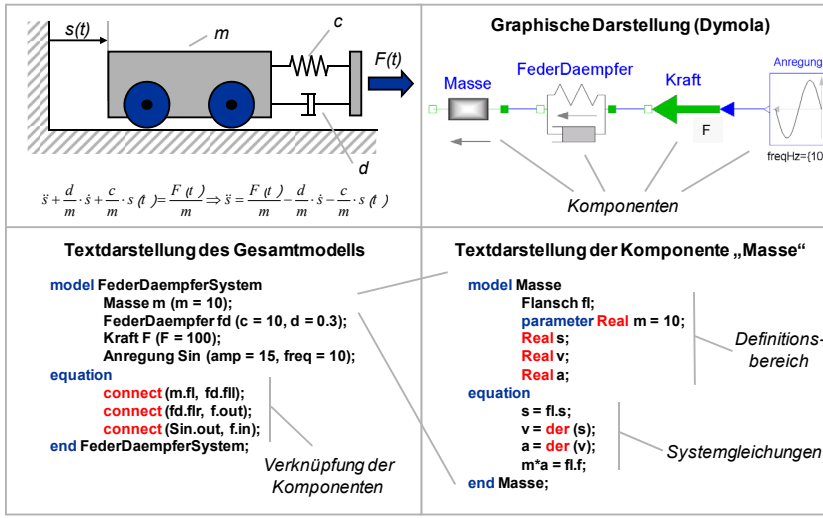


Abbildung 2.15: Feder-Masse-Dämpfer-System in Modelica

Das Modell besteht aus den Komponenten „Masse“ und „Feder-Dämpfer“ sowie der anregenden Kraft. Jede der Komponenten verfügt über bestimmte Variablen, Konstanten und Parameter sowie über einen eigenen Bereich zur Festlegung der Systemgleichungen. Über definierte Schnittstellen, die den Stoff-, Energie- und Informationsfluss im System abbilden, werden die Einzelkomponenten zu einem Gesamtmodell verbunden.

2.3.7 Systems Modeling Language

Die Systems Modeling Language (SysML) ist ein Profil der UML zum durchgängigen Entwurf von Systemen im Sinne des Systems Engineering. Die von der INCOSE[25] entwickelte Erweiterung grenzt sich gegenüber der UML dadurch ab, dass sie nicht den vollen Sprachumfang derselben nutzt. Lediglich ausgewählte Diagramme zur Modellierung von Struktur und Verhalten werden verwendet bzw. neu definiert. So werden beispielsweise zur Strukturmodellierung Blockdiagramme[26] (siehe Abbildung 2.16) vorgeschlagen, die im Wesentlichen den Komponentendiagrammen der UML entsprechen. Andererseits wird die SysML durch zusätzliche Notationen und Diagrammtypen ergänzt. Diese beziehen sich vor allem auf frühe Entwicklungsphasen. So beinhaltet das Profil *(OMG 2007)* mit dem Anforderungsdiagramm ein spezielles

[25] International Council on Systems Engineering.

[26] Eine strukturelle Einheit wird in der SysML als Block bezeichnet.

Konstrukt zur Modellierung der Erwartungen und Restriktionen bezüglich des zu entwickelnden Systems.

Zusätzlich verfügt die SysML über Erweiterungen zur Modellierung von kontinuierlichem Verhalten. Hierzu wurde ein spezieller Typ von Ports (Flow Port) definiert, der die Schnittstellen bezüglich des Daten-, Energie- und Stoffflusses im modellierten System repräsentiert. Die Verarbeitung der Daten durch dezidierte Systembausteine wird entsprechend der Art des Signalflusses mit speziellen Stereotypen gekennzeichnet.

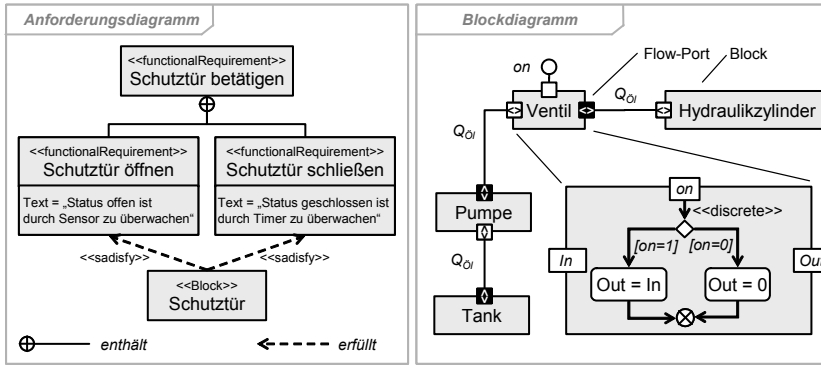


Abbildung 2.16: Anforderungsdiagramm und Blockdiagramm

Die SysML stellt eine noch sehr junge Sprache dar, die noch Schwächen enthält und daher weiter verfeinert werden muss. Diese Unzulänglichkeiten beziehen sich nicht nur auf die Anforderungen bezüglich des Systems Engineerings, sondern auch auf die Genauigkeit der Sprache. Sie eignet sich deshalb primär zur Grobspezifikation eines technischen Systems, unabhängig von spezifischen Fachdisziplinen *(Weilkiens 2006, S. 157-162)*.

2.4 Zusammenfassung

Im Rahmen von Kapitel 2 wurden die zum Verständnis der Arbeit notwendigen Grundlagen erläutert. An eine Übersicht zur modellgetriebenen Softwareentwicklung im Allgemeinen schloss sich die Darstellung verschiedener Möglichkeiten zu deren Umsetzung an. Im Hinblick auf die Anwendung des modellgetriebenen Ansatzes bei der Entwicklung von Steuerungssoftware für automatisierte Fertigungssysteme wurden technische Systeme klassifiziert und die grundlegenden Zusammenhänge bei deren Modellierung erläutert. Eine Einführung von im Umfeld der Arbeit relevanten Beschreibungsmitteln für technische Systeme stand am Ende des Kapitels.

3 Analyse des Aufgabengebietes

3.1 Übersicht

Um die in Abschnitt 2.1.2 erläuterten Vorteile einer modellgetriebenen Softwareentwicklung gegenüber konventionellen Methoden nutzen zu können, ist eine Adaption der eingesetzten Methoden und Modellierungssprachen an die jeweilige Anwendungsdomäne zwingend erforderlich (siehe Abschnitt 2.1.3). Voraussetzung hierfür ist die genaue Kenntnis der domänenspezifischen Rahmenbedingungen und Entwicklungsmethoden sowie der aktuell bestehenden methodischen und technologischen Defizite. Im Folgenden werden daher zunächst der Aufbau automatisierter Fertigungssysteme beschrieben und der Betrachtungsbereich eingeschränkt. Darauf aufbauend wird das in der industriellen Praxis vorherrschende Entwicklungsvorgehen analysiert. Auf der Basis des aktuell bestehenden Handlungsbedarfs, aber auch zukünftiger technologischer Trends, werden abschließend detaillierte, domänenspezifische Anforderungen abgeleitet, die bei der Entwicklung eines modellgetriebenen Ansatzes zu berücksichtigen sind.

3.2 Aufbau automatisierter Fertigungssysteme

3.2.1 Klassifizierung automatisierter Fertigungssysteme

Entsprechend der Definition aus Abschnitt 11.1 werden unter automatisierten Fertigungssystemen Anlagen verstanden, die aus einer oder mehreren Werkzeugmaschinen bestehen und zur Fertigung von Produkten oder Teilen davon zusammenarbeiten. Nach *(DIN 69651)* werden Einzelmaschinen und Mehrmaschinensysteme unterschieden. Bei Einzelmaschinen wird eine Klassifizierung in einfache Maschinen, NC-Maschinen und Bearbeitungszentren vorgenommen. Während einfache Maschinen in der Regel über keine Automatisierung verfügen, besitzen NC-Maschinen die für eine rechnergestützte Fertigung notwendige Numerische Steuerung (NC). Ein Bearbeitungszentrum erweitert eine NC-Maschine um eine automatische Werkzeugwechseleinrichtung und einen dazugehörigen Werkzeugspeicher.

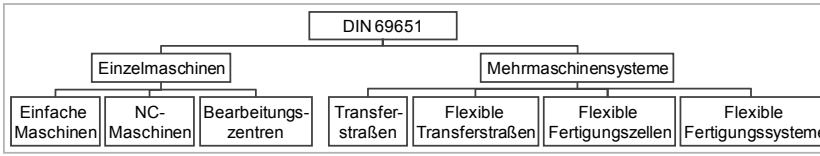


Abbildung 3.1: Einteilung von Fertigungssystemen nach DIN 69651

Bei Mehrmaschinensystemen existiert nach *(Weck & Brecher 2005)* eine Unterteilung in Transferstraßen, Flexible Transferstraßen, Flexible Fertigungszellen und Flexible Fertigungssysteme. Transferstraßen bestehen aus einer Vielzahl von Bearbeitungsstationen, die für eine spezielle Bearbeitungsaufgabe seriell miteinander gekoppelt wer-

den. Der Transport der zu bearbeitenden Bauteile ist gerichtet, linear, in der Regel getaktet und wird durch eine automatisierte Transporteinrichtung realisiert. Flexible Transferstraßen dienen hingegen zur Herstellung eines bestimmten Teilespektrums. Bei den Bearbeitungsstationen handelt es sich nicht um Einzweckmaschinen, sondern um mehrachsige Bearbeitungszentren, die über ein Werkzeugmagazin und eine Werkzeugwechseleinrichtung verfügen. Der Werkstücktransport zwischen den ebenfalls in Reihe geschalteten Stationen wird analog zu den starren Transferstraßen durch ein automatisiertes Transportsystem realisiert. Der Bearbeitungsprozesses läuft getaktet ab oder wird durch Zwischenpuffer entkoppelt. Eine flexible Fertigungszelle zeichnet sich gegenüber den genannten Systemen durch ein wesentlich breiteres Bauteilspektrum aus. Sie besteht aus einem Bearbeitungszentrum, das zusätzlich über einen Werkstückspeicher und eine automatisierte Werkstückwechseleinrichtung verfügt. Dadurch kann ein derartiges System über einen größeren Zeitraum selbstständig arbeiten. Flexible Fertigungssysteme hingegen bestehen aus mehreren Maschinen, die über ein automatisiertes Transportsystem miteinander verbunden werden. Im Gegensatz zu den Flexiblen Transferstraßen wird die Versorgung der einzelnen Maschinen mit Werkstücken nicht durch deren Reihenschaltung eingeschränkt. Durch ein entsprechend gestaltetes Transportsystem kann die Reihenfolgeansteuerung der einzelnen Bearbeitungseinheiten beliebig realisiert und an die Anforderungen der Produktion angepasst werden.

3.2.2 Struktur automatisierter Fertigungssysteme

Unabhängig von der Klassifikation sind automatisierte Fertigungssysteme zum Zwecke der Komplexitätsreduktion hierarchisch strukturierbar. Entsprechende Gliederungen finden sich beispielsweise in *(Pahl u. a. 2005)* oder *(VDI 3260)*. Abbildung 3.2 zeigt eine Einteilung in Anlehnung an *(DIN 40150)*.

Demnach kann ein Fertigungssystem nach der Bau- oder der Funktionsstruktur gegliedert werden. Die unterste Betrachtungsebene bilden die *Elemente*. Bezogen auf Fertigungssysteme sind dies beispielsweise Sensoren, Ventile oder Motoren. Durch Kombination einzelner Elemente zu *Einheiten* können bestimmte elementare Basisfunktionen realisiert werden, die sich durch einen hohen Standardisierungsgrad auszeichnen. Exemplarisch genannt seien pneumatische oder hydraulische Steuerketten zur Realisierung einfacher Bewegungen oder zum Spannen der zu bearbeitenden Bauteile. *Gruppen* kombinieren zusammenwirkende Einheiten zu komplexeren Konstrukten. In Fertigungssystemen sind dies Teilsysteme, wie etwa der Werkzeugwechsler oder das Werkzeugmagazin. Die übergeordnete Hierarchiestufe integriert mehrere Gruppen zu einer *Einrichtung*. Diese Ebene umfasst im Wesentlichen einzelne Maschinen oder Systeme zum Werkzeug- oder Werkstücktransport. Die oberste Ebene hingegen beschreibt das gesamte Produktionssystem und wird als *Anlage* bezeichnet.

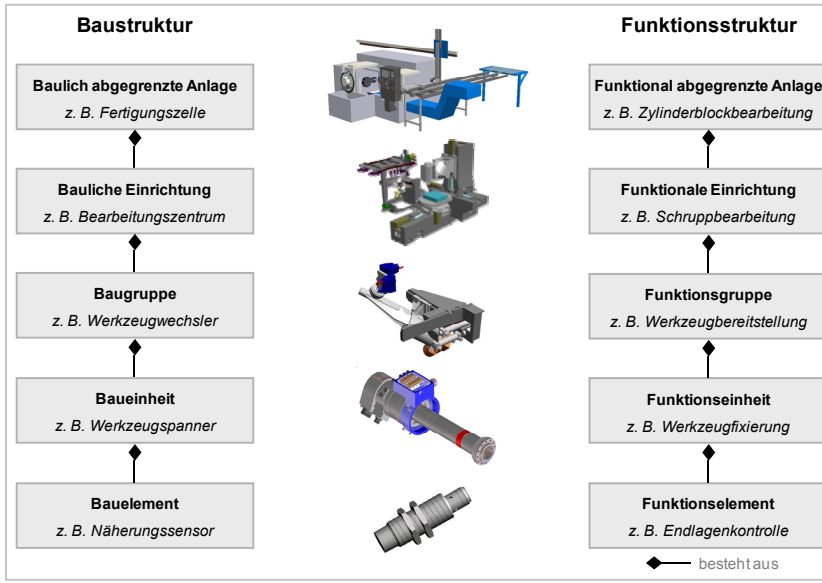


Abbildung 3.2: Strukturierung von Fertigungssystemen

3.2.3 Fertigungssysteme als mechatronische Systeme

Trotz der Dominanz der immanenten mechanischen Funktionalität sind Fertigungssysteme sowie deren untergeordneten Teilsysteme als komplexe mechatronische Produkte zu betrachten (siehe Abbildung 3.3).

Nach *(Lippold 2001)* bestehen diese aus einer Menge von *Sensoren* und *Aktuatoren*, einem *Grundsystem* und einem *IT-System*. Bezogen auf Fertigungssysteme sind Aktuatoren meist als hydraulische, pneumatische oder elektromechanische Einheiten ausgeführt, die auf der Grundlage von Steuersignalen die notwendige Energie zur Bearbeitung bereitstellen oder Bewegungen erzeugen. Diese wirken direkt oder indirekt über ein *mechanisches Trägersystem* auf den *Prozess* (Zerspanprozess, Werkzeug- oder Werkstücktransport, usw.) ein. Durch geeignete Sensoren wird der aktuelle Zustand des Systems erfasst und an ein übergeordnetes IT-System weitergeleitet. Das IT-System vergleicht den Istzustand mit einem gewünschten Sollzustand und leitet entsprechende Stellbefehle für die Aktuatoren ab.

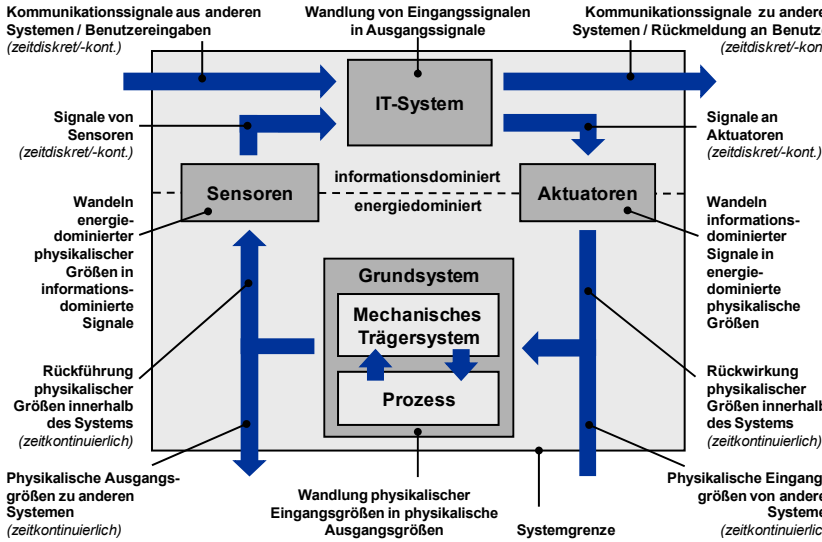


Abbildung 3.3: Fertigungssysteme als mechatronische Systeme (in Anlehnung an Lippold 2001)

Das IT-System automatisierter Fertigungssysteme und Werkzeugmaschinen besteht üblicherweise aus mehreren Subsystemen, die auf unterschiedlichen Technologien basieren und definierte Aufgaben erfüllen, jedoch zur Realisierung der Gesamtfunktion miteinander verknüpft sind. Neben Numerischen Steuerungen, Speicherprogrammierbaren Steuerungen oder Industrie-PCs (IPC) sind dies beispielsweise Bedien- und Beobachtungsgeräte oder Prozessleitsysteme zur Auftragssteuerung. Zur aufgabenbezogenen Klassifizierung wurde von der International Organisation for Standardisation (ISO) ein Ebenenmodell entwickelt, in das sich marktübliche Systeme einordnen lassen.

3.2.4 Steuerungsebenen in der Automatisierungstechnik

Nach *(Pfeifer & Heiler 1987)* werden speziell für Fertigungssysteme fünf funktionale Steuerungsebenen definiert (siehe Abbildung 3.4). Die Planungsebene beinhaltet alle der eigentlichen Fertigung vorgelagerten Aufgaben. Sie umfasst die Bereitstellung von technischen und organisatorischen Vorgaben für die nachfolgende Durchführung, wie beispielsweise die Erstellung der NC-Programme, der Arbeitspläne und der Fertigungszeichnungen *(Peters 2001)*. Die untergeordnete Leitebene realisiert auf der Basis dieser Informationen die Planung und Steuerung zellenübergreifender Vorgänge. Hierzu zählen unter anderem die zeitliche Organisation der Arbeitsaufträge und die Steuerung des zellenübergreifenden Werkzeug- und Materialflusses. Die tatsächliche Aus-

führung eines Bearbeitungsauftrages wird auf der Zellenebene implementiert. Die Funktionen umfassen somit die Steuerung und Überwachung der Aufträge. Die Prozess- und die Feldebene beinhalten die eigentlichen Produktionsmaschinen. Aufgaben dieser Ebenen sind beispielsweise die Steuerung bzw. Regelung und Koordination der Bewegungsachsen, die Umsetzung automatisierter Abläufe sowie die Bereitstellung sicherheitstechnischer Überwachungs- oder Prozesssteuerungs- und Prozessregelungsfunktionen.

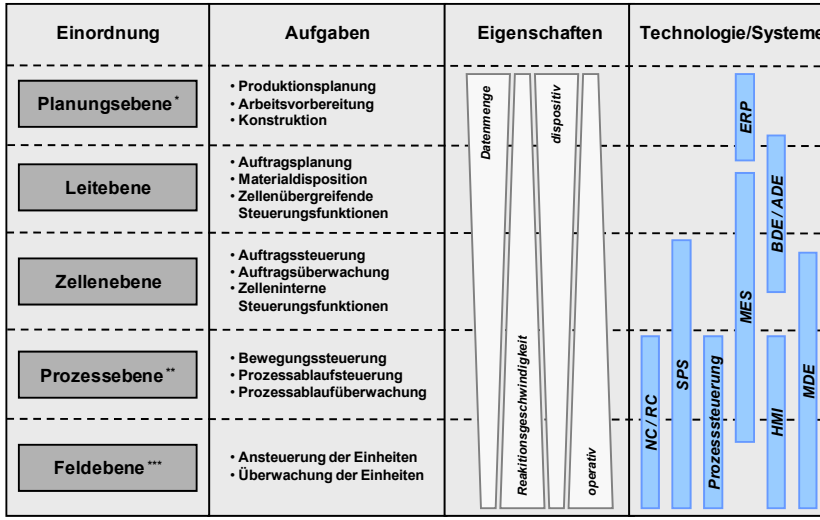


Abbildung 3.4: Steuerungsebenen in Fertigungssystemen

Die Funktionalität auf den höheren Ebenen ist weitestgehend durch dispositive Aufgaben geprägt. Daher müssen entsprechende Systeme auch wesentlich mehr Daten verarbeiten. Die Aufgaben der unteren Ebene hingegen sind durch operative Funktionen gekennzeichnet. Die Menge der zu verarbeitenden Daten ist zwar wesentlich geringer, allerdings werden hohe Anforderungen an die Reaktionsgeschwindigkeit und die Sicherheit bei der Datenverarbeitung gestellt. Entsprechend den unterschiedlichen Anforderungen haben sich für die Umsetzung der dargestellten Funktionen verschiedene technische Lösungen durchgesetzt. Während auf der Planungs-, Leit- und Zellenebene weitestgehend PC-basierte Systeme verwendet werden, sind auf der Prozess- und Feldebene proprietäre Systeme im Einsatz. Beispiele sind Echtzeitsysteme, wie Numerische Steuerungen, Robotersteuerungen oder Prozessregelsysteme, die eine Datenverarbeitung im Bereich unterhalb weniger Millisekunden garantieren müssen. Es ist jedoch ein Trend zur vertikalen Integration der einzelnen Ebenen zu beobachten. Aufgrund der rasanten Leistungsentwicklung von Mikroprozessoren übernehmen innova-

tive Systeme vermehrt Funktionen über- oder untergeordneter Hierarchiestufen und können somit nicht mehr eindeutig einer bestimmten Ebene zugeordnet werden. Als Beispiele seien intelligente Aktoren genannt, die bereits Funktionen zur Selbstdiagnose oder zur Regelung der Bewegungen integrieren, sowie Speicherprogrammierbare Steuerungen, die um Funktionen zur direkten Ansteuerung und Regelung von Servomotoren erweitert wurden.

3.2.5 Steuerungsarchitekturen, Bussysteme und Komponenten

Zur Erfüllung der beschriebenen Funktionen der Werkzeugmaschinen und Fertigungssysteme wurden unterschiedliche, grundlegende Architekturkonzepte entwickelt, die sich am strukturellen Aufbau der Systeme, den Anforderungen einzelner Teilsysteme sowie der Komplexität des Gesamtsystems orientieren. Bezüglich der für die Arbeit relevanten unteren drei Steuerungsebenen (siehe Abschnitt 3.3) erfolgt eine Unterscheidung in zentrale Steuerungen, zentrale Steuerungen mit dezentraler Ein- und Ausgabe, hierarchisch organisierte Steuerungsverbünde, hierarchisch organisierte Verbünde mit direkter Synchronisation und kooperative Steuerungssysteme *(Meier 2001)*. Abschnitt 11.3.1 beinhaltet eine kompakte Beschreibung und Gegenüberstellung dieser Architekturen. Bezüglich der Darstellung typischer Systemstrukturen auf der Planungs-, Leit- und teilweise der Zellenebene wird auf *(Peters 2001)* verwiesen.

Die Verknüpfung der einzelnen IT-Komponenten eines Fertigungssystems wird meist mittels eines oder mehrerer Bussysteme umgesetzt. Auf höheren Ebenen stellt Ethernet aufgrund der hohen Bandbreite und der geringeren Anforderungen bezüglich der Reaktionsgeschwindigkeit mittlerweile einen De-facto-Standard dar. Zur Verbindung der Komponenten auf der Feldebene sowie zur Anbindung an die Prozessebene und teilweise die Zellenebene werden Sensor-/Aktor-Bussysteme, Antriebsbusse, Feldbusse oder herstellerspezifische Systeme eingesetzt. Beispiele für in Fertigungssystemen verbreitete Implementierungen sind AS-I[1], Profibus und Sercos. Neuere Ansätze auf der Basis von Industrial-Ethernet (z. B. Profinet) bieten hierzu erfolgsversprechende Alternativen. Obwohl die Problematik der mangelnden Echtzeitfähigkeit mittlerweile gelöst ist, haben sich diese, nicht zuletzt aufgrund der Fülle der proprietären Implementierungen, noch nicht durchgesetzt. Die Auswahl der Bussysteme aus der Vielzahl der am Markt verfügbaren Lösungen orientiert sich an den Anforderungen bezüglich der Menge der zu übertragenden Daten, der notwendigen Reaktionsgeschwindigkeit und sicherheitsrelevanten Aspekten. Weck und Brecher *(Weck & Brecher 2006)* geben einen Überblick zum Thema. Für detaillierte Informationen wird auf *(Schnell & Wiedemann 2006; Gevatter & Grünhaupt 2006)* verwiesen.

[1] AS-I = Aktuator Sensor-Interface; Profibus = Process Field Bus; Sercos = Serial Realtime Communication System.

Aufgrund der Vielzahl der verfügbaren und in Fertigungssystemen eingesetzten automatisierungstechnischen Komponenten kann im Rahmen der Arbeit diesbezüglich kein detaillierter Überblick gegeben werden. Abbildung 3.5 zeigt die Vielfältigkeit exemplarisch anhand eines Ausschnittes der IT-Systemarchitektur einer flexiblen Fertigungszelle.

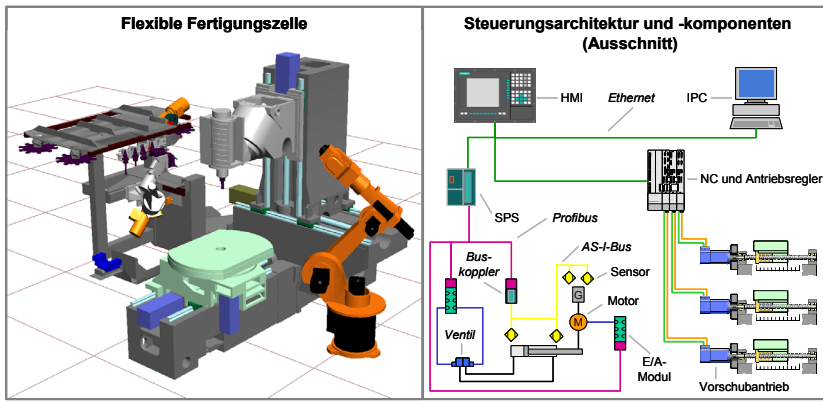


Abbildung 3.5: Struktur und Automatisierungskomponenten einer Fertigungszelle

Eingeschränkt wird die Auswahl, insbesondere der dezentral in den Anlagen verbauten Feldgeräte (Sensoren, Aktoren, Steuerglieder), durch herstellerspezifische Standards oder Kundenvorgaben. Herstellerseitig ist dies in der Minimierung des Verwaltungsaufwandes in Bezug auf die Beschaffung und die Lagerhaltung sowie durch den hohen Einarbeitungs- und Schulungsaufwand begründet. Das Interesse des Kunden besteht darin, die Anzahl der notwendigen Ersatzteile zu reduzieren und unabhängig vom Hersteller eine schnelle und einfache Wartung seiner Maschinen und Anlagen zu ermöglichen.

In Bezug auf die wichtigsten eingesetzten Steuerungsarten werden Speicherprogrammierbare Steuerungen, Numerische Steuerungen und Robotersteuerungen unterschieden. Aufgrund der Relevanz für die vorliegende Arbeit werden im Folgenden die Aufgaben, der Aufbau und die Funktionsweise Speicherprogrammierbarer Steuerungen näher erläutert. Bezüglich der weiteren Steuerungsarten wird auf den Abschnitt 11.3.2 im Anhang der Arbeit verwiesen.

3.2.6 Speicherprogrammierbare Steuerungen

3.2.6.1 Aufgaben der SPS

Speicherprogrammierbare Steuerungen sind nach *(Schweiker 2003, S. 18)* unabhängig von der Fertigungstechnologie Teil eines Fertigungssystems. Sie werden zur Erfüllung

von Aufgaben diverser tieferer Hierarchieebenen eingesetzt. Auf der Feldebene übernehmen sie die Steuerung und Überwachung der einzelnen Funktionseinheiten. Dies beinhaltet die Implementierung der notwendigen logischen Verknüpfungen und Verriegelungen, die Umsetzung von Laufzeit- und Paarfehlerüberwachungen sowie die Erfassung des aktuellen Zustandes. Ebenso realisieren sie Bedienfunktionen, wie beispielsweise das Einschalten der Antriebe oder der Kühlmittelversorgung. Auch die Umsetzung sicherheitsrelevanter Funktionen und die Reaktion auf Fehler bzw. Störungen (z. B. Not-Aus) im System ist Aufgabe der SPS. Auf der Prozessebene werden Maschinenabläufe, wie beispielsweise der Werkzeugwechsel, ausgeführt. Teilweise werden auch die Erfassung, Aufbereitung und Weiterleitung von Prozess- und Maschinendaten und die Verwaltung der Werkzeuge[2] durch die Speicherprogrammierbare Steuerung vorgenommen. Darüber hinaus gehören eine Reihe von Kommunikationsaufgaben zum Funktionsumfang. Auf der Prozessebene ist dies beispielsweise die Synchronisation mit anderen Steuerungen (NC-Steuerung, IPC, usw.). Auf der Zellenebene gehört hierzu unter anderem die Kommunikation mit über- oder untergeordneten Systemen (z. B. weitere SPS, Zellen- oder Leitrechner).

3.2.6.2 Aufbau und Funktionsweise der SPS

Eine Speicherprogrammierbare Steuerung ist durch ihre modulare Struktur gekennzeichnet. Sie besteht aus einer Energieversorgung, einer zentralen Baugruppe mit Speicher, Prozessor und Betriebssystem sowie einer Reihe von Zusatzbaugruppen, die entsprechend den Anforderungen und dem Aufbau des zu steuernden Systems konfiguriert und über ein internes Bussystem miteinander verbunden werden. Beispiele sind Busanschaltungen, Ein- und Ausgabebaugruppen zum direkten Anschluss von Sensoren und Aktuatoren sowie Zähler- oder Reglerbaugruppen. Moderne Ausführungen der zentralen Baugruppe umfassen in der Regel einen leistungsfähigen Prozessor zur Bit- und Wortverarbeitung, einen nichtflüchtigen Programmspeicher, einen erweiterbaren Zusatzspeicher sowie eine Rechnerschnittstelle. Üblicherweise verfügen auch Numerische Steuerungen über eine integrierte SPS mit Busanschaltung, weshalb bei weniger komplexen Maschinen und Anlagen auf zusätzliche Steuerungen verzichtet werden kann. Bei kostengünstigen Systemen wird die SPS-Funktionalität durch einen Industrie-PC mit integrierter Busanschaltung emuliert (Soft-SPS).

Die Ausführung der Steuerungsanweisungen läuft normalerweise zyklisch ab. Nach dem Einlesen aller Eingangssignale in den Speicher (Eingangsabbild) erfolgt die Bearbeitung der Programme. Dabei werden die internen Daten und die Ausgänge entsprechend der Programmlogik geändert und im Speicher abgelegt (Ausgangsabbild). Nach dem Programmende werden die Ausgangssignale aktualisiert und der Zyklus neu gestartet. Hochwertige Steuerungen verfügen zusätzlich über Möglichkeiten zur ereig-

[2] Bei komplexeren Systemen wird diese Aufgabe durch einen IPC oder die numerische Steuerung realisiert.

nisgesteuerten Ausführung bestimmter Funktionen. Die zyklische Bearbeitung wird zeitgesteuert oder bei einer definierten Signaländerung (z. B. Not-Aus) unterbrochen, um die notwendigen Steueranweisungen (z. B. schnelles Stillsetzen der Antriebe) auszuführen. Anschließend wird die zyklische Bearbeitung an der Unterbrechungsstelle wieder aufgenommen.

3.3 Eingrenzung des Betrachtungsbereiches

Die vorliegende Arbeit ist darauf ausgerichtet, die Entwicklung der Steuerungssoftware von Fertigungssystemen durch den Hersteller zu unterstützen. Die Aufgabenstellung ist dabei durch folgende Randbedingungen charakterisiert:

- Zur Erfüllung der Steuerungsfunktionen werden unterschiedliche Steuerungsarten (siehe Abschnitt 3.2.5) auf verschiedenen Hierarchieebenen (siehe Abschnitt 3.2.4) eingesetzt.
- Die Steuerungsaufgaben der oberen Hierarchiestufen sind von der Individualität einzelner Teilsysteme weitestgehend unbeeinflusst. In der Konsequenz werden hierfür meist Standardsoftwaresysteme eingesetzt, die entsprechend der Aufgabenstellung konfiguriert werden. Die Einbindung neuer Maschinen und Anlagen in diese Systeme ist in der Regel nicht Aufgabe des Herstellers.
- Die Funktionen der Numerischen Steuerungen werden größtenteils durch den Steuerungshersteller implementiert. Hauptaufgabe des Maschinenherstellers ist deren Konfiguration (Anzahl der Achsen, Softwareendschalter, Reglereinstellungen, Filterparametrierung, usw.) durch die Festlegung entsprechender Parameter sowie die Anbindung an weitere Steuerungssysteme über eine vordefinierte Schnittstelle.
- Speicherprogrammierbare Steuerungen werden zur Erfüllung der Aufgaben der tieferen Hierarchieebenen eingesetzt (siehe Abschnitt 3.2.6.1). Aufgrund der Individualität sowohl der Funktionalität, der eingesetzten Komponenten als auch der Steuerungsarchitekturen sind entsprechende Programme nicht grundsätzlich generalisierbar. Deshalb und aufgrund der Notwendigkeit eingehender Kenntnisse des gesteuerten Systems ist deren Entwicklung im Wesentlichen Aufgabe des Herstellers der Fertigungssysteme.

Aufgrund der beschriebenen Rahmenbedingungen fokussiert die Arbeit auf die *Entwicklung der Software für Speicherprogrammierbare Steuerungen* automatisierter Fertigungssysteme. Die gewonnenen Ergebnisse bzw. vorgeschlagenen Ansätze sind prinzipiell auch auf andere Bereiche des Maschinenbaus übertragbar, sofern dort zum Werkzeugmaschinenbau vergleichbare Anforderungen vorliegen und sich die geforderten Steuerungsfunktionen entsprechen. Die Übertragbarkeit wird aber im Rahmen der vorliegenden Arbeit nicht untersucht. Des Weiteren werden lediglich die *entwicklungsintensiven Phasen der Auftragsabwicklung* (siehe Abschnitt 3.4) betrachtet. Vorhergehende und nachfolgende Entwicklungsabschnitte und domänenspezifische Aufgaben außerhalb der Softwareentwicklung (z. B. Konzeption neuer Lösungen durch

die mechanische Konstruktion) werden nur am Rande und soweit notwendig diskutiert.

3.4 Entwicklung von Fertigungssystemen

3.4.1 Allgemeines

Die traditionelle Entwicklung von Fertigungssystemen und von deren Komponenten ist durch eine überwiegend sequentielle Vorgehensweise, insbesondere der beteiligten Fachabteilungen gekennzeichnet *(Simon 2006, S. 53; Baudisch 2003, S. 4)*. Dabei werden folgende drei, im Rahmen der Arbeit relevante, Phasen unterschieden:

- Produktplanung und Konstruktion der Maschinen und Anlagen
- Konzeption und Implementierung der Steuerungssoftware
- Steuerungstest, Inbetriebnahme und Wartung

3.4.2 Produktplanung und Konstruktion der Maschinen und Anlagen

Aus methodischer Sicht kann der iterative Prozess der mechanischen Konstruktion *(VDI 2221)* in die Phasen *Anforderungsklärung*, *Konzeption (VDI 2222)*, *Entwurf (VDI 2223)* und *Ausarbeitung* gegliedert werden[3]. Ausgangspunkt für die Entwicklung ist eine ausführliche Klärung der Aufgabenstellung und der Anforderungen in Form eines Pflichtenheftes. Die Basis hierfür bildet bei Serienprodukten eine gründliche Marktanalyse, die im Rahmen der Produktplanung durchgeführt wird. Bei Kleinserien- oder Einzelprodukten stehen dagegen detaillierte kundenspezifische Anforderungen im Vordergrund. Teilweise werden die Bearbeitungsprozesse und die zu fertigenden Bauteile exakt spezifiziert, wobei klare Vorgaben bezüglich der erreichbaren Fertigungsgenauigkeiten, der maximalen Taktzeiten und weiterer Randbedingungen einzuhalten sind. Die Konzeptionsphase umfasst in Bezug auf die Entwicklung von Fertigungssystemen mehrere Schritte. Bei komplexen, verketteten Systemen erfolgt zunächst die Festlegung der technologischen Verfahren und der Bearbeitungsabläufe sowie des Anlagendesigns. Anschließend werden die wesentlichen Maschinenfunktionen definiert und strukturiert (Funktionsstruktur). Darauf aufbauend findet die Verteilung der erforderlichen Bewegungsachsen auf die unterschiedlichen Maschinenkomponenten, die Festlegung ihrer Lage im Raum sowie die Auswahl der verwendeten Führungs- und Arbeitsprinzipien statt. Die Hauptaufgabe der Konstruktion besteht somit darin, möglichst einfache, fertigungsgerechte und damit kostengünstige Einzelkomponenten so anzuordnen und zu kombinieren, dass sich eine kompakte montage- und transportgerechte Gesamtstruktur ergibt, die die im Pflichtenheft festgelegten Leistungsanforderungen erfüllt. Ergebnis der Konzeptionsphase ist ein grobgranulares Modell der Ma-

[3] Die Zuordnung einzelner Aufgaben zu den Phasen sowie deren Bezeichnung ist je nach Branche unterschiedlich.

schine bzw. Anlage (Konzeptmodell), das die notwendigen Funktionen in Form prinzipieller Lösungen implementiert und zu einer Gesamtstruktur (Wirkstruktur) integriert. Die Entwurfsphase umfasst die Auswahl, Dimensionierung und räumliche Festlegung der gestaltbestimmenden Komponenten, wie beispielsweise der Führungen, Antriebe und Gestelle, sowie vermehrt die Integration extern entwickelter und produzierter Teilsysteme. Letzteres ist in der Notwendigkeit zur Reduktion der Fertigungstiefe durch den steigenden Kostendruck *(Simon 2006, S. 24)* begründet. Die Konstrukteurinnen und Konstrukteure entwickeln sich folglich zunehmend von Systemgestaltern zu Systemkonfiguratoren *(Schneider 2000)*. Die Ausarbeitung umfasst die Festlegung geometrischer Details sowie die Ergänzung der Modelle um Angaben bezüglich der Herstellung und Verwendung in Form von Fertigungszeichnungen, Montageplänen, Stücklisten und Vorschriften. Aufgrund des hohen Zeit- und Kostendrucks ist der Bau von Prototypen nur mehr eingeschränkt möglich. Gerade im Sondermaschinenbau, aber verstärkt auch bei der Entwicklung von Serienmaschinen, wird das entwickelte System direkt an den Kunden ausgeliefert. Dem entsprechend ist der Absicherung der Produkteigenschaften mittels virtueller Prototypen im Rahmen des Entwurfs eine steigende Bedeutung beizumessen.

Die Art der Konstruktionsaufgabe definiert sich entsprechend den Anforderungen sowie dem bestehenden Produktspektrum des Herstellers. Primäres Ziel ist die Konfiguration der Maschinen und Anlagen aus existierenden Teilsystemen und Zulieferkomponenten. Sind die Anforderungen durch diese nicht erfüllbar, so sind Anpasskonstruktionen oder Variantenkonstruktionen notwendig. Erstere sind nach *(Pahl u. a. 2005, S. 4)* dadurch gekennzeichnet, dass bekannte und bewährte Lösungsprinzipien eingesetzt, jedoch einzelne Teile und Baugruppen vollständig neu entwickelt werden. Systematische Variantenkonstruktionen hingegen zeichnen sich durch die Konfiguration des Gesamtsystems aus Baukästen oder die Entwicklung von Baureihen aus. Hierbei wird die Funktionsstruktur der Maschine meist nicht verändert. Modifikationen ergeben sich aber aus geometrischen Umgestaltungen sowie der Anpassung bestehender Baugruppen an neue Technologien. Anpass- und Variantenkonstruktionen stellen die im Werkzeugmaschinenbau am weitesten verbreitete Konstruktionsart dar. Wahl *(Wahl 2000)* verweist diesbezüglich auf Untersuchungen, wonach diese 85 Prozent der Entwicklungszeit im Rahmen der mechanischen Konstruktion beanspruchen. Als vollständige Neukonstruktionen sind lediglich zehn Prozent der Entwicklungen zu betrachten.

3.4.3 Konzeption und Implementierung der Steuerungssoftware

Die Steuerungsentwicklung und Elektrokonstruktion beginnt üblicherweise erst nach dem Abschluss der wesentlichen Arbeiten am Entwurf des mechanischen Aufbaus *(Schaich 2001)*. In Anlehnung an die Vorgehensweise im Rahmen der mechanischen Konstruktion wurden hierfür adaptierte Entwicklungsprozesse erarbeitet *(z. B.*

VDI/VDE 3683; Pickhardt 2000, S. 173-187). Nach *(Weck & Brecher 2006, S. 132)* werden die Entwicklungsaufgaben in die Phasen Spezifikation, Programmentwurf, Implementierung, Test sowie Wartung und Pflege eingeordnet.

Eine detaillierte Spezifikation der Steuerungsaufgaben wird in der industriellen Praxis in der Regel nicht oder nur für ausgewählte Funktionen durchgeführt. Die wichtigsten Informationsquellen sind Entwurfsunterlagen der mechanischen Konstruktion, wie beispielsweise Prinzipskizzen und Anlagenlayouts sowie tabellarische Auflistungen der relevanten Komponenten und softwarespezifische Funktionen aus der Auftragsbeschreibung. Darüber hinaus wird ein nicht unwesentlicher Anteil der Aufgaben nicht oder nur unvollständig dokumentiert. Die Übergabe der Informationen beruht auf mündlichen Absprachen zwischen der mechanischen Konstruktion und der Softwareentwicklung. Teilweise wird das notwendige Wissen aufgrund der Erfahrung mit ähnlichen Systemen auch implizit als bekannt vorausgesetzt. Auf der Basis einer Analyse der erhaltenen Informationen sowie herstellerspezifischer Standards wird die prinzipielle, meist funktional gegliederte Programmstruktur konzipiert. Parallel dazu beginnt die Festlegung der Steuerungshardware und die Elektrokonstruktion (Schaltschrankaufbau, Installationspläne). Anhand der Programmstruktur wird überprüft, inwieweit bestimmte Funktionen aus anderen Projekten zur Aufgabenerfüllung eingesetzt werden können. Ist dies nicht der Fall, so werden die zu entwickelnden Programmmodule vollständig neu entworfen und implementiert oder angepasst.

3.4.4 Steuerungstest, Inbetriebnahme und Wartung

Der Test der Steuerungssoftware geschieht soweit möglich am Entwicklungsarbeitsplatz. Wurde dieser früher durch sogenannte Schalterkästen[4] und die Zuhilfenahme der Steuerungshardware realisiert, so werden heute vorwiegend Steuerungsemulatoren, die in eine entsprechende Entwicklungsumgebung für die Software integriert sind, eingesetzt. Zur Nachbildung des gesteuerten Systems werden Eingangssignale per Hand geändert und die Reaktion der Softwarebausteine beobachtet. Da diese Vorgehensweise das Zeitverhalten nur unzureichend nachbilden kann und viel Abstraktionsvermögen voraussetzt, können komplexere Steuerungsfunktionalitäten auf diese Weise nicht mehr geprüft werden. Der endgültige Softwaretest wird an den fertig aufgebauten Maschinen und Anlagen durchgeführt und geht fließend in die Inbetriebnahme über. Diese stellt eine der entscheidenden Phasen bei der Realisierung automatisierter Anlagen dar, da erst hier das ordnungsgemäße Zusammenwirken der mechanischen, elektromechanischen, hydraulischen und pneumatischen Komponenten mit der Steuerungssoftware überprüft werden kann. Dabei wird zweckmäßigerweise ein Bottom-up-Ansatz angewendet. In einem ersten Schritt werden die Komponenten der Feldebene

[4] Ein Schalterkasten ist ein elektrisches Schaltpult mit Stellelementen und Anzeigen, das dazu dient, das Verhalten des gesteuerten Systems gegenüber der Steuerung zu simulieren und deren Reaktion zu beobachten.

(Funktionseinheiten) auf ihre korrekte Funktion und mögliche Fehlerfolgen hin überprüft. Zunächst steht die Verifikation der Programmlogik im Vordergrund. Anschließend wird in Abstimmung mit dem gesteuerten System die Einstellung sämtlicher Parameter (z. B. Werte für Laufzeitüberwachungen, Sollpositionen oder Toleranzschwellen) vorgenommen. Darauf aufbauend erfolgt die Prüfung übergeordneter Abläufe, der Funktionen des Automatikbetriebes und der Kommunikationsaufgaben sowie die Integration von Teilsystemen externer Zulieferer.

Während der Test und die Inbetriebnahme der gewünschten Soll-Funktionen meist weniger problematisch sind, stellt eine eingehende Überprüfung der Routinen zur Fehlerbehandlung eine Herausforderung dar. Dies ist darauf zurückzuführen, dass durch ungewünschte Reaktionen der Steuerung eine Gefährdung für die Maschinen oder gar das Inbetriebnahmepersonal riskiert wird. Teilweise können bestimmte Funktionen daher nicht getestet werden. Vor dem Hintergrund potentieller Folgekosten, der herausragenden Bedeutung der Zuverlässigkeit und Verfügbarkeit der entwickelten Systeme für den Unternehmenserfolg *(BMBF 2002, S. 29)* und der Tatsache, dass ein großer Teil der Steuerungssoftware für Fehlererkennungs- und Diagnoseroutinen verantwortlich zeichnet, ist dies als problematisch zu betrachten.

Eingeschränkt werden die Möglichkeiten zum Softwaretest und zur Inbetriebnahme auch durch organisatorische und methodische Randbedingungen. Aufgrund des hohen Zeitdruckes im Rahmen der Auftragsabwicklung und der hohen Kapitalbindung ist die Verfügbarkeit und der aufgabengerechte Montagezustand der Anlagen bzw. ihrer Teilsysteme für den Vorgang der Inbetriebnahme nicht sichergestellt. Softwaretest und Inbetriebnahme können daher meist nur zum Teil durchgeführt werden und finden folglich oft erst beim Kunden statt. Ebenso ist eine systematische Überprüfung aller Funktionen nur eingeschränkt möglich. Neben den genannten Randbedingungen ist dies im Fehlen entsprechender Testspezifikationen begründet.

Die Wartung und Pflege der Steuerungssoftware von Fertigungssystemen umfasst vor allem die Beseitigung nicht erkannter Fehler, Funktionserweiterungen und Funktionsveränderungen, die Portierung auf andere Zielsysteme sowie Maßnahmen zur Steigerung der Effizienz. Entscheidend für eine reibungslose Umsetzung der genannten Aktivitäten ist eine transparente und nachvollziehbare Dokumentation des Programms, aber auch des Softwareentwurfs *(Kohring 1993)*. Die Basis für eine systematische Wartung und Pflege ist daher bereits während der Projektierung der Maschinen und Anlagen zu schaffen. In der Praxis wird diese Forderung jedoch aufgrund des zusätzlichen Aufwandes nur eingeschränkt umgesetzt.

3.5 Schlussfolgerungen

3.5.1 Übersicht

Eine Analyse der Entwicklung der Steuerungssoftware von Fertigungssystemen und aktuelle Studien zu zukünftigen Tendenzen im Umfeld der Automatisierungstechnik zeigen eine Reihe organisatorisch-methodischer und technologischer Defizite sowie Herausforderungen. Diese werden im Folgenden dargestellt.

3.5.2 Organisatorische und methodische Defizite

Unzulänglichkeiten bei der Aufgabenklärung und der Angebotsspezifikation
Im Rahmen der Angebotsphase werden die beteiligten Fachabteilungen nur selten mit einbezogen, sodass die Klärung der Aufgabenstellung nur in einem unzureichenden Umfang stattfindet. Gerade bei technologischen Innovationen oder Neuentwicklungen ist dem Kunden die notwendige Funktionalität nicht immer vollständig bewusst. Technische Risiken sind in der Folge nicht erkennbar, wodurch erhebliche Defizite bezüglich der Planung der Auftragsabwicklung und der Kostenkalkulation entstehen. Im besonderen Maße ist die Softwareentwicklung davon betroffen, da in den meisten Kundengesprächen der mechanische Aufbau der Maschinen sowie die Fertigungstechnologie im Vordergrund stehen.

Organisatorische Trennung von mechanischer Konstruktion und Softwareentwicklung
Aufgrund des mechatronischen Charakters automatisierter Fertigungssysteme erfordert die Entwicklung eine frühe und präzise Abstimmung der beteiligten Fachdisziplinen. Da bei den meisten Herstellern von Fertigungssystemen eine Abteilungsstruktur mit klarer räumlicher und/oder organisatorischer Trennung nach wie vor die vorherrschende Organisationsform darstellt, wird der Informationsaustausch erschwert. Teilweise erfüllen Mitarbeiterinnen bzw. Mitarbeiter eine übergeordnete Schnittstellenfunktion, eine derartige Lösung ist aber in der Praxis nicht Standard.

Fehlende interdisziplinäre, mechatronische Denkweise der Entwicklungsdomänen
Die an der Entwicklung beteiligten Fachdisziplinen verfügen über eine völlig unterschiedliche Ausbildung, eine dem Aufgabengebiet angepasste Denkweise und eine fachspezifische Semantik der Begrifflichkeiten. Mitarbeiterinnen und Mitarbeiter der mechanischen Konstruktionsabteilung haben im Allgemeinen eine gestaltorientierte Auffassung der Entwicklungsaufgabe. Ihr Fokus liegt darauf, die Form und Anordnung der einzelnen Bauteile in geeigneter Weise festzulegen und diese so zu dimensionieren, dass die gestellten Leistungsanforderungen erfüllt werden können. Die Softwareentwicklerinnen bzw. Softwareentwickler haben hingegen eine weitestgehend

funktionsorientierte Sicht. Ihre Aufgabe ist es, den Zustand des gesteuerten Systems zu erfassen, die Ausführung bestimmter Funktionen zu realisieren und auf Fehler oder Störungen im System in geeigneter Weise zu reagieren. Dies ist auch die Hauptursache dafür, dass Programme für Speicherprogrammierbare Steuerungen in der Praxis meist funktional gegliedert werden. Die Diskrepanz zwischen der gestaltorientierten Strukturierung der Fertigungssysteme und dem funktionsorientierten Aufbau der Steuerungsprogramme hat zur Folge, dass eine Wiederverwendung bereits erstellter Softwarekomponenten in der Regel nur bedingt möglich ist oder einen hohen Anpassungsaufwand erfordert.

Unzureichende Spezifikation der Entwicklungsaufgabe
Eine hinreichende Spezifikation der Aufgabenstellung für die Softwareentwicklung ist in der Praxis nicht üblich *(Beckering 2003)*. Dies ist nach wie vor so, obwohl die meisten Softwarefehler nicht erst bei der Programmerstellung, sondern bereits während der Spezifikation der Steuerungsaufgabe zustande kommen *(Weck & Brecher 2006, S. 132)*. Problematisch dabei ist, dass diese Unzulänglichkeiten erst im Rahmen der Inbetriebnahme der Maschinen und Anlagen entdeckt werden. In der Folge trägt dies mit dazu bei, dass die Inbetriebnahme der Steuerungstechnik für ca. neunzig Prozent der gesamten Inbetriebnahmezeit eines Fertigungssystems verantwortlich zeichnet *(Schelberg 1994)*. In manchen Fällen können die genannten Defizite sogar dazu führen, dass die mechanische Konstruktion noch während der Inbetriebnahme umfassenden Änderungen unterzogen werden muss.

Sequentielle Entwicklung statt Concurrent Engineering
Die Softwareentwicklung wird typischerweise erst in die Entwicklung eingebunden, nachdem die mechanische Konstruktion im Wesentlichen abgeschlossen ist. Diese sequentielle Vorgehensweise führt zu teiloptimierten Lösungen, die im Rahmen kosten- und zeitintensiver Iterationen erarbeitet werden müssen. Gerade bei Produkten mit mechatronischem Charakter ist der Ansatz des Concurrent Engineering als viel versprechende Alternative zum bestehenden Entwicklungsvorgehen zu betrachten. Ziel ist es, die Entwicklungszeit bei gleichzeitiger Steigerung der Qualität drastisch zu verkürzen, indem Arbeitsschritte parallelisiert, die interdisziplinäre Zusammenarbeit bereits in frühen Entwicklungsphasen forciert und die Durchgängigkeit der Informationsflüsse sichergestellt wird.

3.5.3 Technologische Defizite

Einsatz domänenspezifischer Beschreibungsmittel
Soweit eine schriftliche Spezifikation der Aufgabenstellung überhaupt durchgeführt wird (siehe Abschnitt 3.4.3), geschieht diese unter Zuhilfenahme domänenspezifischer Beschreibungsmittel, vorwiegend aus dem Umfeld der konstruktiven Entwicklung. Beispiele sind Prinzipskizzen, Fertigungszeichnungen und Stücklisten. Diese Doku-

mente werden noch durch Funktionsdiagramme nach *(VDI 3260)* ergänzt, um Abläufe zu spezifizieren. Erfolgt keine direkte Implementierung der Funktionen ohne formale Zwischenschritte, so wird der Softwareentwurf mit Beschreibungsmitteln aus der Automatisierungstechnik realisiert. Beispiele hierfür sind endliche Automaten (siehe Abschnitt 11.2.2) oder Funktionspläne nach *(DIN EN 60848)*. Das unterschiedliche Systemverständnis der beteiligten Fachbereiche, gepaart mit den domänenspezifischen Beschreibungsmitteln und der Komplexität der Aufgabenstellung, erschwert die Abstimmung der Entwicklungsaufgabe und führt zu Missverständnissen sowie Konzept- und Softwarefehlern *(Heverhagen 2003, S. 17; Bock & Zühlke 2006)*. Notwendig ist daher eine Integration der fachspezifischen Sichtweisen zu einer gemeinsamen, leicht verständlichen Spezifikation der Entwicklungsaufgabe *(Siegler 1998, S. 2-2; Möhringer 2004, S. 22)*.

Integration von Simulationsmethoden in den Entwicklungsprozess
Der Nutzen des Einsatzes von Simulationswerkzeugen bei der Entwicklung von Fertigungssystemen ist unbestritten. Nur durch eine Absicherung der Produkteigenschaften, bereits in frühen Phasen des Entwicklungsprozesses, ist eine zielgerichtete, kundenorientierte und wirtschaftliche Realisierung möglich. Innovative Modellierungstechniken und Simulationsmethoden stellen hierzu einen viel versprechenden Ansatz dar. In einigen Domänen des Werkzeugmaschinenbaus werden entsprechende Vorgehensweisen und Softwarewerkzeuge bereits eingesetzt. So ist beispielsweise die FEM (Finite-Elemente-Methode) zur Optimierung des dynamischen und teilweise des thermoelastischen Verhaltens der Maschinenstruktur bereits im praktischen Einsatz. Im Bereich der Softwareentwicklung hingegen ist die Anwendung von Simulationswerkzeugen noch weitestgehend auf das universitäre Umfeld beschränkt *(Bender 1999; Zäh u. a. 2003, Danzer u. a. 2006)*. Obwohl der Nutzen klar aufgezeigt werden konnte *(Stetter 2005; Wünsch 2006)*, wird ein industrieller Einsatz von den Werkzeugmaschinenherstellern unter Bezugnahme auf den hohen Aufwand zur Modellerstellung als kritisch betrachtet.

Durchgängigkeit und Wiederverwendung von Entwicklungsdaten
Die Unterstützung der Entwicklung durch innovative Rechnerwerkzeuge ist mittlerweile in der industriellen Praxis weit verbreitet. Stellen diese offene Schnittstellen und Datenformate bereit, so ergibt sich die Möglichkeit zur vertikalen Integration der Systeme und damit zum Aufbau effizienter Prozessketten. Durch die Weiterverarbeitung bestehender Entwicklungsinformationen werden wiederkehrende Engineeringaufwände vermieden und potentielle Fehlerquellen eliminiert *(Anderl & Trippner 2000)*. Eine zusätzliche horizontale Integration der Systeme (Datenintegration) unterschiedlicher Entwicklungsdomänen ist von strategischer Bedeutung, jedoch bislang noch nicht Stand der Technik. Innovative Ansätze werden beispielsweise in *(Zäh & Lercher 2004)* und *(Gräb 2001)* vorgestellt.

3.5.4 Trends in der Produktions- und Automatisierungstechnik

Bezüglich der zukünftigen Entwicklungen im Umfeld der Produktions- und Automatisierungstechnik wurden in Zusammenarbeit von Industrieunternehmen, Verbänden und universitären Einrichtungen eine Reihe von Studien und Untersuchungen durchgeführt *(BMBF 2002; GMA 2003; BDI u. a. 2005; BMBF 2005; ZVEI 2006)*. Deren Ergebnisse beziehen sich sowohl auf strategische Aufgaben des Managements wie auch auf konkrete technologische Entwicklungen. Die für den Kontext der Arbeit relevanten Tendenzen werden im Folgenden dargestellt.

Steigende Bedeutung der Softwareentwicklung
Obwohl die Softwareentwicklung bereits heute einen hohen Anteil an der Wertschöpfung hat, wird sich dieser zukünftig noch weiter erhöhen. Bislang über Hardware realisierte Funktionen werden aus Kostengründen oder aufgrund steigender Anforderungen durch innovative softwaretechnische Lösungen ersetzt. Damit einher geht die Zunahme der Komplexität der Programme. Gefordert sind folglich Entwicklungsmethoden und Werkzeuge, die nicht nur ein Management der Komplexität erlauben, sondern auch Test- und Diagnoseverfahren zur Absicherung der Funktionalität bereits im Entwicklungsstadium *(ZVEI 2006)*.

Intelligente Feldgeräte
Bei der Entwicklung neuer Hardware- und Softwarelösungen kommt der Aktorik und Sensorik eine steigende Bedeutung zu. Diese werden zunehmend intelligent (z. B. Vorverarbeitung der Messwerte, integrierte Regelung, Selbstdiagnose), wodurch die Steuerungen von Routinetätigkeiten entlastet werden. Mittelfristig ist ein Marktanteil derartiger Systeme von circa 30 Prozent prognostiziert *(Gevatter & Grünhaupt 2006, S. 477)*. Aufgrund der Vielfalt der am Markt verfügbaren Komponenten kommt der Standardisierung von Schnittstellen und Funktionen sowie der Entwicklung und Anwendung von Baukastenkonzepten eine steigende Bedeutung zu.

Relevanz von Modellen
Rechnergestützte Modelle werden in der Zukunft einen höheren Stellenwert einnehmen. Bedingt wird dies durch die Notwendigkeit zur Absicherung der Produkteigenschaften in frühen Entwicklungsphasen und zur Komplexitätsbeherrschung. Neben der Entwicklung selbst erfolgt auch ein verstärkter Einsatz von Modellen im Betrieb sowie in weiteren Phasen des Produktlebenszyklus. Als für Fertigungssysteme relevante Beispiele seien der Einsatz für die Kompensation von Störeinflüssen (Temperatureinfluss, Bearbeitungskräfte) *(Großmann & Wunderlich 2002),* zur Diagnose bzw. Fehlerfrüherkennung *(Maier u. a. 2007)* oder die Nutzung der CAD-Modelle für Schulungs- und Marketingzwecke genannt. Untersuchungen zur zukünftigen Entwicklung mechatronischer Systeme *(BMBF 2005)* unterstreichen insbesondere die Notwendigkeit modellgetriebener Entwicklungsansätze.

3.6 Domänenspezifische Anforderungen an einen modellgetriebenen Ansatz

3.6.1 Übersicht

Eine Analyse der Ausführungen in den vorhergehenden Abschnitten zeigt, dass die aufgeführten Defizite und die sich abzeichnenden Trends in der Automatisierungstechnik die Vorteile und die Notwendigkeit einer modellgetriebenen Entwicklung gleichermaßen unterstreichen. Ein entsprechender Ansatz erfordert aber auch die Berücksichtigung einer Reihe domänenspezifischer Anforderungen, die sich direkt oder indirekt aus den zuvor gegebenen Erläuterungen ableiten lassen.

3.6.2 Methodische Anforderungen

Integration in ein mechatronisches Entwicklungsvorgehen
Bei der Entwicklung eines modellgetriebenen Ansatzes sind die Defizite der vorherrschenden Entwicklungsabläufe zu berücksichtigen. Insbesondere ist eine verbesserte Integration der Softwareentwicklung in die Geschäftsprozesse der mechanischen und elektrischen Konstruktion erforderlich. Folglich bedarf es einer verstärkten Berücksichtigung softwaretechnischer Fragestellungen, sowohl bei der Diskussion der Aufgabenstellung mit den Kunden als auch im Rahmen der Konzeption der Maschinen und ihrer Baugruppen. Ebenso ist die Parallelisierung bislang sequentiell ablaufender Entwicklungsschritte notwendig.

Förderung der fachbereichsübergreifenden Zusammenarbeit
Die Förderung der domänenübergreifenden, teamorientierten Zusammenarbeit bei der Entwicklung legt auch ein Überdenken bisheriger Organisationsformen und der damit verbunden Einschränkungen nahe. Dies ist jedoch nicht hinreichend. Notwendig ist die Gestaltung angepasster Entwicklungsprozesse mit klaren Aufgabenverteilungen zwischen den Domänen, die Definition der fachbereichsübergreifenden Entwicklungsartefakte (Spezifikationen, Dokumentationen, Modelle, usw.) und die Unterstützung der Abstimmungsprozesse durch geeignete Hilfsmittel.

Skalierbarkeit und Flexibilität in Bezug auf die Aufgabenstellung
Die Anwendung der modellgetriebenen Softwareentwicklung soll für unterschiedlich gestaltete Aufgabenstellungen möglich sein. Damit sind sowohl Neuentwicklungen einzelner Baugruppen oder Maschinen als auch Änderungs- und Variantenkonstruktionen zu unterstützen. Voraussetzung hierfür ist die flexible Anpassbarkeit der Engineeringprozesse an die jeweilige Aufgabenstellung.

Effizienz und Effektivität
Vor dem Hintergrund des hohen Kostendrucks in der Werkzeugmaschinenbranche muss die Effizienz der modellgetriebenen Softwareentwicklung gewährleistet werden. Die eingesetzten Hilfsmittel müssen die notwendigen Modellierungsmöglichkeiten bereitstellen, jedoch unabhängig von der Aufgabenstellung auch eine zielgerichtete, effiziente Entwicklung ermöglichen. Diese und die weiteren genannten methodischen Anforderungen bedingen die Berücksichtigung einer Reihe technologischer Fragestellungen, die im Folgenden diskutiert werden.

3.6.3 Technologische Anforderungen

Integrierte Modellbildung von steuerndem und gesteuertem Teilsystem
Der mechatronische Charakter von Fertigungssystemen (siehe Abschnitt 3.2.3) erfordert es, dass bei der Modellbildung nicht nur die Software, sondern auch das gesteuerte System mit berücksichtigt wird. Daher ist neben der Anordnung und Funktionsweise der Sensorik und Aktorik auch die mechanische Grundstruktur in die Modellbildung mit einzubeziehen. In Abhängigkeit von der Steuerungsaufgabe kann ergänzend die Berücksichtigung der auszuführenden Prozesse in abstrahierter Form notwendig sein.

Interdisziplinarität und Praxisnähe der Beschreibungsmittel und Modellkonstrukte
Trotz des domänenspezifisch unterschiedlichen Ausbildungshintergrunds und daraus resultierenden Differenzen im Systemverständnis ist es erforderlich, dass Experten für Fachaufgaben bei der Modellierung aktiv in Entwicklungsteams zusammenarbeiten. Modellbildungstechniken und Beschreibungsmittel müssen deshalb leicht verständlich und praxistauglich sein, zum domänenübergreifenden, gesamtheitlichen Problem- und Systemverständnis beitragen und die Entwicklung innovativer Lösungen durch entstehende Synergieeffekte unterstützen. Gerade im Hinblick auf den weiter steigenden Automatisierungsgrad und die damit einhergehende Komplexität zukünftiger technischer Systeme ist die Ressource „Mensch“ stärker zu berücksichtigen. Dies betrifft den gesamten Lebenszyklus von der Planung bis hin zur Inbetriebnahme, den Betrieb und die Instandhaltung *(Litz & Frey 1999)*. Gleichzeitig ist die Modellbildung aber so zu gestalten, dass die Spezifikation der Entwicklungsaufgabe mit hinreichender Genauigkeit möglich ist. Somit gilt es, den Zielkonflikt zwischen Allgemeingültigkeit, Modellzweck und Anschaulichkeit durch die Auswahl und Integration geeigneter domänenspezifischer, sich gegenseitig ergänzender Beschreibungsmittel zu lösen.

Simulation als Hilfsmittel zur Validierung und Verifikation
Die Integration von Simulationsmethoden zur Validierung und Verifikation der Modelle und damit der Aufgabenstellung ist in verschiedenen Entwicklungsphasen notwendig. Während Simulationsmethoden in der Konzeptionsphase die Erarbeitung, Darstellung und Diskussion prinzipieller Lösungen unterstützen müssen, ist im Rah-

men der Test- und Inbetriebnahmephase die Steuerungssoftware einer eingehenden Prüfung zu unterziehen. Demzufolge sind der Detaillierungsgrad der Modelle, aber auch die eingesetzten Simulationsmethoden mit fortschreitendem Entwicklungsverlauf an die dem Modellzweck zugrunde liegenden Anforderungen anzupassen. Hierbei gilt es auch zu berücksichtigen, dass in frühen Engineeringstadien nur unvollständige und unscharfe Datenbestände zur Verfügung stehen. Um die für die Simulation notwendige Ausführbarkeit der Modelle sicherzustellen, ist es erforderlich, diese soweit zu formalisieren und zu abstrahieren, dass eine eindeutige Interpretation durch entsprechende Simulatoren ermöglicht wird.

Durchgängigkeit der Modellbildung/Konsistenz der Entwicklungsdaten
Im Sinne einer effizienten Entwicklung und zur Gewährleistung konsistenter Entwicklungsdaten ist die Durchgängigkeit der Modellbildung von wesentlicher Bedeutung. Um diese zu ermöglichen, ist es notwendig, Modellierungskonzepte und Beschreibungsmittel auszuwählen, die eine schrittweise Verfeinerung der Modelle zulassen. Ist mit dem sich ändernden Modellzweck eine Änderung der Modellkonstrukte, ihrer Notation, Semantik und Syntax notwendig, so gilt es, die bereits bestehenden Informationsinhalte durch die Festlegung geeigneter Transformationsregeln weiter zu verwenden. Zur Vermeidung von Fehlinterpretationen und zur Entlastung des Entwicklungspersonals von aufwendigen Routinetätigkeiten sollte dieser Vorgang soweit wie möglich automatisiert werden.

Wiederverwendbarkeit und Komplexitätsbeherrschung durch Objektorientierung, Hierarchisierung und Modularisierung
Aufgrund des hohen Anteils an Änderungs- und Variantenkonstruktionen (siehe Abschnitt 3.4.2) ist eine Wiederverwendung bereits bestehender Modellbausteine im Sinne einer effizienten Modellbildung zwingend erforderlich. Voraussetzung ist daher eine modulare Gestaltung der Modelle, die Anwendung von Prinzipien der Objektorientierung sowie eine generische Struktur und spezifische Anpassung von Teilmodellen durch Parametrisierung. Die beschriebenen Möglichkeiten und der Einsatz von Methoden zur hierarchischen Gestaltung der Modelle tragen entscheidend zur Beherrschung der Komplexität heutiger und zukünftiger Fertigungssysteme bei.

Nutzung der Modelle für weitere Aufgaben
Der Nutzen der modellgetriebenen Entwicklung ist nicht nur auf das Softwareengineering beschränkt. Darüber hinaus ergeben sich eine Reihe im Rahmen der Arbeit nur peripher betrachteter Einsatzpotentiale. Beispiele sind die Nutzung der Modelle als Basis für die Erstellung der Elektrodokumentation und der Einsatz für Schulungs- und Wartungszwecke. Voraussetzung hierfür ist jedoch die Nutzung von Standards sowie offener Datenformate im Rahmen der Modellbildung.

Erweiterbarkeit
Um auf zukünftige Entwicklungen reagieren zu können, besteht die Notwendigkeit, den Ansatz und insbesondere die eingesetzten Mittel zur Modellbildung an geänderte Anforderungen anzupassen. Daher ist es erforderlich, diesen Sachverhalt bei der Auswahl geeigneter Beschreibungsmittel zu beachten.

Berücksichtigung von Normen und Standards im Anwendungsbereich
Bestehende Normen sowie hersteller- und kundenspezifische Standards und Vorschriften müssen im Rahmen der Modellbildung beachtet werden können.

3.7 Zusammenfassung

In Kapitel 3 wurde der Aufbau von automatisierten Fertigungssystemen beschrieben und darauf aufbauend der Betrachtungsbereich der Arbeit eingeschränkt. Ebenso wurden die vorherrschenden Rahmenbedingungen bei der Entwicklung von Fertigungssystemen im Detail untersucht. Auf Basis der identifizierten methodischen und technologischen Defizite sowie zukünftigen Trends in der Produktions- und Automatisierungstechnik erfolgte abschließend die Ableitung von domänenspezifischen Anforderungen an einen modellgetriebenen Entwicklungsansatz (siehe Abbildung 3.6).

Methodische Anforderungen				Technologische Anforderungen							
Integration in ein mechatronisches Entwicklungsvorgehen	Förderung der fachbereichsübergreifenden Zusammenarbeit	Skalierbarkeit und Flexibilität	Effizienz und Effektivität	Integrierte Modellbildung	Interdisziplinarität und Praxisnähe der Beschreibungsmittel	Simulationsgestützte Validierung und Verifikation	Durchgängigkeit der Modellbildung	Wiederverwendbarkeit, Hierarchisierung und Modularisierung	Nutzung für weitere Aufgaben	Erweiterbarkeit	Berücksichtigung von Normen und Standards

Abbildung 3.6: Domänenspezifische Anforderungen

4 Analyse und Bewertung bestehender Ansätze

4.1 Übersicht

Die folgenden Abschnitte analysieren den Stand der Forschung und Technik im Umfeld der Entwicklung automatisierter Systeme, bewerten diesen anhand der in Kapitel 3 herausgearbeiteten Anforderungen und stellen vorhandene Optimierungspotentiale dar. Ein Schwerpunkt liegt auf der methodischen Betrachtung der Geschäftsprozesse. Dem entsprechend werden Vorgehensmodelle zur Systementwicklung untersucht. Im Fokus liegen mechatronische Systeme. Ergänzend wird auch die Methodik der allgemeinen Softwareentwicklung im Hinblick auf die Zielsetzung der Arbeit analysiert. Anschließend folgt eine Vorstellung und Einordnung der Forschungsarbeiten, die sich mit der Entwicklung der Steuerungssoftware automatisierter Systeme unter Nutzung von Modellen beschäftigen. Das Kapitel schließt mit einer zusammenfassenden Bewertung der diskutierten Arbeiten.

4.2 Vorgehensmodelle zur Systementwicklung

Bezüglich der Entwicklung technischer Systeme werden in der Literatur eine Reihe von generischen und für spezielle Aufgabengebiete oder Fachbereiche adaptierte Vorgehensmodelle genannt. Aufgrund der Vielzahl der existierenden Arbeiten beschränken sich die folgenden Ausführungen auf wesentliche und weit verbreitete Ansätze. Eine eingehende Betrachtung von Vorgehensmodellen für die Softwareentwicklung findet sich beispielsweise in *(Dumke 2001)*. Bezüglich einer umfassenden Darstellung methodischer Ansätze für die Entwicklung mechatronischer Systeme wird auf *(Möhringer 2004)* verwiesen.

Im Maschinenbau integrieren die Richtlinien VDI 2221, VDI 2222 und VDI 2223 verschiedene methodische Arbeiten *(Pahl u. a. 2005; Rodenacker 1991; Roth 2000; u. a.)* zu einem international anerkannten Vorgehensmodell für die Produktentwicklung. Der Geschäftsprozess der Entwicklung wird in einzelne Hauptphasen unterteilt, die weitere Arbeitsabschnitte enthalten. Diese werden je nach Aufgabenstellung vollständig, teilweise oder mehrmals iterativ durchlaufen. Mit jedem Arbeitsabschnitt werden festgelegte Ergebnisse generiert, die als Eingangsgröße für den nächsten Prozessschritt dienen.

Albertz *(Albertz 1995)* beschreibt darauf aufbauend ein erweitertes Vorgehensmodell zur Entwicklung von Werkzeugmaschinen. Ziel ist es, durch die Integration von Simulationsmethoden in den Entwicklungsprozess die Optimierung des dynamischen Verhaltens der Baugruppen und des Gesamtsystems zu ermöglichen. Schneider *(Schneider 2000)* ergänzt den Ansatz mit der Intention, die Effizienz der Anwendung von Simulationswerkzeugen für die Analyse der strukturdynamischen Eigenschaften zu erhöhen

und gleichzeitig die Abhängigkeit der Methoden von maschinenspezifischen Eigenschaften zu reduzieren.

Bekannte Vorgehensmodelle im Bereich der Softwareentwicklung sind das Wasserfallmodell, das Spiralmodell und das V-Modell.

Das erweiterte Wasserfallmodell *(Dumke 2001, S. 104)* ist ein sehr einfaches und verbreitetes, sequentielles Vorgehensmodell. Es umfasst die Phasen Anforderungsanalyse, Systemdesign, Programmierung und Modultest, Integrations- und Systemtest sowie Betrieb und Wartung. Jede der genannten Phasen ist durch einen eindeutigen Start- und Endpunkt und die Festlegung konkreter Ergebnisse gekennzeichnet. Iterationen zur vorhergehenden Phase sind möglich, um aufgedeckte Unzulänglichkeiten zu beseitigen. Wesentliche Vorteile des Ansatzes sind die stringente Abgrenzung der Phasen und die damit verbundene einfache Kontrolle des Entwicklungsfortschritts. Als nachteilig ist die mangelnde Flexibilität bezüglich Änderungen und im Vorgehen zu betrachten. Da es schwierig ist, bereits zu Projektbeginn alle Anforderungen eindeutig und endgültig zu bestimmen, ist eine praxisnahe Anwendung auf einfache Projekte beschränkt. Als Defizit ist auch die späte Durchführung der Softwaretests zu betrachten. Änderungsprozesse und die Korrektur von Fehlern sind daher mit hohem Aufwand verbunden.

Das Spiralmodell *(Boehm u. a. 1998)* ist eine Weiterentwicklung des Wasserfallmodells, bei der die Wirtschaftlichkeit der Projektdurchführung im Fokus steht. Das Softwareengineering wird demnach als iterativer Prozess betrachtet, bei dem jeder Iterationszyklus in vier Aufgabengebiete unterteilt und sukzessive bearbeitet wird (siehe Abbildung 4.1 a). Zu Beginn werden die Ziele der jeweiligen Entwicklungsphase definiert, Alternativen zur Realisierung identifiziert und die Rahmenbedingungen festgelegt. Daran schließen sich eine Evaluierung der Alternativen, unter Berücksichtigung der Ziele und Randbedingungen, und die Ableitung von Maßnahmen zur Risikominimierung an. Nach der operativen Durchführung der phasenspezifischen Aufgaben erfolgt die Planung der nächsten Entwicklungsphase. Wird eine Weiterführung des Projektes auf der Basis einer Ergebnisbewertung und der Projektplanung als sinnvoll erachtet, wird der nächste Entwicklungsschritt initialisiert.

Das V-Modell *(Versteegen 2000)* ist ein umfangreiches und sehr detailliertes, aus vier Teilmodellen bestehendes Vorgehensmodell zur Entwicklung von IT-Systemen. Die zueinander in enger Beziehung stehenden Submodelle behandeln das Projektmanagement, das Konfigurationsmanagement, die Qualitätssicherung und die System- bzw. Softwareerstellung. Neben der Zuordnung von Verantwortlichkeiten und der Definition von Meilensteinen und Entscheidungspunkten bezüglich der Projektabwicklung beschreibt das V-Modell auch, welche Tätigkeiten im Rahmen der Systementwicklung zu erledigen sind, welche Ergebnisse generiert und welche Hilfsmittel hierzu verwendet werden können. Ein wesentlicher Unterschied zu anderen Vorgehensmodellen ist

die Vorgabe fester Entwicklungsstrategien. Während der Analysephase und im Rahmen des Entwurfs wird ein Top-down-Ansatz verfolgt, indem die Entwicklungsaufgabe modularisiert und sukzessive verfeinert wird. Die Implementierungs-, Integrations- und Testphase entspricht einer Bottom-up-Strategie zur Synthese des Gesamtsystems (siehe Abbildung 4.1 b).

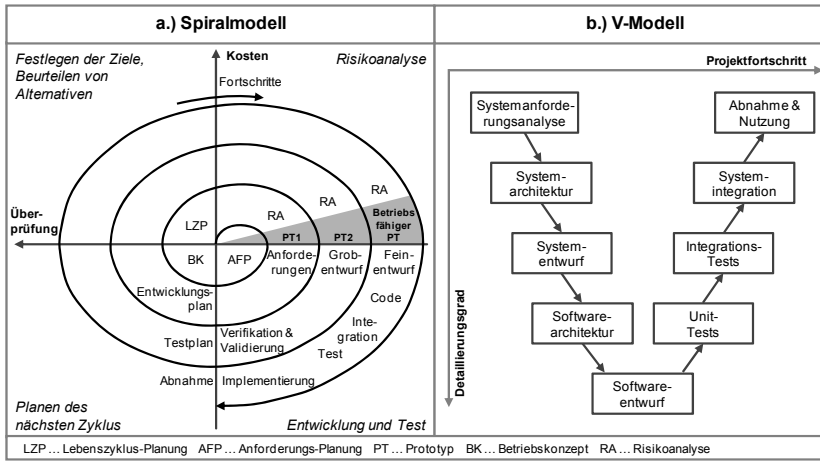


Abbildung 4.1: Spiralmodell und V-Modell

Wesentliche Nachteile des V-Modells sind der hohe Aufwand zur Adaption an organisations- und auftragsspezifische Gegebenheiten, Defizite bezüglich der Einbindung der Auftraggeber und die mangelnde Produktzentriertheit. Mit der Weiterentwicklung „V-Modell XT“ *(Rausch & Broy 2007)* wird deshalb versucht, durch eine stärkere Modularisierung und die Bereitstellung von Möglichkeiten zur vereinfachten Anpassung (Tailoring) die genannten Unzulänglichkeiten zu beseitigen.

Die meisten Vorgehensmodelle für die Entwicklung mechatronischer Systeme adaptieren das V-Modell, da dieses die grundsätzlichen Eigenschaften der mechatronischen Produktentwicklung mit den iterativen Abhängigkeiten zwischen den Entwicklungsphasen widerspiegelt *(Gausemeier u. a. 2000b)*. Gausemeier und Lückel *(Gausemeier & Lückel 2000)* stellen einen im Rahmen des Verbundprojektes EU-MECH[1] erarbeiteten Ansatz vor, bei dem der Schwerpunkt auf der Entwicklung von Prinziplösungen und der domänenübergreifenden Produktkonzeption liegt. Das aus dem V-Modell bekannte Top-down- und Bottom-up-Vorgehen über mehrere Ebenen wird durch eine Gliederung nach den Produktentstehungsphasen (Aufgabenklärung, Konzeption, Entwurf, Ausarbeitung, Fertigung) ersetzt. Die Integration von Werkzeugen zur Modell-

[1] EU-MECH = Entwicklungsumgebungen Mechatronik (Verbundprojekt gefördert durch das BMBF).

bildung und Simulation in den Entwicklungsprozess soll die interdisziplinäre Abstimmung unterstützen und damit eine stringente Produktentwicklung ermöglichen.

Das im Rahmen des Verbundprojektes EQUAL[2] entwickelte 3-Ebenen-Vorgehensmodell *(Stützel & Russ 2005)* legt den Schwerpunkt auf die entwicklungsbegleitende Qualitätssicherung beim Softwareengineering für eingebettete Systeme. Ziel ist die Integration bestehender Entwicklungsprozesse in einen gemeinsamen Geschäftsprozess und die Einführung fachlich/technischer und organisatorischer Synchronisationspunkte auf der System-, Subsystem- und Komponentenebene. Die organisatorische Synchronisation erfolgt im Rahmen des Entwurfs und wird durch die Festlegung ausführlicher Pflichtenhefte unterstützt. Die fachlich/technische Synchronisation geschieht im Zuge der Integration zusammenwirkender Hardware- und Softwarebausteine, wird jedoch bereits in der Entwurfsphase detailliert spezifiziert. Da auch die Entwicklung mechatronischer Systeme in der Regel nicht sofort zum gewünschten Ergebnis führt, definiert das Vorgehensmodell auch eine iterativ-inkrementelle Vorgehensweise zur sukzessiven Optimierung der Produkteigenschaften und -funktionen.

Die zum Teil auf den oben genannten Arbeiten basierende VDI-Richtlinie 2206 *(VDI 2206)* stellt einen praxisorientierten Leitfaden für die Entwicklung mechatronischer Produkte und eine Ergänzung zu domänenspezifischen Richtlinien des methodischen Konstruierens und der Entwicklung von Geräten mit Mikroelektronik *(VDI/VDE 2422)* dar. Die Richtlinie legt den Schwerpunkt auf den Systementwurf und bezieht sich bevorzugt auf mechatronische Systeme, die aus diskreten mechanischen und elektrischen Komponenten und integrierender Informationstechnik bestehen. Im Kern definiert die VDI 2206 ein dreiteiliges Vorgehensmodell (siehe Abbildung 4.2). Während der sogenannte Mikrozyklus eine allgemeine, iterative Vorgehensweise zur Lösung von Problemen beschreibt, spezifiziert der Makrozyklus in Anlehnung an das V-Modell die logische Abfolge der wesentlichen Teilschritte bei der Entwicklung mechatronischer Systeme, legt jedoch deren zeitliche Reihenfolge nicht fest. Als dritter Baustein des Vorgehensmodells werden Prozessbausteine für wiederkehrende Arbeitsschritte im Rahmen des Makrozyklus definiert. Ergänzend behandelt die Richtlinie auch Fragestellungen des modellgestützten Systementwurfs, bleibt jedoch aufgrund der produktspezifischen Anforderungen auf einem generellen Niveau.

[2] EQUAL = Embedded Quality (Verbundprojekt gefördert durch das BMBF).

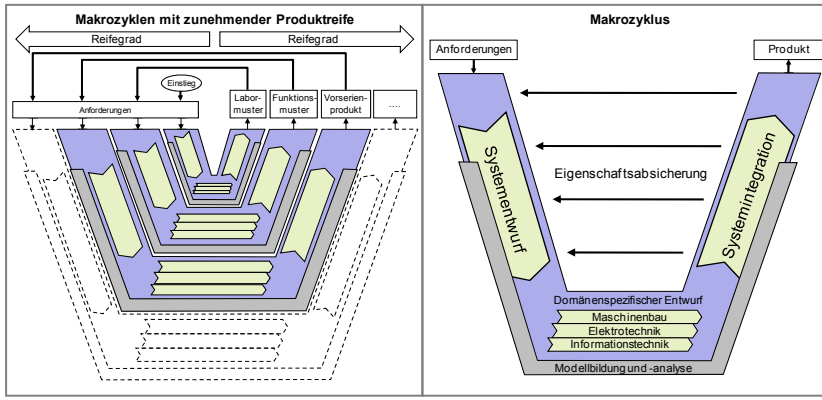


Abbildung 4.2: VDI 2206 – Entwicklungsmethodik für mechatronische Systeme

Neben den Vorgehensmodellen zur Entwicklung mechatronischer Systeme im Allgemeinen existieren auch Ansätze, die speziell auf die Entwicklung automatisierter Systeme ausgelegt sind. Metz *(Metz 1997)* stellt einen Ansatz vor, der Vorgehensweisen der objektorientierten Softwareentwicklung mit den Abläufen bei der Abwicklung von Automatisierungsprojekten verbindet. Die Methode OOMSA[3] integriert verschiedene Sichten auf das Entwicklungsobjekt, wobei jede Sicht bestimmte Teilaspekte des Systems, wie beispielsweise die Struktur, das dynamische Verhalten oder die Funktionalität berücksichtigt. Ausgehend von den Anforderungen werden zunächst Arbeiten bezüglich der Hardwareprojektierung angestoßen und das geforderte Verhalten zur Aufgabenerfüllung analysiert. Darauf aufbauend wird die Aufteilung der Funktionen auf einzelne Teilsysteme festgelegt und deren dynamisches Verhalten detailliert spezifiziert und modelliert. In einem letzten Schritt wird auf der Basis der vorhergehenden Prozessschritte die Implementierung der Steuerungssoftware realisiert.

Die objektorientierte Vorgehensweise ODEMA[4] *(Westkämper & Braatz 2001)* ist speziell auf die Anforderungen verteilter Automatisierungssysteme ausgerichtet. Für unterschiedliche Steuerungsebenen (siehe Abschnitt 3.2.4), Entwicklungsphasen und Detaillierungsgrade der Betrachtung werden Beschreibungsmittel vorgeschlagen, um bestimmte Aspekte der Entwicklung darzustellen. Bezüglich der Definition einzelner Prozessschritte und ihrer Reihenfolge wird die Flexibilität von ODEMA hervorgehoben *(Braatz 2000)*, gleichzeitig aber auch auf die Abhängigkeit der einzelnen Engineeringphasen und Detaillierungsebenen, und damit eine implizite zeitliche Reihenfolge, hingewiesen.

[3] OOMSA = Objektorientierte Modellierung und Synthese von Automatisierungssystemen.

[4] ODEMA = Object-oriented method for the development of technical multi-agent-systems.

4.3 Bewertung der Vorgehensmodelle

Abbildung 4.3 zeigt zusammenfassend eine Gegenüberstellung der beschriebenen Ansätze. Eine vollständige Erfüllung der definierten Anforderungen zur Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme ist bei keinem der vorgestellten Vorgehensmodelle gegeben.

Anforderung / Vorgehensmodell	Förderung der fachbereichsübergreifenden Zusammenarbeit	Parallelisierung der Entwicklung	Skalierbarkeit und Flexibilität: Unterscheidung von Projektarten	Skalierbarkeit und Flexibilität: Adaptierbar an den Projektumfang	Unterstützung von Prinzipien der Wiederverwendung	Integrierte Modellbildung: Berücksichtigung des steuernden Systems	Integrierte Modellbildung: Berücksichtigung des gesteuerten Systems	Durchgängigkeit in der Anwendung	Integration von Simulationsmethoden
Spiralmodell	○	○	○	○	○	●	○	◑	○
Wasserfallmodell	○	○	○	○	○	●	○	◑	○
V-Modell	○	○	○	◑	○	●	○	◑	○
EU-MECH	●	●	○	◑	●	◑	●	◑	◑
3-Ebenen-Vorgehensmodell	●	●	○	◑	●	●	◑	◑	◑
VDI 2206	●	◑	◑	●	◑	◑	◑	●	◑
OOMSA	◑	◑	○	○	◑	●	○	◑	○
ODEMA	○	○	○	○	●	●	○	●	○

● Kriterium erfüllt ◑ Kriterium bedingt erfüllt ○ Kriterium nicht erfüllt

Abbildung 4.3: Bewertung der Vorgehensmodelle

Die Ansätze zur Entwicklung mechatronischer Produkte bieten jedoch eine weitgehende Übereinstimmung und somit eine gute Grundlage zur Entwicklung einer angepassten Vorgehensweise.

4.4 Softwareentwicklung für automatisierte Fertigungssysteme

4.4.1 Übersicht

Die Arbeiten bezüglich der Entwicklung der Software automatisierter Fertigungssysteme unter Nutzung von Modellen lassen sich im Wesentlichen in Ansätze zur Softwareerstellung und zur simulationsgestützten Verifikation und Validierung unterteilen.

4.4.2 Ansätze zur Erstellung der Software

Molt *(Molt 2003)* beschreibt eine domänenübergreifende Technik zur Spezifikation der Software automatisierter Fertigungsanlagen. Hierzu wurde eine semiformale Spra-

che, basierend auf der UML und der SDL[5] entwickelt, die sich auf die notwendigsten Notationselemente beschränkt und deshalb auch disziplinübergreifend leicht verständlich ist, jedoch keine exakten Verhaltensbeschreibungen ermöglicht. Der Fokus der Betrachtung liegt auf der Modellierung der benötigten Softwarefunktionen und somit einer sehr frühen Entwicklungsphase. Trotz des funktionalen Charakters der Beschreibung erfolgt eine Modularisierung und dem entsprechend eine konkrete Abbildung auf definierte Softwaremodule, indem Prinzipien der Objektorientierung angewandt werden. Die nachfolgende Entwicklungsphase des Softwaredesigns wird mit dem Verweis auf konventionelle Software-Spezifikationstechniken nur peripher betrachtet.

In *(Lutz 1999)* wird eine Methode zur Entwicklung der Software für Speicherprogrammierbare Steuerungen von Fertigungseinrichtungen vorgestellt. Besonders die methodische Unterstützung bei der Entwicklung wiederverwendbarer Bausteine und deren Einsatz in Baukastensystemen wird explizit betrachtet. Folglich liegt der Schwerpunkt der Arbeit auf der Strukturierung der Steuerungsprogramme entsprechend der Baukastensystematik der Fertigungseinrichtungen, der Anwendung von Methoden der objektorientierten Softwareentwicklung und der Konfiguration und Parametrierung der Steuerungsprogramme unter Nutzung von Bibliotheken. Prinzipiell wird zwischen einem Anlagen- und einem Steuerungsmodell unterschieden, die in einem Informationsmodell zusammengefasst sind. Im Anlagenmodell werden Anlagenmodellobjekte zur Abbildung der Funktionen der Fertigungseinrichtungen beschrieben und hierarchisch strukturiert. Diesen werden im Steuerungsmodell entsprechende Steuerungsmodellobjekte zugeordnet, die das steuerungstechnische Verhalten der Anlagenmodellobjekte in Form von Zustandsgraphen dokumentieren. Die Modellierung der Fertigungseinrichtungen geschieht anhand der graphischen Darstellung und Verknüpfung der einzelnen Objekte des Anlagenmodells. Darauf aufbauend wird auf der Grundlage der Abhängigkeiten zwischen dem Anlagen- und dem Steuerungsmodell und unter Nutzung von Hilfsfunktionen ein Funktionsbausteindiagramm *(DIN EN 61499-1)* erstellt. In einem letzten Schritt wird unter Verwendung des Funktionsbausteindiagramms und der Zustandsgraphen das Steuerungsprogramm in AWL generiert.

Im Rahmen des Verbundprojektes Föderal *(Litto u. a. 2004)* wurde der Ansatz von Lutz im Hinblick auf Datendurchgängigkeit und Konsistenz der Modelle ausgebaut und durch die Integration der Elektrokonstruktion in die Modellbildung erweitert. Ausgangspunkt für die Konfiguration eines Fertigungssystems ist ein funktional strukturierter, disziplinübergreifender Baukasten, der aufgrund der Dominanz der mechanischen Konstruktion an der montagetechnischen und baulichen Struktur der Fertigungssysteme orientiert ist. Die Modellbildung wird durch die Auswahl und die Verknüpfung einzelner Komponenten *(Korajda u. a. 2004)* realisiert. Jedem dieser Elemente sind eine oder mehrere disziplinspezifische, wiederverwendbare Komponenten zuge-

[5] SDL = Specification and Description Language.

ordnet (siehe Abbildung 4.4). Dies können beispielsweise mechanische Komponenten, Teilschaltungen für den Stromlaufplan oder Softwarekomponenten sein. Auf der Basis der funktionalen Verknüpfung und firmenspezifischer Regeln werden disziplinspezifische Unterlagen, wie beispielsweise der Stromlaufplan, durch Generatoren *(Litto & Lewek 2005)* erstellt.

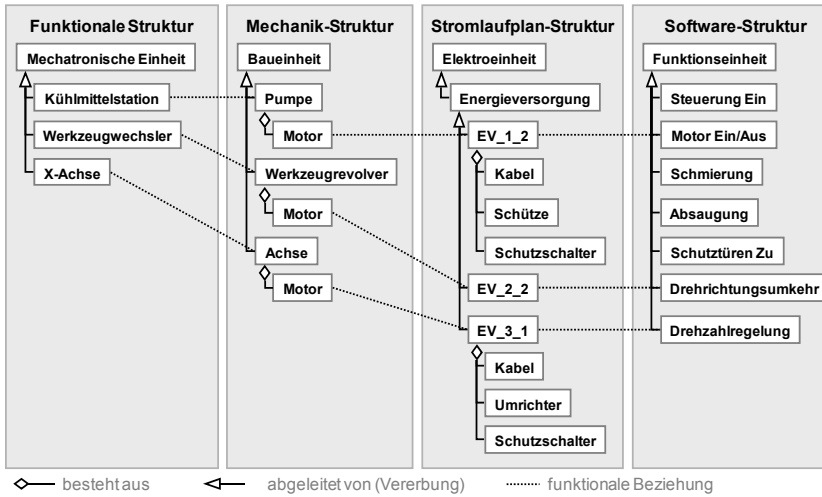


Abbildung 4.4: Disziplinübergreifendes Baukastensystem nach Föderal

In Bezug auf die Softwareentwicklung liegt der Schwerpunkt auf der Konfiguration, das heißt der Selektion und Verknüpfung vorgefertigter Softwarebausteine. Die Koordination und Kommunikation zwischen den Bausteinen realisiert eine speziell programmierte Softwarekomponente, der sogenannte Softwarebus *(Litto u. a. 2004)*. Unzureichend unterstützt werden frühe Entwicklungsphasen und die Entwicklung maschinenspezifischer Funktionen. Eine modellgestützte Absicherung der Funktionalität unter Anwendung von Simulationswerkzeugen wird nicht untersucht.

Osmers *(Osmers 1998)* stellt eine Methode zum Projektieren der Software für Speicherprogrammierbare Steuerungen vor, die auf einem dreidimensionalen Modell des zu automatisierenden Systems und der Interaktion der Entwicklerinnen bzw. Entwickler mit den Elementen des Modells beruht. Hierzu wird ein hierarchisch aufgebautes VR-Modell[6] ausgehend von der Feldebene bis hin zur Anlagenebene definiert. Den strukturellen Einheiten werden dann entsprechende funktionale Einheiten gegenübergestellt und als Softwarebausteine deklariert. Die eigentliche Programmierung ge-

[6] VR-Modell = Virtual Reality-Modell (eine dreidimensionale, meist vereinfachte Repräsentation eines Systems mit Möglichkeiten zur Interaktion durch die Anwenderinnen bzw. Anwender).

schieht unter Anwendung einer Bottom-up-Strategie. Durch die Auswahl von Sensoren und Aktoren im VR-Modell und deren dialogbasierte Verknüpfung mittels Logikbausteinen wird die Funktionalität auf der untersten Ebene definiert. Die Festlegung der Abläufe auf der übergeordneten Hierarchiestufe erfolgt in Analogie zum „Teachen" von Robotern, indem Aktoren im Modell sukzessive bewegt und die gewünschten Sollpositionen protokolliert werden. Auf den weiteren Ebenen wird die Ausführungsreihenfolge der Abläufe untergeordneter Steuerungsebenen festgelegt. Die Programmierung auf diesen Hierarchiestufen wird rein dialogbasiert, ohne Interaktion mit dem VR-Modell, umgesetzt. Eine Betrachtung von Möglichkeiten zur Simulation unter Nutzung des VR-Modells schließt die Ausführungen ab. Weiterführende Arbeiten *(Spath & Landwehr 2000*) beschäftigten sich mit der Vereinfachung der Erstellung der dreidimensionalen Modelle und der intuitiveren Gestaltung der Interaktion mit dem Benutzer. Ein besonderer Vorteil des Ansatzes ist die Nutzung der Modelle zur Diskussion im Entwicklungsteam und für Schulungs- und Diagnosezwecke. Aufgrund der eingeschränkten Möglichkeiten zur Festlegung des Softwareverhaltens und der mangelnden Flexibilität im Vorgehen ist die Anwendung der Methode jedoch auf Steuerungsaufgaben mit geringem Komplexitätsgrad limitiert. Auch die direkte Zuordnung von Softwarefunktionen zu Strukturkomponenten schränkt die Gestaltungsfreiheit im Rahmen der Programmierung ein.

Kohring *(Kohring 1993)* betrachtet im Rahmen seiner Arbeit das systematische Projektieren und Testen der SPS-Software für Werkzeugmaschinen. Hierzu wurde in einem ersten Schritt ein universeller Gliederungsvorschlag für ein Pflichtenheft erarbeitet, das von den Werkzeugmaschinenherstellern an das jeweilige Produktspektrum angepasst werden kann. Besonders betont wird der dynamische Charakter des Dokuments und die Notwendigkeit der sukzessiven Vervollständigung während der Entwicklung. Es begleitet somit den Auftrag von der Angebotsphase bis zur Fertigstellung und stellt das zentrale Bindeglied zwischen den einzelnen Fachabteilungen dar. Neben der eigentlichen Pflichtenheftgestaltung werden auch Ergebnisse bezüglich der Umsetzung der organisatorischen Abläufe bei dessen Erstellung und Konkretisierung beschrieben. Des Weiteren wird eine praxisnahe Systematik zur Ableitung von Testfällen aus den im Pflichtenheft enthaltenen Informationen vorgestellt. Der abschließende Themenkomplex umfasst die Diskussion eines Testsystems zur Absicherung der Softwarequalität. Nähere Informationen hierzu sind Abschnitt 4.4.3 zu entnehmen.

Braatz *(Braatz 2005)* beschreibt einen Ansatz zur objektorientierten Spezifikation von Steuerungen. Hierzu werden bewährte Methoden der Informationstechnik verwendet, die auf der UML in der Version 1.5 als Beschreibungsmittel basieren. Besondere Berücksichtigung finden dezentrale Steuerungen auf der Grundlage von Produktionsagenten. Ein Produktionsagent ist aus logischer Sicht als eine Zusammenfassung der notwendigen Softwarefunktionalität zu verstehen, um autonom bestimmte Funktionen in der Produktion wahrzunehmen. Da dies implizit auch die mechanische Unabhän-

gigkeit der hierfür benötigten Systemkomponenten erfordert, kann ein Produktionsagent als eine aus logischer und physischer Sicht flexible und autonome Einheit bezeichnet werden. Die Spezifikationstechnik beinhaltet daher Möglichkeiten zur Abbildung von Funktionen auf verschiedenen Steuerungsebenen. Insbesondere der Kommunikation zwischen den einzelnen Produktionsagenten, und somit der Funktionalität auf der Zellenebene, wird zentrale Bedeutung beigemessen. Besonderer Wert wird auch auf die Unabhängigkeit der Methode von spezifischen Implementierungen gelegt. Folglich werden konkrete Ausprägungen der Produktionsagenten und deren Implementierung sowie nachstehende Entwicklungsphasen im Rahmen der Arbeit nicht mehr betrachtet.

Ebenfalls mit dem Einsatz der UML bei der Entwicklung automatisierungstechnischer Lösungen beschäftigen sich die Arbeiten von Vogel-Heuser. Untersuchungsschwerpunkte sind wirtschaftliche Aspekte in Bezug auf die Entwicklung, aber auch technologische Fragestellungen, insbesondere im Hinblick auf den Einsatz der UML zur Spezifikation kontinuierlicher Systeme. Durch experimentelle Untersuchungen wurde der Nutzen der Modellierung für die Effizienz der Steuerungsprogrammierung *(Vogel-Heuser & Friedrich 2005)* und die modulare und damit wiederverwendungsgerechte Gestaltung der Programme *(Vogel-Heuser u. a. 2005)* nachgewiesen. Festgehalten wird allerdings auch, dass eine Anpassung an die jeweilige Domäne und die Vorgabe von Richtlinien bezüglich der Anwendung notwendig sind *(Vogel-Heuser & Friedrich 2006)*. In *(Vogel-Heuser & Katzke 2005)* wird mit der UML-PA ein Profil der UML für die Anwendung in der Prozessautomatisierung vorgestellt. Schwerpunkt ist die Erweiterung um Modellkonstrukte zur Spezifikation von Echtzeiteigenschaften sowie charakteristischer Aufgaben der Prozessautomatisierung, wie beispielsweise Regelungen.

Eine Methode zur Modellierung dezentraler Steuerungssysteme wird von Meier *(Meier 2001)* vorgestellt. Die dreistufige Vorgehensweise sieht in einem ersten Schritt die Festlegung der Steuerungsaufgaben vor. Hierzu werden colorierte Petri-Netze eingesetzt, die um das Konstrukt der funktionalen Module ergänzt werden, um eine Strukturierung der Aufgaben zu ermöglichen. Der zweite Schritt beinhaltet die Modellierung der Steuerungstopologie. Dabei werden die Zuordnung der Sensoren und Aktoren im System zu den Steuerungsmodulen und die Kommunikationsverbindungen der Steuerungs-Teilsysteme abgebildet. Zusätzliche Ergänzungen betreffen die Festlegung der Leistungsfähigkeit der einzelnen Module und der Kommunikationsverbindungen. Der abschließende Arbeitsschritt umfasst die Verteilung der Softwarefunktionen auf die einzelnen Steuerungsmodule sowie die Bewertung der Verteilung im Hinblick auf die Leistungsfähigkeit. Für beide letztgenannten Modellierungsschritte werden proprietäre Modellkonstrukte eingesetzt.

Sabbah *(Sabbah 2000)* beschreibt aufbauend auf den Arbeiten von Koch *(Koch 1996)* und Wagner *(Wagner 1997)* eine Methode zur Entwicklung störungstoleranter Steue-

rungen für Fertigungssysteme. Das vorgestellte Konzept berücksichtigt alle Phasen der Störungsbehandlung (Störungserkennung, -lokalisierung, -behebung) und unterstützt den Benutzer durch aktive Führung bei der Wiederaufnahme des regulären Betriebs. Sabbah definiert hierzu einen störungstoleranten Steuerungsbaustein, der neben den eigentlichen Steuerungsfunktionen auch über Mechanismen zur Störungsbehandlung verfügt. Zur Modellierung der Steuerungsaufgaben wird ein Zustandsgraph, eine erweiterte Form der endlichen Automaten, eingesetzt. Die Abbildung der Algorithmen zur Störungsbehandlung erfolgt auf der Grundlage eines Fehlerbaums, d.h. eines strukturierten Graphen, der einen komplexen Ablauf beschreibt, um in Interaktion mit dem Benutzer die Störungsbehandlung durchzuführen. Bezüglich der Modellierung konzentriert sich der Ansatz auf die Darstellung der Beschreibungsmittel zur Abbildung der Funktionalität der Steuerungsbausteine und der Interaktion zwischen den Zustandsgraphen und den Fehlerbäumen. Peripher wird auch die Strukturierung der Software betrachtet. Die Arbeit umfasst folglich im Wesentlichen die Phase des Steuerungsentwurfs. Vorgelagerte Entwicklungsschritte werden nicht berücksichtigt. In Bezug auf nachgelagerte Aktivitäten im Entwicklungsprozess wird eine prototypische Umsetzung auf der Basis kommerzieller Systeme vorgestellt. Deren Architektur sieht eine Realisierung der Steuerungsfunktionen auf einer Speicherprogrammierbaren Steuerung, die Abarbeitung der Fehlerbäume auf einem Industrie-PC und die Interaktion der beiden Teilsysteme über eine feldbusbasierte Kommunikationsverbindung vor.

Der Fokus der Arbeiten von Gewald und Mikk *(Gewald & Mikk 2003)* liegt auf der Überführung von UML-basierten Modellen in die Sprachen der IEC 61131-3 und deren Erweiterung um Wiederverwendungskonzepte. Ziel ist es, Teile einer Automatisierungslösung zu beschreiben und diese während der Projektierung und Entwicklung dezidierter Systeme erneut zu benutzen. Dazu wird eine Spezifikationsmöglichkeit für Funktionsbausteine und Funktionsbausteindiagramme mit der UML vorgestellt. Auf Basis dieser Modellelemente wird ein Komponentenkonzept beschrieben, das eine kontrollierte Anpassung an eine bestimmte Aufgabe durch Parametrierung ermöglicht. Die vorgestellte Methode stellt einen Bottom-up-Ansatz dar, mit dem versucht wird, elementare Funktionsbausteine und ihre Verknüpfung explizit mit der UML zu beschreiben. Dadurch werden die entsprechenden Diagramme relativ komplex. Des Weiteren ist die Methode schwer in einen mechatronischen Engineeringansatz integrierbar, der im Sinne einer Top-down-Strategie zunächst eine abstrakte Spezifikation der Entwicklungsaufgabe und anschließend deren sukzessive Verfeinerung erfordert.

Mit HybridUML *(Bertenkötter u. a. 2004)* existiert eine weitere Methode zur Modellierung der Software automatisierter Systeme. Schwerpunkt ist die Berücksichtigung von Anforderungen bezüglich des Echtzeitverhaltens. Dazu wurde die UML um Notationselemente zur Definition von zeitlich kontinuierlichem Systemverhalten erweitert. Der Ansatz beruht auf einer systemtheoretischen Betrachtung mit einer klaren Trennung der Modellierung von Struktur und Verhalten und der Bereitstellung von Mög-

lichkeiten zur Dekomposition und Hierarchisierung der Modelle. Die Beschreibung kontinuierlicher Verhaltensaspekte geschieht auf der Basis zeitabhängiger algebraischer Bedingungen und Differentialgleichungen, die den Systemzuständen zugeordnet werden. Der Ansatz ist daher als eine Integration hybrider Automaten (siehe Abschnitt 11.2.3) in die UML zu betrachten.

Heverhagen *(Heverhagen 2003)* beschäftigt sich mit der Verknüpfung der unteren Steuerungsebenen mit den übergeordneten Hierarchiestufen der Betriebsdatenerfassung, Produktionsplanung und Auftragssteuerung. Im Fokus liegt die Konzeption und Implementierung geeigneter Schnittstellen, der sogenannten Funktionsbausteinadapter, um unterschiedliche Kommunikationskonzepte zu integrieren. Diese ummanteln Funktionsbausteine der IEC 61131-3 so, dass diese bereits beim Entwurf der PC-basierten Softwaresysteme höherer Steuerungsebenen mit der UML berücksichtigt werden können. Zentrales Element eines Funktionsbausteinadapters ist ein Protokollautomat. Dieser generiert aus nachrichtenbasierten Eingangssignalen Zuweisungen an Schnittstellenvariablen, die wiederum mit den Eingängen der Funktionsbausteine verknüpft werden. Umgekehrt werden Änderungen der Ausgangssignale eines Funktionsbausteins durch den Protokollautomaten des Funktionsbausteinadapters interpretiert, gegebenenfalls in Nachrichten umgesetzt und an übergeordnete Systeme weitergeleitet.

Der Ansatz von Rogério und Carvalho *(Rogério & Carvalho 2004)* bezieht sich auf die modellgetriebene Entwicklung von Steuerungssoftware für Kraftwerke und Kraftwerksverbünde. Im Rahmen eines mehrstufigen Prozesses zur Spezifikation der Software wird der Einsatz einer speziellen Sprache vorgeschlagen. Zur Modellierung struktureller Aspekte werden Elemente der UML eingesetzt. Die Abbildung des Verhaltens der einzelnen Systemkomponenten erfolgt mithilfe einer speziellen Variante von Petri-Netzen. Betrachtet werden lediglich höhere Steuerungsebenen. Besonders die Entwicklung der Funktionalität zur Kommunikation zwischen einzelnen Kraftwerken und deren Interaktion soll durch die beschriebene Methode unterstützt werden. Die Modellierung des Verhaltens der Komponenten konzentriert sich daher auf logische und zeitliche Aspekte der Kommunikation zwischen einzelnen Subsystemen.

Davidson und McWhinnie *(Davidson & McWhinnie 1996)* beschreiben eine objektorientierte Methode zur Entwicklung der Software Speicherprogrammierbarer Steuerungen unter Verwendung der OMT7 für die Modellbildung. Zur Strukturierung der Funktionen werden Objekte definiert, die im weiteren Entwicklungsverlauf als Funktionsbausteine der IEC 61131-3 interpretiert und ausprogrammiert werden. Der systematische Ansatz unterstützt alle Entwicklungsphasen von der Analyse bis zur Implementierung, bezieht aber das zu steuernde System nicht in die Betrachtung mit ein.

[7] OMT = Object Modeling Technique.

4.4.3 Ansätze im Bereich der Verifikation und Validierung

Die Arbeiten von Albert und Tomaszunas diskutieren den Einsatz von Simulationssystemen zur Absicherung der Software Speicherprogrammierbarer Steuerungen von Produktionsanlagen. Tomaszunas *(Tomaszunas 1998)* stellt ein Simulationssystem auf der Basis eines Echtzeitbetriebssystems vor. Der Simulationsrechner wird mittels einer E/A-Schnittstellenkarte mit der realen Steuerung verbunden und simuliert sowohl das gesteuerte System als auch den zugrunde liegenden Prozess. Die Modellierung der Maschinen und Anlagen erfolgt unter Nutzung der Methode ROOM[8] (*Selic u. a. 1994)*. Schwerpunkt der Arbeit ist die Entwicklung von Konzepten zur aufwandsarmen Modellbildung, um die Wirtschaftlichkeit des simulationsgestützten Softwaretests sicherzustellen. Um dies zu erreichen, wird eine komponentenbasierte Modellierung des gesteuerten Systems und der darauf aufbauende Einsatz von Bibliotheksmodulen vorgeschlagen. Der Fokus von Albert *(Albert 1998)* liegt auf der Erarbeitung der zur Modellbildung notwendigen Beschreibungs- und Strukturierungsformen. Er kombiniert hierzu unterschiedliche Methoden zum Entwurf von Softwarearchitekturen und integriert diese zu einem speziell an die Anforderungen der Maschinenmodellierung adaptierten Ansatz.

Das Ziel eines effizienten Aufbaus der Maschinenmodelle wird in der weiterführenden Arbeit von Xu *(Xu 2003)* verfolgt. Der Lösungsansatz impliziert zum einen die Entwicklung eines Modellierungsrahmens für die Erstellung der Maschinenmodelle. Dieser umfasst die Bildung einheitlicher Modellstrukturen, die Festlegung von Konventionen bezüglich Daten und Schnittstellentypen und die Spezifikation einer einheitlichen Form für die Dokumentation der Komponenten. Zum anderen steht die flexible Variantenbildung im Vordergrund. Es wird eine Methode vorgestellt, die den Aufbau adaptiver Module unterstützt. Diese können durch Parametrierung an die jeweilige Maschinenvariante angepasst werden.

Schaich *(Schaich 2001)* entwickelt ebenfalls ein Konzept zur Beschreibung des Verhaltens von Produktionsmaschinen sowie der durch diese auszuführenden Prozesse. Schwerpunkt der Arbeit ist die Darstellung eines Referenzmodells zur Verhaltensbeschreibung, das physikalische und informationstechnische Aspekte gleichermaßen berücksichtigt. Ziel ist es, den Aufwand zur Beschaffung von Informationen zu verringern und Missverständnisse im Verlauf der Entwicklung zu vermeiden. Der zum Teil auf eigens definierten Modellierungstechniken beruhende Ansatz stellt Elemente zur Beschreibung einzelner Prozess- und Maschinenbausteine zur Verfügung, umfasst aber auch Methoden zur Abbildung der Interaktionen zwischen den Modellelementen und den Aufbau von Baukastensystemen. Auf eine formale Definition der einzelnen Bestandteile des Referenzkonzeptes wird zugunsten einer flexiblen Einsetzbarkeit ver-

[8] ROOM = Realtime Object Orjented Modeling.

zichtet. Schaich beschreibt auch den Einsatz der Modellierungstechnik zum einfachen Aufbau von Simulationsmodellen für den Softwaretest. Dieser erfolgt, indem bestehende Bibliotheksbausteine in abstrahierter und anwendergerechter Form dargestellt und mithilfe eines graphischen Editors parametriert und konfiguriert werden.

Auch die Arbeiten von Eckes *(Eckes 2007)* beschäftigen sich mit der Unterstützung der Inbetriebnahme und dem Test der Software für automatisierte Fertigungssysteme. Ausgehend von einem zum Teil bzw. vollständig montierten System und der häufig nicht getesteten Software wird der Anlaufprozess durch ein auf der Augmented-Reality[9] (AR) basierendes Verfahren optimiert. Eckes beschreibt am Rande auch eine Methode zum Entwurf eines Grundgerüsts der Steuerungsprogramme und zum Aufbau von Modellen zum simulationsgestützten Softwaretest. Der Schwerpunkt der Arbeiten liegt jedoch auf der Aufbereitung der Entwicklungsdaten zur AR-gestützten Inbetriebnahme. Die Informationen aus der Entwicklungsphase dienen primär dazu, die möglichen Zustände des Fertigungssystems sowie die verbaute Aktorik und Sensorik zu identifizieren und den aktuellen Zustand dieser Systemelemente während des Anlaufs zu visualisieren. Das zur Unterstützung der Inbetriebnahme und des Softwaretests entwickelte AR-System erlaubt es, verdeckte oder nicht vorhandene Komponenten des Fertigungssystems in Form eines vereinfachten, dreidimensionalen Modells zu visualisieren und deren aktuellen Betriebszustand anhand der graphischen Repräsentation darzustellen. Der Zustand der entsprechenden Softwarekomponenten wird ebenfalls in das Blickfeld des Inbetriebnahmepersonals eingeblendet. Dies geschieht durch das direkte Auslesen expliziter Zustandsvariablen aus der Steuerung und die darauf basierende Visualisierung des aktiven und der möglichen Folgezustände anhand von Zustandsdiagrammen.

Beim Konzept von Kohring *(Kohring 1993)* wird eine zweite Speicherprogrammierbare Steuerung zur Verifikation und Validierung der entwickelten Programme genutzt. Das Verhalten des gesteuerten Systems wird anhand vordefinierter, funktionaler Simulationselemente konfiguriert und parametriert. Dies geschieht unter Zuhilfenahme eines graphischen Editors und der bereits im Pflichtenheft spezifizierten Entwicklungsinformationen. Auf der Grundlage der verwendeten Simulationselemente, korrespondierenden Funktionsbausteinen und der Konfigurations- und Parametrierinformationen wird ein SPS-Programm generiert, welches das Maschinenverhalten nachbildet. Nach der Koppelung der Simulatorsteuerung und der Steuerung mit den zu testenden Programmen über ein Feldbussystem kann das Softwareverhalten getestet werden.

[9] Unter Augmented Reality (Erweiterte Realität) ist die rechnergestützte Erweiterung der Realitätswahrnehmung mit virtuellen Informationen in Echtzeit zu verstehen. Die Erweiterung bezieht sich meist nur auf die visuelle Wahrnehmung und wird realisiert, indem Zusatzinformationen (3D-Modelle, Text, usw.) in das Blickfeld des Nutzers eines entsprechenden technischen Systems eingeblendet werden.

Ein weiterentwickelter Ansatz wird von Dierßen *(Dierßen 2002)* vorgeschlagen. Durch die Koppelung einer realen Steuerung mit einem Modell einer Werkzeugmaschine soll die Inbetriebnahme unterstützt werden. Die entwickelte Simulationsarchitektur besteht aus der im realen System eingesetzten Steuerungshardware und den implementierten Programmen, einer dreidimensionalen Visualisierung und einem echtzeitfähigen System zur Simulation des Maschinenverhaltens. Die über ein Feldbussystem mit der Steuerungshardware verbundene Echtzeitsimulation bildet das Verhalten des gesteuerten Systems und die Kommunikationsfunktionen nach. Die angekoppelte, dreidimensionale Visualisierung des Systemverhaltens erlaubt eine anschauliche Darstellung und Diskussion und ermöglicht so eine einfache Überprüfung der Korrektheit der Steuerungsfunktionalität. Der Schwerpunkt der Arbeit liegt auf der Beschreibung der Architektur des Simulationssystems, der einzelnen Teilsysteme und der Systemkoppelung sowie der Darstellung einer exemplarischen Umsetzung. Darüber hinaus betont Dierßen den Nutzen der virtuellen Maschinenmodelle für die der Entwicklung nachfolgenden Produktlebensphasen. Als mögliche Anwendungsfelder werden der Einsatz für Schulung- und Wartungszwecke sowie die Verwendung der Modelle für Marketing und Verkauf angegeben.

Kreusch *(Kreusch 2002)* erweitert den Anwendungsbereich des simulationsgestützten Softwaretests auf Numerische Steuerungen. Hierzu wird eine Echtzeit-Simulationsumgebung konzipiert, die es ermöglicht, sowohl die Steuerungsfunktionalität als auch die Maschineneigenschaften zu verifizieren. Das Hardware-in-the-Loop-Simulationssystem[10] besteht im Kern aus einer Numerischen Steuerung, einem Echtzeit-Simulationsrechner mit verschiedenen Schnittstellenkarten zum Daten- und Signalaustausch und einer dreidimensionalen Visualisierung. Das Verhalten der Vorschubachsen wird auf der Grundlage vorangegangener Untersuchungen mit MATLAB/Simulink modelliert und ein entsprechender Echtzeitcode für die Abarbeitung auf dem Simulationsrechner erzeugt. Im Simulationsbetrieb erfolgt die Kommunikation mit der Numerischen Steuerung im Lagereglertakt. Die dreidimensionale Darstellung der Achsbewegungen wird mit geringerer Priorität realisiert, erlaubt jedoch eine qualitative Bewertung. Um auch eine quantitative Beurteilung des Verhaltens der Vorschubachsen zu ermöglichen, können die Soll- und Ist-Achskoordinaten mitprotokolliert werden.

Ein ähnliches System wird von Pritschow und Röck *(Pritschow & Röck 2004)* vorgestellt (siehe Abbildung 4.5). Erweiterungen betreffen vor allem die zusätzliche Möglichkeit zur Simulation der Reglerbaugruppen *(Röck & Hamm 2006)* sowie die Anbin-

[10] Hardware-in-the-Loop-(HiL)-Simulation bezeichnet ein Verfahren, bei dem ein reales elektronisches Steuergerät oder eine reale mechatronische Komponente über ihre Ein- und Ausgänge an ein Simulationsmodell des zu steuernden Systems angeschlossen wird.

dung eines echtzeitfähigen Simulationssystems zur Berechnung des dynamischen Verhaltens der Maschinenmechanik.

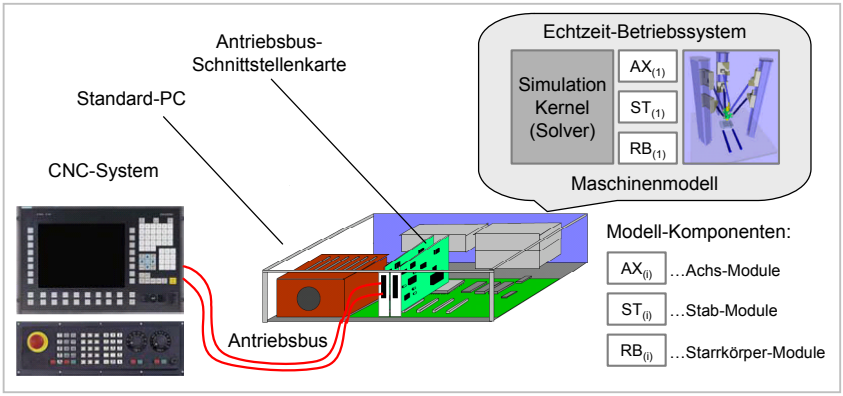


Abbildung 4.5: Hardware-in-the-Loop-Simulation (Röck & Pritschow 2004)

Min *(Min u. a. 2002)* beschreibt eine Lösung zum Test der Software und zur Visualisierung des Verhaltens komplexer Fertigungsanlagen. Der Schwerpunkt liegt auf der Verifikation der Kommunikationsfunktionen. Das Verhalten einzelner Maschinen wird durch entsprechende Simulatoren abgebildet. Diese werden im Sinne eines Hardware-in-the-Loop-Verfahrens an PC-basierte Numerische Steuerungen mit eingebauter SPS-Funktionalität gekoppelt, sind jedoch im Gegensatz zu den bisher beschriebenen Ansätzen direkt in den Steuerungsrechner integriert. Um die Verbindungen zwischen einzelnen Subsystemen abzubilden, wird eine eigens implementierte, Ethernet-basierte Kommunikationsarchitektur eingesetzt. Dieses Netzwerk wird auch zur dreidimensionalen Darstellung des Verhaltes der Anlage genutzt, indem entsprechende Applikationen auf Daten zurückgreifen, die von den Simulatoren zur Verfügung gestellt werden.

Freund und Schluse *(Freund & Schluse 2004)* legen den Schwerpunkt auf die zustandsorientierte Modellierung, Parametrierung und Simulation von technischen Systemen, im Besonderen von Industrierobotern. Um das Verhalten der einzelnen Komponenten abzubilden, wird die Integration von auf Petri-Netzen basierten Algorithmen in das zur Visualisierung eingesetzte 3D-Simulationssystem COSIMIR vorgeschlagen. Das Modell einer Komponente besteht zum einen aus einer dreidimensionalen Repräsentation, zum anderen aus einer Verhaltensbeschreibung in Form eines eigens entwickelten Derivats eines Petri-Netzes, das um Konstrukte zur Abbildung kontinuierlicher Aspekte erweitert wurde. Durch die Koppelung der auf diese Art beschriebenen Hardwaremodule an eine virtuelle Robotersteuerung bzw. Speicherprogrammierbare Steuerung *(Freund u. a 2000)* kann die Software getestet und das Systemverhalten anhand der visuellen Darstellung analysiert und diskutiert werden.

Im Fokus der Arbeit von Wünsch *(Wünsch 2008)* steht die Wirtschaftlichkeit der simulationsgestützten (virtuellen) Inbetriebnahme automatisierter Produktionssysteme. Aufbauend auf der Darstellung von organisatorischen und methodischen Maßnahmen zur Qualitätssicherung stellt Wünsch einen umfangreichen Technologiebaukasten vor, der den aktuellen Stand der Forschung und Technik bei der Inbetriebnahme von Produktionssystemen erfasst und in eine generische Architektur einer geeigneten Simulationsumgebung einordnet. Der Technologiebaukasten dient als Grundlage zum Aufbau eines Inbetriebnahmelabors, das speziell auf das Produktspektrum und die Anforderungen der Anlagenhersteller zugeschnitten ist. Ein weiterer Schwerpunkt liegt auf der Ausarbeitung einer methodischen Vorgehensweise, die, ausgehend von einer Überprüfung des Reifegrades eines Unternehmens in Bezug auf das rechnergestützte Engineering, die notwendigen Schritte festlegt, um eine effiziente und effektive Einführung der virtuellen Inbetriebnahme zu ermöglichen. Um den notwendigen Engineeringaufwand auf ein Minimum zu begrenzen, wird eine kennzahlenbasierte Bewertungsmethode vorgestellt, die es erlaubt, die Anlagenteile mit den höchsten Potentialen für die virtuelle Inbetriebnahme zu identifizieren.

4.5 Bewertung der Ansätze zur Softwareentwicklung

Abbildung 4.6 zeigt eine zusammenfassende Gegenüberstellung der beschriebenen Arbeiten. Diese versteht sich nicht als eine globale Bewertung im Sinne einer objektiven Beurteilung der Qualität eines Ansatzes, sondern als eine vergleichende Übersicht im Hinblick auf die in Abschnitt 3.6 formulierten Anforderungen.

Es ist festzustellen, dass die Entwicklung der Steuerungssoftware automatisierter Fertigungsanlagen unter Nutzung von Modellen nicht durchgängig angewandt wird. Existierende Ansätze beziehen sich aufgrund unterschiedlicher Schwerpunktsetzung auf einige ausgewählte Phasen des Entwicklungsprozesses. Ferner wird, entsprechend dem Fokus der jeweiligen Arbeit, meist nur das gesteuerte oder das zu steuernde System in die Modellbildung mit einbezogen. Eine integrierte Betrachtung im Sinne des Systems Engineering zeichnet sich erst jetzt als ein zukünftiger Trend in der Softwareentwicklung ab *(Weilkiens 2005)*. Die Unterstützung eines mechatronischen Entwicklungsvorgehens, und damit eine effiziente und effektive Entwicklung ausgereifter Lösungen, wird aufgrund der genannten Randbedingungen und der primär domänenspezifischen Modellierungstechniken nur unzureichend umgesetzt.

	Anforderungen / Ansatz	Durchgängigkeit					Integrierte Modellbildung								
		Spezifikation	Entwurf	Implementierung	Integration	Test/Inbetriebnahme	Berücksichtigung des steuernden Systems	Berücksichtigung des gesteuerten Systems	Explizite Unterstützung einer mechatronischen Entwicklung	Interdisziplinarität und Praxisnähe der Beschreibungsmittel	Hoher Formalitätsgrad der Beschreibungsmittel	Skalierbarkeit und Flexibilität	Berücksichtigung von Normen und Standards	Erweiterbarkeit	Unterstützung der Wiederverwendung
Erstellung der Software	Molt 2003	●	◑	○	○	○	●	◑	●	●	◑	◑	○	●	◑
	Lutz 1999	○	●	●	◑	○	●	◑	○	◑	●	●	●	○	●
	Litto u.a 2004	○	●	◑	◑	○	●	◑	●	◑	◑	●	●	◑	●
	Osmers 1998	○	○	●	○	◑	●	●	◑	●	◑	○	◑	○	◑
	Kohring 1999	●	○	○	○	●	◑	◑	◑	◑	◑	○	◑	○	◑
	Braatz 2005	●	●	○	○	○	●	◑	○	○	◑	●	◑	●	◑
	Vogel-Heuser 2005	●	●	◑	○	○	●	○	○	◑	◑	●	◑	●	◑
	Meier 2001	○	●	◑	◑	○	●	○	○	○	●	○	○	○	○
	Sabbah 2000	○	●	◑	○	◑	●	○	○	◑	◑	○	○	○	○
	Gewald & Mikk 2003	○	●	●	○	○	●	○	○	○	◑	◑	◑	●	◑
	Bertenkötter 2004	○	●	◑	○	○	●	◑	◑	○	◑	●	○	◑	◑
	Heverhagen 2004	○	●	◑	○	○	●	○	○	○	●	○	◑	○	○
	Rogério & Carvalho 2004	○	◑	◑	○	○	●	○	○	○	◑	◑	○	◑	○
	Davidson & McWhinnie 1996	◑	●	◑	○	○	●	○	○	○	○	◑	○	●	○
Verifikation & Validierung	Albert & Tomaszunas 1998	○	○	○	◑	●	○	●	○	○	●	●	○	●	●
	Xu 2003	○	○	○	◑	●	○	●	○	◑	◑	◑	○	◑	●
	Schaich 2001	○	○	○	◑	●	○	●	○	◑	◑	◑	○	◑	●
	Eckes 2007	○	◑	◑	◑	●	◑	●	◑	◑	◑	◑	◑	◑	◑
	Dierßen 2002	○	○	○	○	●	○	●	○	●	◑	◑	○	○	○
	Kreusch 2002	○	○	○	○	●	◑	●	○	●	●	○	○	◑	○
	Pritschow & Röck 2004	○	○	○	◑	●	◑	●	○	●	●	◑	○	◑	◑
	Min 2005	○	◑	○	○	●	●	◑	○	◑	◑	○	○	◑	○
	Freund & Schlusse 2004	○	◑	○	◑	●	◑	●	◑	●	●	◑	○	◑	●
	Wünsch 2008	○	○	○	◑	●	○	●	◑	◑	◑	●	○	◑	◑

● erfüllt ◑ teilweise erfüllt ○ nicht erfüllt

Abbildung 4.6: Bewertung von Ansätzen zur Softwareentwicklung

Insbesondere der Zielkonflikt zwischen der Praxisnähe der Beschreibungsmittel und der Notwendigkeit zur Formalisierung der Modelle bietet noch weitere Optimierungspotentiale. Ebenso kann die Entwicklungseffizienz durch eine verbesserte Unterstützung der Wiederverwendung einzelner Modellbausteine, durch eine verstärkte Einbeziehung bestehender Standards und durch die damit einhergehende Verwendung offener Datenformate weiter gesteigert werden.

5 Rahmenkonzept zur modellgetriebenen Softwareentwicklung

5.1 Aufgabenstellung und Übersicht

Wie in Abschnitt 2.1.3 erläutert, umfasst die Konzeption eines modellgetriebenen Ansatzes zur Softwareentwicklung eine Reihe von Aufgaben (siehe Abbildung 5.1). Zum einen gilt es, eine auf die Anforderungen der jeweiligen Domäne zugeschnittene *Modellierungssprache* (Domain Specific Language – DSL) zu entwickeln. Diese muss an die technologischen und fachspezifischen Anforderungen angepasst, gleichzeitig aber auch für eine *interdisziplinäre Modellbildung* im Sinne des Concurrent Engineering geeignet sein. Die Integration der Modellbildung in eine *methodische Vorgehensweise* ist ebenso notwendig. Es muss definiert werden, wie die Methodik in die Organisation der Geschäftsprozesse bei der Entwicklung automatisierter Fertigungssysteme eingebunden werden kann. Dabei ist auch der Charakter der Entwicklungsaufgabe (Änderungs- oder Neukonstruktion, Komplexität der Anlage, usw.) zu berücksichtigen. Des Weiteren ist die Anwendung der DSL in den unterschiedlichen Entwicklungsphasen festzulegen. Hierzu gilt es, entsprechend angepasste *Abstraktionslevels* zu spezifizieren und diesen die jeweiligen Methoden und Notationen der Modellierungstechnik zuzuordnen.

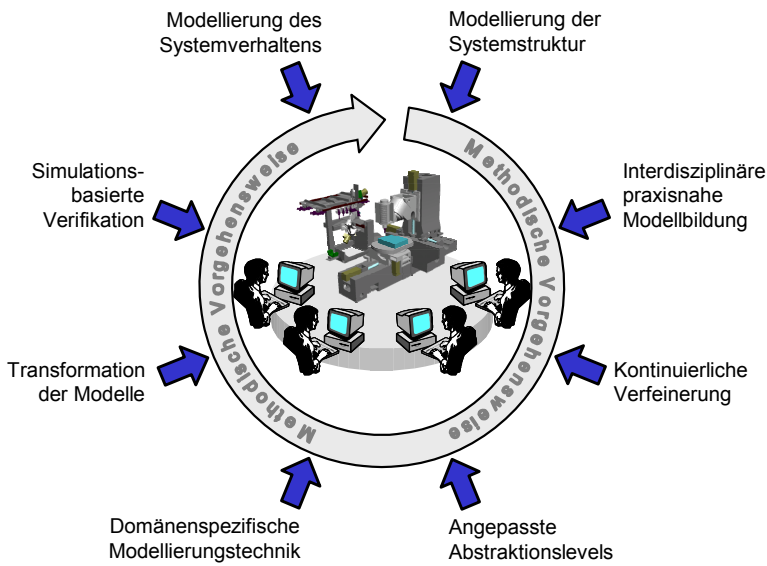


Abbildung 5.1: Aufgaben bei der Konzeption eines modellgetriebenen Ansatzes

Auch die Integration von *Simulationsmethoden* als Mittel zur Verifikation und Validierung der entwickelten Programme und die Festlegung von Strategien zur *Verfeinerung* der Modelle sowie der notwendigen *Transformationen* ist als Aufgabe bei der Konzeption eines modellgetriebenen Entwicklungsansatzes zu betrachten.

Die folgenden Abschnitte umfassen die Vorstellung der wesentlichen Basiskonzepte eines für die Entwicklung der SPS-Steuerungsprogramme von Fertigungssystemen geeigneten Ansatzes (siehe Abbildung 5.2). In Abschnitt 5.2 wird aufbauend auf den Erläuterungen in Abschnitt 2.2 sowie den allgemeinen und domänenspezifischen Anforderungen an die modellgetriebene Softwareentwicklung das fundamentale Konzept der Modellbildung beschrieben und auf grundlegende, anzuwendende Prinzipien eingegangen. Die nachfolgenden Ausführungen in Abschnitt 5.3 beschäftigen sich mit der Darstellung eines geeigneten Vorgehensmodells, das die Anforderungen und Randbedingungen der Entwicklung und den Aufbau von Fertigungssystemen (siehe Abschnitte 3.2 und 3.4) gleichermaßen berücksichtigt.

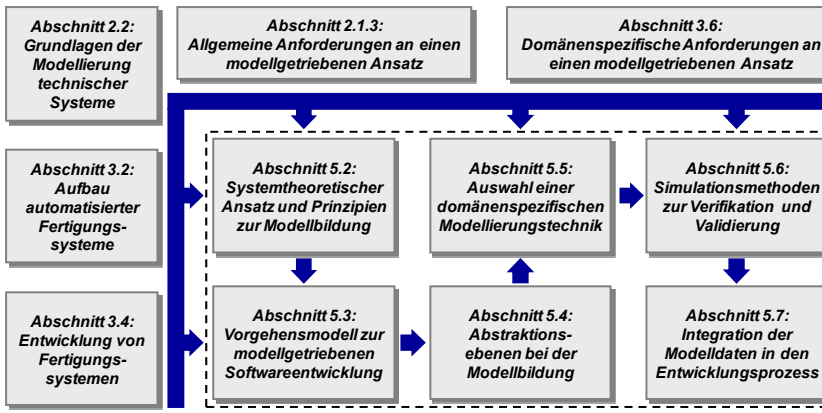


Abbildung 5.2: Aufbau von Kapitel 5 und Bezugnahme zu anderen Teilen der Arbeit

Da das Engineering durch eine schrittweise Detaillierung und Präzisierung der Entwicklungsinformationen gekennzeichnet ist, wird ein Verfeinerungskonzept für die Modelle erforderlich. Im Rahmen von Abschnitt 5.4 werden daher verschiedene Abstraktionsebenen eingeführt und in das Entwicklungsvorgehen integriert. Die nachstehenden Ausführungen in Abschnitt 5.5 betreffen die Auswahl einer geeigneten Grundlage für die Spezifikation einer domänenspezifischen Sprache zur Modellbildung. Hierzu erfolgt zunächst eine abstrakte, systemtheoretische Betrachtung automatisierter Fertigungssysteme und darauf aufbauend die Selektion prinzipiell geeigneter Modellbildungsmethoden. Abschnitt 5.6 diskutiert die Integration von Simulationsmethoden zur entwicklungsbegleitenden Absicherung der Steuerungssoftware und ordnet diese

den definierten Abstraktionsebenen zu, während in Abschnitt 5.7 die Integration des Konzepts in den Gesamtentwicklungsprozess erörtert wird.

5.2 Systemtheoretischer Ansatz und Prinzipien zur Modellbildung

Für die Modellbildung wird aufgrund der fachbereichsübergreifenden Gültigkeit ein systemtheoretischer Ansatz entsprechend Abschnitt 2.2 vorgeschlagen. Das Fertigungssystem wird in einzelne *Teilsysteme* untergliedert, die über ein definiertes *Verhalten* und eine festgelegte *Struktur* verfügen und zueinander in *Beziehung* (Wirkbeziehungen) stehen (siehe Abbildung 5.3).

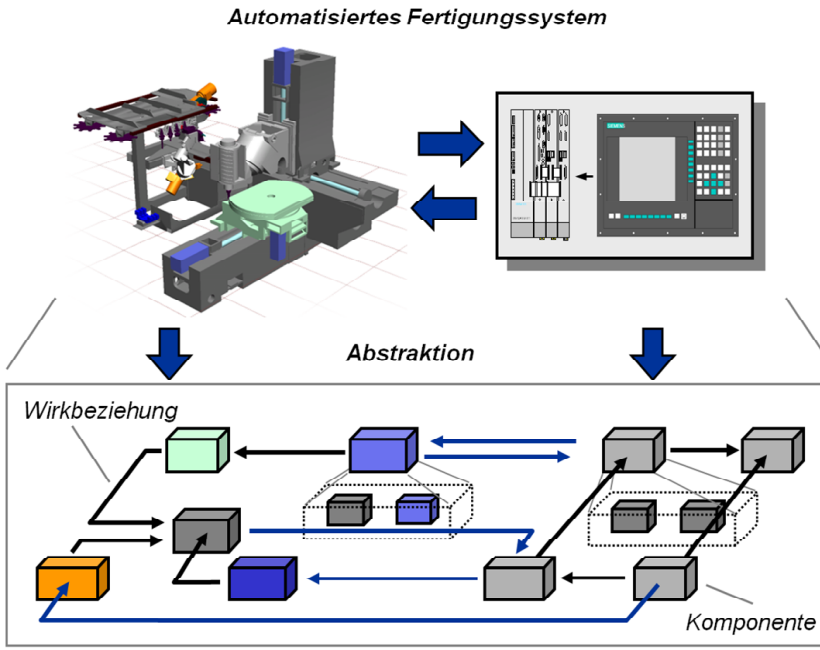


Abbildung 5.3: Systemtheoretischer Ansatz zur Modellbildung

Bei der Modellbildung sind im Hinblick auf die in Abschnitt 3.6 dargelegten Anforderungen einige fundamentale Prinzipien zu berücksichtigen.

Prinzip der Signalflussorientierung

Die *Wirkbeziehungen* zwischen den einzelnen Komponenten sollten auf dem Stoff-, Energie- und Informationsfluss im realen System basieren und anhand der Abstraktion zu *Signalflüssen* nachgebildet werden. Dadurch wird eine realitätsnahe Verknüpfung der Baugruppen bzw. Komponenten erreicht. Durch diese Art der Modellierung wird

nicht nur eine verbesserte Wiederverwendung einzelner Modellbausteine unterstützt, sondern auch die durchgängige Weiterverwendung der Entwicklungsinformationen für andere Aufgaben, wie beispielsweise die Fluid- oder Installationsplanung, ermöglicht.

Prinzip der integriert-distrahierten Modellbildung

Wie in Abschnitt 3.2.3 dargestellt, sind automatisierte Fertigungssysteme als mechatronische Systeme zu betrachten. Charakteristisches Merkmal ist jedoch die Tatsache, dass diese im Gegensatz zu klassischen mechatronischen Produkten zwar eine *funktionale*, nicht aber eine *räumliche* Integration aufweisen. Das gesteuerte Teilsystem ist aus diskreten mechanischen, elektromechanischen, hydraulischen und pneumatischen Komponenten aufgebaut, das steuernde System besteht aus einer oder mehreren Steuerungen. Während die funktionale Integration also eine gemeinsame Modellbildung von gesteuertem und steuerndem System erfordert, ist aufgrund der lokalen Verteilung zur Erfüllung der Anforderungen eine getrennte Modellierung der beiden Teilsysteme (*physikalisches Modell*[1] und *Steuerungsmodell*) notwendig. Anhand der Umsetzung mehrerer Beispiele aus der industriellen Praxis konnte gezeigt werden, dass Anwenderinnen bzw. Anwender bei der integrierten Modellbildung dazu neigen, die Funktionalität des steuernden und des gesteuerten Teilsystems zu vermischen, wodurch sowohl die Wiederverwendbarkeit einzelner Modellbausteine als auch deren durchgängige Nutzung beeinträchtigt wird. Die Modellbildung erfordert daher eine komponentenorientierte Strukturierung der Modelle, die sich an der *Baustruktur* der Maschinen bzw. Anlagen oder ihrer Teilsysteme orientiert. Die Integration der einzelnen Subsysteme sollte analog zur realen Umsetzung nach dem Prinzip der Signalflussorientierung erfolgen.

Entsprechend der Funktion eines automatisierten Fertigungssystems muss dessen physikalisches Modell auch die Abbildung der *Prozesse* (Fertigungsprozesse oder Hilfsprozesse) umfassen, insofern dies für die Erstellung der Steuerungssoftware oder deren simulationsgestützte Verifikation und Validierung notwendig ist.

Prinzip der hierarchischen Komposition

Durch die Zusammensetzung einzelner Komponenten zu Subsystemen wird eine hierarchische Struktur des betrachteten Systems aufgebaut. Die Hierarchisierung der Modelle sollte im Wesentlichen auf der physikalischen Umsetzung basieren, die durch die mechanische Konstruktion festgelegt wird. Submodelle entsprechen folglich einzelnen Baugruppen bzw. übergeordneten Teilsystemen, die bestimmte Funktionen erfüllen. Da diese Subsysteme zunehmend als Bestandteil eines Baukastensystems entwickelt werden, unterstützt diese Vorgehensweise die einfache und schnelle Konfiguration der Fertigungssysteme bzw. Modelle entsprechend den Anforderungen des Kunden.

[1] Die Bezeichnung „physikalisches Modell“ für das gesteuerte System ist nicht vollständig zutreffend, unterstreicht jedoch dessen Charakter als System, das durch physikalische Gesetzmäßigkeiten gut beschreibbar ist.

Prinzip der korrespondierenden Teilsysteme

Die Strukturierung der Steuerungssoftware sollte sich am Aufbau des physikalischen Teilsystems orientieren. Physikalisch vorhandene Systemkomponenten sollten bei der Modellbildung durch korrespondierende Softwarebausteine ergänzt werden. Dieses Grundprinzip ist zwar in der Praxis nicht immer zweckmäßig und teilweise auch nicht einhaltbar. So hat beispielsweise eine Softwarekomponente zur Überwachung von Sicherheitsfunktionen nicht zwangsläufig eine hardwaremäßige Entsprechung. Dennoch sollte es aber im Hinblick auf eine modulare Systemgestaltung und damit im Sinne der Wiederverwendbarkeit, der Konfigurierbarkeit und einer effizienten Entwicklung – soweit sinnvoll und realisierbar – beachtet werden.

Prinzip der hinreichenden Detaillierung

Der Detaillierungsgrad der Modelle ist entsprechend dem Modellzweck zu gestalten. Daher gilt es, im Rahmen der Modellbildung die Exaktheit der Nachbildung von Struktur und Verhalten an die Anforderungen der Entwicklungsaufgabe und die jeweiligen Entwicklungsphase anzupassen. Zu genaue Abbildungen der realen Zusammenhänge führen bezüglich der Aussagefähigkeit nicht zwangsläufig zu einem Mehrwert, resultieren aber in einem zum Teil unverhältnismäßig hohen Modellierungsaufwand. Gerade in Bezug auf den simulationsgestützten Softwaretest sind zu detaillierte Modelle aufgrund des hohen Rechenaufwandes nicht vorteilhaft.

Prinzip der adaptierten Modellierungstechnik

Die Modellierungstechnik zur Abbildung des Verhaltens sollte sich an den systemtechnischen Eigenschaften der jeweiligen Komponenten bzw. Subsysteme orientieren (siehe Abschnitt 2.2.4). Dem entsprechend sind Methoden einzusetzen, die auf den Charakter des nachzubildenden (Teil-)Systems abgestimmt sind und eine hinreichende Präzisierung zulassen.

5.3 Vorgehensmodell zur modellgetriebenen Softwareentwicklung

Als Grundlage für eine methodische Vorgehensweise zur interdisziplinären Entwicklung im Sinne des Concurrent Engineering wird die VDI-Richtlinie VDI 2206 *(VDI 2206)* verwendet. Diese erfüllt die domänenspezifischen Anforderungen weitestgehend (siehe Abschnitt 4.3) und ist speziell auf die Entwicklung funktional integrierter mechatronischer Produkte zugeschnitten *(VDI 2206, S. 4)*. Als weiterer Vorteil wird die Möglichkeit zur Definition von *Prozessbausteinen* betrachtet. Diese bilden die Basis zur Bearbeitung von im Rahmen der Entwicklung wiederkehrenden Aufgabenstellungen und erlauben eine Anpassung der Vorgehensweise an die Besonderheiten bei der Entwicklung von Fertigungssystemen, den Umfang der Aufgabenstellung und die Konstruktionsart (Neukonstruktion, Änderungskonstruktion, usw.). Abbildung 5.4 zeigt eine Gegenüberstellung der in Abschnitt 3.4 beschriebenen Entwicklung von Fertigungssystemen mit einem interdisziplinären Ansatz in Anlehnung an VDI 2206.

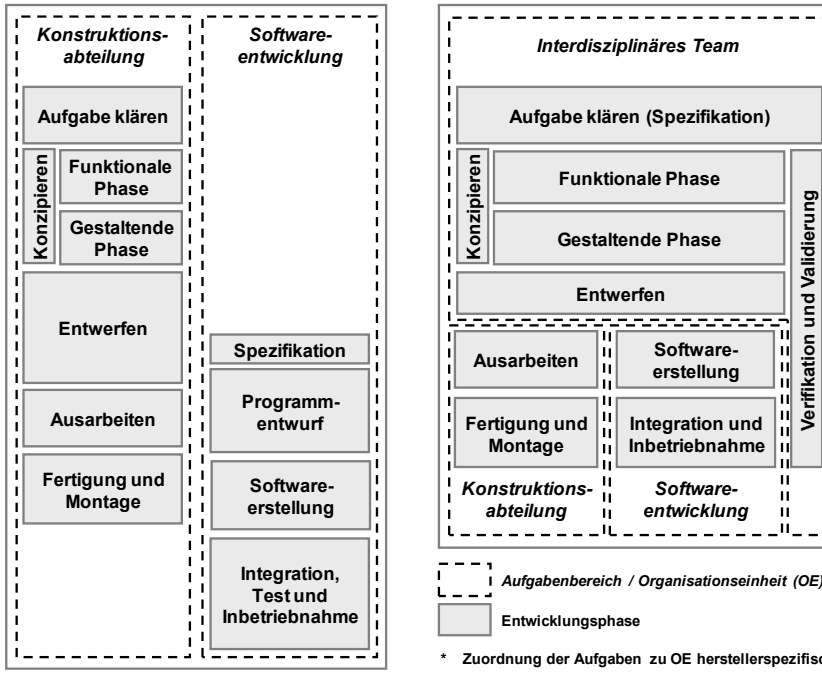


Abbildung 5.4: Traditionelle Entwicklung (links) und interdisziplinärer Ansatz (rechts)

Der dargestellte Ansatz zeichnet sich insbesondere durch ein gemeinsames Engineering in den frühen Entwicklungsphasen aus. Neben der Nutzung von Synergieeffekten, die zu einer Steigerung der Produktqualität führen, wird dadurch auch das frühzeitige Erkennen von Fehlern und folglich eine effiziente und effektive Entwicklung ermöglicht. Abbildung 5.5 zeigt ein iteratives Vorgehensmodell zur Umsetzung der in Abbildung 5.4 dargestellten Entwicklungsphasen mit Bezug auf das modellgetriebene Softwareengineering und die fachbereichsübergreifende Abstimmung. Dieses definiert somit einen flexiblen Geschäftsprozess zum Aufbau der Modelle von automatisierten Fertigungssystemen.

Für das Vorgehensmodell werden sechs Prozessbausteine festgelegt, welche die Aufgaben der Entwicklung und im Speziellen der Modellbildung einordnen und beschreiben.

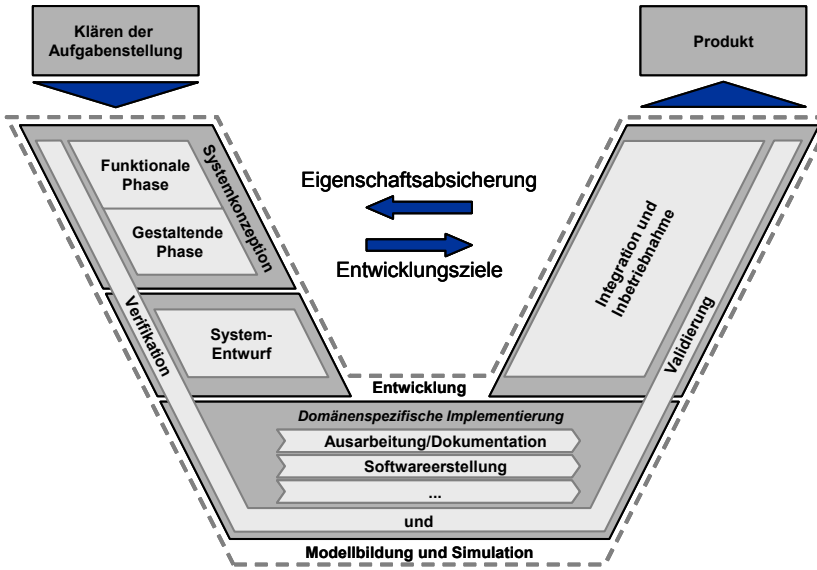


Abbildung 5.5: Vorgehensmodell zur modellgetriebenen Softwareentwicklung

Klären der Aufgabenstellung

Die Klärung der Aufgabenstellung umfasst die Festlegung von Kostenzielen und Leistungsdaten (Arbeitsraum, Bearbeitungsaufgaben, usw.) mit Bezug auf die Ergebnisse von Marktanalysen und Kundenanforderungen. Von Belang für die Softwareentwicklung ist neben den Leistungsdaten auch der Automatisierungsgrad der Maschinen und Anlagen. Darüber hinaus gilt es festzulegen, welche kundenspezifischen Randbedingungen und Standards zu beachten sind. Um die Betriebs-, Wartungs- und Lagerhaltungskosten zu minimieren, bestimmen firmeninterne Richtlinien der Kunden häufig die einsetzbare Steuerungshardware und schränken die Auswahl der verwendbaren mechanischen, elektromechanischen, pneumatischen und hydraulischen Komponenten ein. Darüber hinaus werden auch die Programmiersprachen festgelegt, um dem eigenen Instandhaltungs- und Wartungspersonal bei geringfügigen Problemen die Prüfung von Software- oder Hardwarefehlern zu ermöglichen. Ergebnis der Analysephase ist die Dokumentation aller technischen, wirtschaftlichen und organisatorischen Anforderungen in Form eines Pflichtenheftes. In Bezug auf die modellgetriebene Softwareentwicklung ist dieses als Eingangsgröße für den nachfolgenden Entwicklungsschritt der Konzeption zu betrachten.

Systemkonzeption

Die Aufgaben bei der Systemkonzeption (siehe Abbildung 5.6), dem wichtigsten Abschnitt der Softwareentwicklung *(Lutz 1999, S. 26)*, können in eine funktionale und

eine gestaltende Phase unterteilt werden, wobei der Übergang zwischen beiden Phasen fließend und von der Konstruktionsart abhängig ist. In einem ersten Schritt werden auf Basis der Anforderungen die notwendigen Funktionen des automatisierten Fertigungssystems festgelegt. Da die zu lösende Entwicklungsaufgabe in der Regel zu komplex ist, um daraus unmittelbar die technische Realisierung abzuleiten, ist es zweckmäßig, die Gesamtfunktionalität in einzelne Teilfunktionen aufzugliedern und in Form einer *Funktionshierarchie* zu strukturieren. Ansatze hierfür werden beispielsweise von Pahl *(Pahl u. a. 2005)* oder Langlotz *(Langlotz 2000)* vorgestellt. Böger *(Böger 1998)* beschreibt ein Referenzmodell zur Funktionsstrukturierung, das für Werkzeugmaschinen geeignet ist. Demnach werden Hauptfunktionen und Nebenfunktionen unterschieden. Während erstere primär für den Bearbeitungsprozess relevant sind (Schnittbewegungen erzeugen, Werkzeug fixieren, usw.), da sie im Kraftfluss liegen, sind Nebenfunktionen (Werkzeugbereitstellung, Werkstückbereitstellung, Prozesskühlung, usw.) als unterstützend zu betrachten. Im Rahmen der Arbeit sind diese jedoch von vorrangigem Interesse, da deren Steuerung und Überwachung üblicherweise durch Speicherprogrammierbare Steuerungen realisiert wird. Eine weitere, speziell für Fertigungssysteme geeignete Strukturierung der Funktionen, die sich an der Baustruktur der Maschinen und Anlagen orientiert, wurde von der FWF[2] *(Pörnbacher & Wünsch 2004, S. 25)* erarbeitet. Demnach werden unter anderem Funktionseinheiten, Funktionsgruppen und Funktionsuntergruppen unterschieden. Während Funktionseinheiten elementare Funktionen, wie beispielsweise das Spannen des Werkzeuges in der Spindel übernehmen, realisieren Funktionsuntergruppen und Funktionsgruppen komplexere übergeordnete Maschinenfunktionen und die Koordination der untergeordneten Einheiten. Ergebnis der funktionalen Phase ist eine hierarchische Strukturierung der Funktionen, die zur Umsetzung der im Pflichtenheft beschriebenen Aufgaben erforderlich sind.

Erster Schritt der gestaltenden Phase ist der Aufbau der prinzipiellen Baustruktur der Maschinen und Anlagen sowie der Softwarearchitektur und deren Abbildung in einem Modell. Die Baustruktur wird weitestgehend durch die mechanische Konstruktion auf der Grundlage diverser Konstruktionsprinzipien (montagegerecht, fertigungsgerecht, kostengünstig, usw.) festgelegt. Die parallel dazu stattfindende Konzeption der Softwarearchitektur orientiert sich soweit möglich an der Strukturierung der mechanischen, hydraulischen, pneumatischen und elektromechanischen Einheiten bzw. Baugruppen der Maschinen und Anlagen und berücksichtigt darüber hinaus ergänzend im Pflichtenheft angegebene sowie technologische Restriktionen der Umsetzung. Anschließend erfolgt die *Zuordnung der Funktionen* der Funktionshierarchie zu den strukturellen Einheiten und die Beschreibung von deren wesentlichen Eigenschaften anhand von Attributen. Ergebnis ist ein hierarchisches Modell des Fertigungssystems, das sowohl das steuernde wie auch das gesteuerte System umfasst, wobei deren Kom-

[2] Forschungsvereinigung Werkzeugmaschinen und Fertigungstechnik e. V.

ponenten durch eine Menge von Funktionen und Eigenschaften in ihrer Aufgabe beschrieben werden.

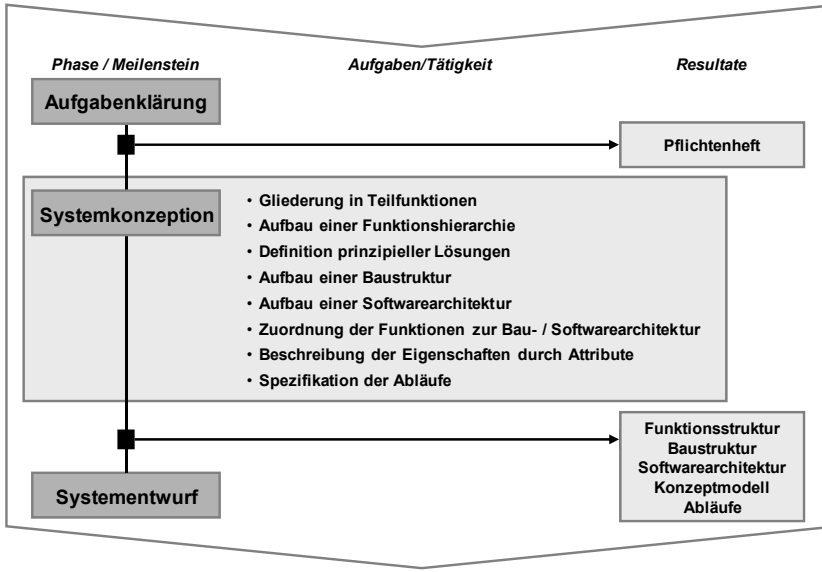


Abbildung 5.6: Prozessbaustein zur Systemkonzeption

Von besonderer Bedeutung für die Entwicklung der Steuerungsprogramme in dieser frühen Entwicklungsphase ist nach Untersuchungen der FWF *(Pörnbacher & Wünsch 2004)* die detaillierte Spezifikation der *auszuführenden Abläufe*, ohne die technische Realisierung im Detail zu betrachten. Daher gilt es, diese bereits im Rahmen der Systemkonzeption hinreichend zu modellieren. Diese Vorgehensweise ermöglicht es, nicht nur die Zuordnung der Funktionen zu den jeweiligen Systemkomponenten und die Systemstruktur weiter zu detaillieren, sondern dient auch der Diskussion und Abstimmung der Entwicklung im interdisziplinären Team. So kann beispielsweise die notwendige Sensorik festgelegt werden, die zur eindeutigen Identifikation des Zustandes einzelner Teilsysteme durch die Steuerung notwendig ist. Darüber hinaus stellt die Definition der Abläufe auch eine wesentliche Grundlage für die Verifikation der Softwarefunktionalität dar.

Ein weiterer Schritt in der gestaltenden Phase der Konzeption ist die Detaillierung der notwendigen Funktionen der Komponenten. Diese Aufgabe wird bewältigt, indem diesen soweit möglich bestehende Prinziplösungen zugeordnet werden. Sind vorhandene Ansätze in Bezug auf die Anforderungen zur Funktionserfüllung nicht geeignet, so werden prinzipiell neue Konzepte oder Lösungsvarianten entwickelt. Analog dazu

wird die Softwarefunktionalität detailliert und anhand geeigneter Beschreibungsmittel (siehe Kapitel 6) abgebildet. Ergebnis ist ein *Konzeptmodell* des physikalischen Teiles des Fertigungssystems sowie der Steuerungssoftware. Ergänzt wird dieses durch eine dreidimensionale Darstellung, die durch die mechanische Konstruktion aufgebaut wird. Hierzu werden in der industriellen Praxis üblicherweise 3D-CAD-Systeme eingesetzt.

Systementwurf

Im Rahmen des Systementwurfs wird das Konzeptmodell zu einem *Entwurfsmodell* verfeinert. Dabei werden die eingesetzten prinzipiellen Lösungen detailliert und unter Berücksichtigung der Anforderungen dimensioniert und geometrisch ausgearbeitet oder als Zukaufteile ausgewählt. Aufgrund der Detaillierung der Entwicklungsinformationen ist eine Verfeinerung der Systemstruktur und insbesondere des Verhaltens der jeweiligen Modellkomponenten möglich. Trotz der Tatsache, dass einzelne Baugruppen automatisierter Fertigungssysteme verstärkt durch Zulieferer bereitgestellt werden, müssen diese bei der Modellbildung mit berücksichtigt werden. Wenn diese Teilsysteme über eine eigene Steuerung verfügen, reicht es in der Regel aus, die *Schnittstellen* zu dieser exakt zu spezifizieren. Ist dies nicht der Fall, so müssen die für die Softwareentwicklung relevanten Komponenten des physikalischen Teilsystems (zumindest in vereinfachter Form) und die entsprechenden Softwarebausteine im Modell ergänzt werden. Weitere Verfeinerungen betreffen die Modellierung von Systemkomponenten, die über keine Entsprechung im physikalischen Teilmodell verfügen, und die Ergänzung um zusätzliche, softwarerelevante Funktionen und Parameter. Als Beispiele hierfür seien Laufzeitüberwachungen, Verriegelungen oder Betriebsarten genannt. Ziel der Modellbildung ist es, anhand der domänenspezifischen Modellierungssprache das Verhalten aller Teilsysteme möglichst vollständig und hinreichend genau abzubilden und zu formalisieren.

Softwareerstellung (Implementierung)

Die Phase der Implementierung umfasst im Wesentlichen die Detaillierung des Steuerungsmodells. Das physikalische Teilmodell ist nur insoweit zu verfeinern, dass Modellparameter, die im Rahmen der Ausarbeitung durch die mechanische Konstruktion festgelegt werden, anzupassen sind. Die Softwareentwicklung hingegen muss zum einen die Datentypen der Ein-, Ausgangs- und Zustandsgrößen der Softwarekomponenten spezifizieren. Darauf aufbauend können die Adressbereiche für die Kommunikation mit dem physikalischen Teilsystem festgelegt werden. Hierbei ist in der Regel eine Zuteilung zu bestimmten Funktionen bzw. Komponenten entsprechend internen Richtlinien zu beachten. Zum anderen gilt es, auf Basis der Anforderungen, der Komponenten des physikalischen Teilsystems und der notwendigen Ein- und Ausgänge, die Architektur der Steuerungshardware und deren räumliche Verteilung zu definieren. Weitere Aufgaben betreffen die Spezifikation der Kommunikationsverbindungen sowie die Festlegung von Namenskonventionen, Zielsprachen und weiteren Randbedingungen

für die nachfolgende Erstellung der Steuerungsprogramme durch entsprechende Codegeneratoren. Dabei ist auch die Einbindung bestehender Codefragmente aus Bibliotheken oder von Systemanbietern zu beachten. Eine vollständige Generierung der Steuerungsprogramme ist zwar anzustreben, in der Praxis jedoch meist nicht mit vertretbarem Aufwand umzusetzen. Daher sind manuelle Ergänzungen der generierten Software ebenfalls als Teil der Implementierungsphase zu betrachten.

Integration und Inbetriebnahme

Im Anschluss an die Erstellung der Steuerungsprogramme erfolgt der Prozessschritt der Integration und Inbetriebnahme. Bezüglich der Softwareentwicklung besteht die Aufgabe in der Zusammenführung einzelner Teilsysteme, der Integration von Zulieferkomponenten und der abschließenden Verifikation des Zusammenwirkens zwischen der Steuerungssoftware und dem Fertigungssystem. Darüber hinaus gilt es, eine Feineinstellung der Systemparameter, wie beispielsweise von Laufzeitüberwachungen, Verfahrwegen und -geschwindigkeiten oder Endschaltern vorzunehmen, und Abläufe zu optimieren. In Bezug auf den Modelleinsatz spielt das physikalische Teilmodell in diesem Prozessschritt die entscheidende Rolle. In einem ersten Schritt ist eine Vorab-Inbetriebnahme des Fertigungssystems oder einzelner Baugruppen anhand des Modells und unter Nutzung von geeigneten Simulationssystemen zu realisieren. Da auch das Modell der mechanischen, elektromechanischen, hydraulischen und pneumatischen Komponenten und ihrer Interaktion nicht alle Eigenschaften des realen Systems vollständig abbilden kann oder Modellparameter aufgrund mangelnder Erfahrung nicht richtig eingestellt werden können, erlaubt auch eine simulationsgestützte Inbetriebnahme[3] nicht die Optimierung aller Systemparameter und die vollständige Beseitigung der Softwarefehler. Untersuchungen *(Wünsch 2008; Stetter 2005)* zeigen jedoch, dass es möglich ist, durch Anwendung geeigneter Methoden bis zu 85 Prozent der Unzulänglichkeiten zu beseitigen.

Zur vollständigen Inbetriebnahme wird das physikalische Teilmodell sukzessive durch das reale Fertigungssystem ersetzt. Diese Vorgehensweise ermöglicht es, die Inbetriebnahme entsprechend dem aktuellen Stand der Fertigungs- und Montagearbeiten anzupassen und somit parallel zu diesen Tätigkeiten durchzuführen, wodurch die Durchlaufzeit bei der Auftragsabwicklung entscheidend verkürzt werden kann.

Verifikation und Validierung

Die Verifikation und Validierung der Modelle und der Steuerungssoftware wird parallel zur Entwicklung durchgeführt. Sie umfasst alle Entwicklungsschritte von der Konzeption der Maschinen und Anlagen bis zur Implementierung und geht fließend in die Integrations- und Inbetriebnahmephase über. Ziel ist es, die Entwicklung weitestgehend anhand der Modelle abzusichern und damit Unzulänglichkeiten bei der Abstimmung oder Fehlentwicklungen möglichst frühzeitig zu erkennen. Die fundamentale

[3] Die simulationsgestützte Inbetriebnahme wird häufig auch als virtuelle Inbetriebnahme bezeichnet.

Basis für die Verifikations- und Validierungsphase stellen geeignete Simulatoren dar. Diese sind in ihrer Funktionalität an die Granularität und den Detaillierungsgrad der Modelle und damit der Entwicklung anzupassen. Gleichzeitig müssen die Modelle formal soweit ausgearbeitet sein, dass ihre Interpretierbarkeit durch den Simulator sichergestellt ist. In der Konzeptionsphase hat die Simulation einen unterstützenden Charakter und dient vorwiegend der Darstellung und anschaulichen Repräsentation von umzusetzenden Abläufen. Mit zunehmendem Detaillierungsgrad sind darüber hinausgehende Aufgaben möglich. Ziel der Entwurfsphase ist die Verifikation des Steuerungsmodells anhand des physikalischen Teilmodells. Mit dem Übergang in die Integrations- und Inbetriebnahmephase liegt der Fokus auf der Überprüfung des Softwarecodes, dem Test der Schnittstellen zu sekundären Systemen und der Voreinstellung der Softwareparameter. Die Inbetriebnahme der Software kann somit zum Teil auch als abschließende Aufgabe der Verifikations- und Validierungsphase betrachtet werden.

5.4 Abstraktionsebenen bei der Modellbildung

Zum Aufbau der Modelle werden im Rahmen dieser Arbeit drei Abstraktionsstufen definiert (siehe Abbildung 5.7). Für die frühen Entwicklungsphasen wird eine *funktionale Modellbildung* vorgeschlagen. In einem ersten Schritt gilt es, die im Rahmen der Konzeption den einzelnen Systembausteinen zugeordneten Funktionen und Eigenschaften im interdisziplinären Team unter Nutzung geeigneter Modellnotationen zu beschreiben. Im Fokus stehen die wichtigen Funktionalitäten, die dem aktuellen Entwicklungstand entsprechen, und die Interaktion einzelner Komponenten. In dieser Phase ist die technologische Umsetzung sowohl auf der softwaretechnischen wie auch auf der konstruktiven Ebene noch von sekundärer Bedeutung. Wichtig ist, dass eindeutig spezifiziert wird, wie bestimmte Funktionen in ihrem logischen Ablauf aussehen sollen und welche Attribute eine bestimmte Komponente kennzeichnen. Ebenso ist entscheidend, dass trotz der unterschiedlichen Denkweise und Begriffswelt der beteiligten Entwicklungsdisziplinen ein gemeinsames Verständnis der Maschinenfunktionen entwickelt und abgebildet wird. Diese Form der Modellbildung ist folglich der *präskriptiven (vorschreibenden) Modellierung* zuzuordnen.

Der Übergang zur nachfolgenden Abstraktionsebene ist als Wechsel von der präskriptiven zur *deskriptiven (beschreibenden) Modellierung* zu betrachten. Im Gegensatz zur übergeordneten Detaillierungsebene gilt es, das zur Funktionserfüllung notwendige Verhalten einzelner Modellkomponenten vollständig zu beschreiben und dabei insbesondere die Abhängigkeit zwischen den verschiedenen Funktionen zu beachten.

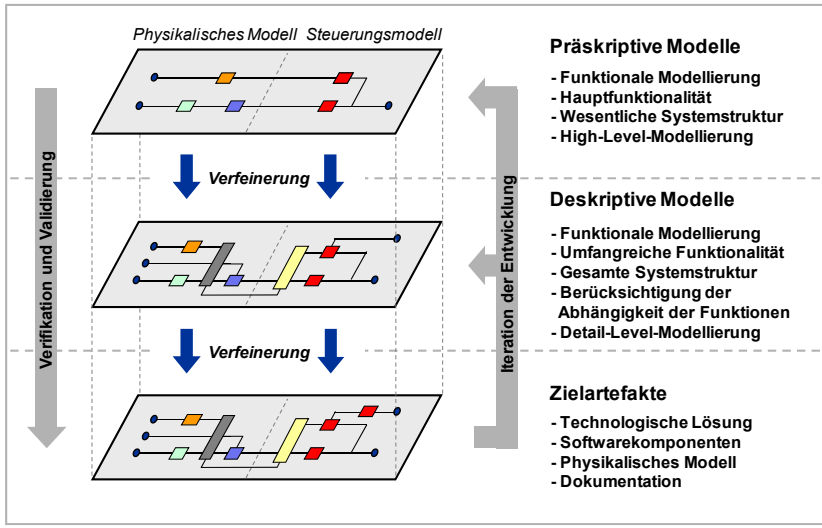


Abbildung 5.7: Abstraktionsebenen bei der Modellbildung

Dennoch sollte auch auf dieser Ebene soweit möglich das Prinzip der funktionalen Modellierung angewandt werden. Dadurch ist eine frühzeitige Detaillierung der Entwicklung bzw. der Modelle möglich, ohne sich bereits auf bestimmte Hardwarekomponenten festlegen zu müssen. Die Fokussierung auf ein funktionales Engineering wird auch durch entsprechende Untersuchungen gestützt. So konnte beispielsweise gezeigt werden, dass 80 bis 90 Prozent der elektrischen Funktionalität eines Fertigungssystems als Hardware-unabhängig betrachtet werden kann *(Herkommer 2003)*.

Für die *Erstellung der Zielartefakte*[4] ist hingegen eine detaillierte Beachtung der technologischen Umsetzung und folglich die Verfeinerung und Ergänzung der Modelle um spezifische Eigenschaften (siehe Abschnitt 5.3) erforderlich. Beispielsweise gilt es, die zugrunde liegende Hardware des Systems festzulegen und die Parameter und Eigenschaften einzelner Bausteine den daraus resultierenden Gegebenheiten anzupassen. Ebenso sind den Modellelementen die notwendigen Informationen zuzuweisen, die für eine weitestgehend automatisierte Verarbeitung zu Softwarebausteinen oder die Generierung von Simulationsmodellen für bestimmte Zielsysteme notwendig sind. Abschließende manuelle Ergänzungen sind ebenfalls der genannten Abstraktionsebene zuzuordnen.

[4] Unter Zielartefakten sind die Endergebnisse der Entwicklung (Dokumentation, Programme, Funktionsbausteine, Funktionen, Simulationsmodelle, usw.) zu verstehen. Sie werden häufig auch als Implementierungsmodelle bezeichnet.

5.5 Auswahl einer domänenspezifischen Modellierungstechnik

5.5.1 Vorgehen zur Entwicklung einer DSL

Voraussetzung zur Modellbildung ist die Bereitstellung einer geeigneten Technik, die den Anforderungen des Anwendungsgebietes entspricht und die notwendigen Beschreibungskonstrukte definiert.

Abbildung 5.8 zeigt zum Zweck der genaueren Klärung und Einordnung des Begriffs in Anlehnung an *(Stahl & Voelter 2005, S. 66)* den Aufbau einer domänenspezifischen Modellierungssprache (DSL).

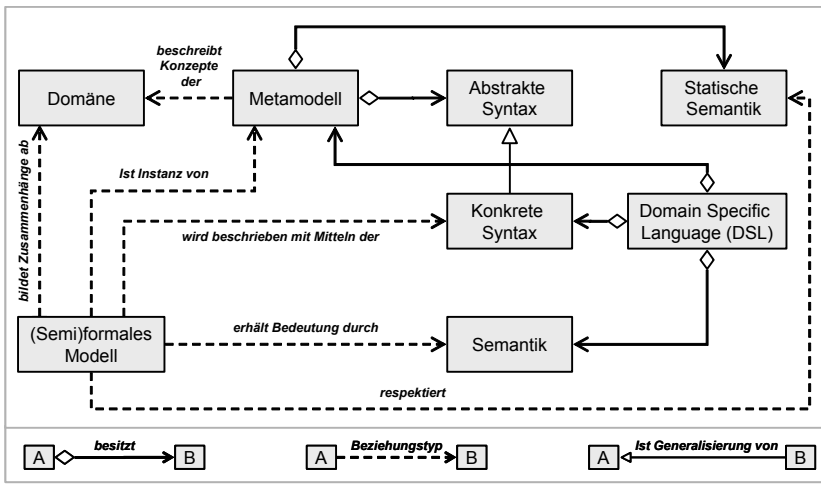


Abbildung 5.8: Aufbau einer Domain Specific Language

Demnach wird eine DSL durch ein Metamodell, eine konkrete Syntax und eine (dynamische) Semantik beschrieben. Das Metamodell legt in abstrakter Form fest, wie die Modelle der betrachteten Domäne aufgebaut sind. Dazu definiert es eine abstrakte Syntax und eine statische Semantik. Während die abstrakte Syntax bestimmt, welche elementaren Modellierungselemente existieren und verwendet werden können, beschreibt die statische Semantik, wie die Beziehungen zwischen diesen zu gestalten sind und welche Randbedingungen beachtet werden müssen. Bei einem Signalinterpretierten Petri-Netz (siehe Abschnitt 2.3.3) legt die statische Semantik beispielsweise unter anderem fest, dass Eingangssignale nur an Transitionen gebunden werden dürfen, während Ausgangssignale den Stellen des Netzes zuzuordnen sind. Die konkrete Syntax ist als die Realisierung der abstrakten Syntax zu betrachten. Sie beschreibt wie deren Umsetzung in Bezug auf die Modellierungssprache aussieht. Für das genannte Beispiel definiert sie unter anderem, dass Stellen als Kreise dargestellt werden, wäh-

rend Transitionen durch schmale Balken zu symbolisieren sind. Die dynamische Semantik hingegen gibt den einzelnen Konstrukten der DSL und damit den erstellten Modellen eine Bedeutung. So kann beispielsweise eine Stelle in einem Signalinterpretierten Petri-Netz als die Zuordnung definierter Werte zu den Zustandsgrößen des modellierten Systems aufgefasst werden.

Um sinnvolle und gültige Modelle bilden zu können, muss die Modelliererin bzw. der Modellierer folglich die ihm zur Verfügung stehenden Sprachmittel verstehen und einordnen können. Die Semantik muss entweder gut dokumentiert oder intuitiv klar sein. Dies wird dadurch erleichtert, dass die DSL bekannte Konzepte der Domäne aufgreift, sodass eine Expertin bzw. ein Experte sich in ihrer/seiner Sprach- und Begriffswelt wiederfindet. Des Weiteren sind aber auch der Zweck der Modellbildung und die grundlegenden Eigenschaften der betrachteten technischen Systeme zu berücksichtigen.

Im Folgenden wird daher, als Basis zur Auswahl einer domänenspezifischen Modellierungssprache für die modellgetriebene Softwareentwicklung im zugrunde liegenden Betrachtungsbereich, eine systemtechnische Einordnung automatisierter Fertigungssysteme vorgenommen. Daran schließt sich eine Bewertung bestehender Beschreibungsmittel für technische Systeme, insbesondere automatisierter Anlagen, an. Ziel ist die Selektion prinzipiell geeigneter Methoden, die auf bestehenden Standards basieren, und deren Kombination zu einem durchgängigen Gesamtkonzept. Darauf aufbauend kann die Semantik, die Syntax und das Metamodell der DSL festgelegt werden.

5.5.2 Systemtechnische Einordnung automatisierter Fertigungssysteme

Automatisierte Fertigungssysteme sind als komplexe technische Systeme mit mechatronischem Charakter zu betrachten. Aufgrund der funktionalen, jedoch nicht räumlichen Integration erfolgt im Rahmen der Modellbildung eine Aufteilung in ein Steuerungsmodell und in ein physikalisches Modell. Ersteres entspricht in der praktischen Umsetzung der Speicherprogrammierbaren Steuerung. Das physikalische Teilmodell wird im Wesentlichen durch das mechanische Grundsystem mit zusätzlichen elektrischen, elektromechanischen, pneumatischen und hydraulischen Komponenten realisiert. In Bezug auf den Modellzweck dient das Steuerungsmodell dazu, die Funktionalität der Steuerungsprogramme zu definieren und auszuarbeiten. Das physikalische Teilmodell legt die Randbedingungen fest, an welche die Softwarefunktionalität anzupassen ist. Zum anderen ist dieses im Zuge der Modellbildung aber auch soweit zu detaillieren, dass dadurch eine simulationsgestützte Verifikation und Validierung der entwickelten Programme möglich wird. Unter Beachtung des Modellzweckes und der generellen Eigenschaften können die beschriebenen Teilsysteme klassifiziert werden (siehe Abbildung 5.9).

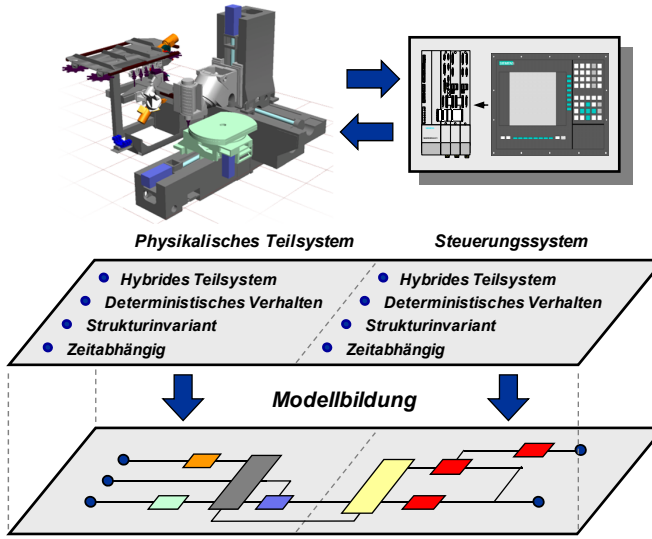


Abbildung 5.9: Systemtechnische Einordnung automatisierter Fertigungssysteme

Eine Analyse des Aufbaus und der Arbeitsweise Speicherprogrammierbarer Steuerungen zeigt, dass die Struktur der Programme als invariant zu betrachten ist. Während bei konventionellen IT-Systemen typischerweise strukturelle Objekte zur Laufzeit erzeugt und wieder gelöscht werden, erfolgt bei der SPS beim Hochlauf das Laden aller Bausteine und Funktionen in den Arbeits- bzw. Systemspeicher. Die Modifikation von Variablen (Merkern[5] oder Datenbausteinen) und Ausgangssignalen oder das Ausführen bestimmter Funktionen werden durch das Ändern von Eingangssignalen oder internen Zustandsgrößen ausgelöst. Dem entsprechend kann das Systemverhalten als ereignisdiskret charakterisiert werden. Diese Betrachtung ist jedoch nicht vollständig richtig, da in der Praxis durchaus ein zeitabhängiges Verhalten zu implizieren ist. Ursache hierfür ist neben der Verarbeitung quasikontinuierlicher Größen[6] die weit verbreitete Anwendung von Timern, wodurch trotz der zyklischen Arbeitsweise Speicherprogrammierbarer Steuerungen ein direkter Zeitbezug vorhanden ist.

Das Verhalten Speicherprogrammierbarer Steuerungen ist weiterhin als deterministisch zu betrachten. Zwar kann bei der konventionellen Programmierung trotz der eindeutigen Definition der Reihenfolge der Signalverarbeitung *(DIN EN 61131-3)* durch-

[5] Einfache Variablen werden bei Speicherprogrammierbaren Steuerungen als Merker bezeichnet.

[6] Die Verarbeitung quasikontinuierlicher Größen wird auch als Wortverarbeitung bezeichnet.

aus ein nichtdeterministisches Verhalten auftreten. Dies ist jedoch im Sinne der Entwicklung eines vorhersehbaren Systemverhaltens zwingend zu vermeiden.

Die Steuerungsprogramme eines automatisierten Fertigungssystems sind üblicherweise sowohl als statisch wie auch dynamisch zu klassifizieren. Verknüpfungssteuerungen, beispielsweise zur Realisierung sicherheitsrelevanter Aufgaben, oder Funktionen (FC) zur Ausführung von Berechnungen besitzen häufig keine Zustandsgrößen und erzeugen bei gleicher Eingangsbelegung ein eindeutiges Ausgangsbild. Demzufolge ist ihr Verhalten als statisch einzuordnen. Bei Ablaufsteuerungen hingegen ist das Verhalten in Bezug auf eine definierte Eingangsbelegung aufgrund der zwangsgesteuerten Ausführung auch vom aktuellen Zustand abhängig und somit als dynamisch zu betrachten.

In einem ersten Schritt zur systemtechnischen Klassifikation des physikalischen Teilsystems ist im Hinblick auf den Modellzweck zu bestimmen, wie detailliert das Systemverhalten abzubilden ist. Unter Berücksichtigung der zyklischen Arbeitsweise der Steuerung und der Zykluszeit, die bei üblichen Fertigungssystemen bei ca. 25 bis 100 Millisekunden liegt, ist es aus Sicht der Steuerung und vor dem Hintergrund der Intention des simulationsgestützten Softwaretests hinreichend, die Schnittstelle zwischen beiden Teilsystemen sowie das logische Verhalten und das Zeitverhalten des physikalischen Teilsystems zu modellieren. Hierzu würden zeitabhängige, ereignisdiskrete Beschreibungsmittel ausreichend sein. Unter Beachtung des Prinzips der Signalflussorientierung sowie der Tatsache, dass die in Fertigungssystemen verbauten Komponenten (siehe Abbildung 5.10[7]) auf der Basis physikalischer Gesetze mathematisch sehr gut abbildbar sind, wird jedoch deren Einordnung als (Teil-)Systeme mit kontinuierlichem bzw. hybridem Verhalten als geeignet befunden.

Die Beschreibung durch mathematische Zusammenhänge auf der Grundlage physikalischer Gesetzmäßigkeiten impliziert deterministisches Modellverhalten. Dies stellt zwar eine Einschränkung des realen Systemverhaltens dar, das durchaus auch stochastische Eigenschaften aufweisen kann (beispielsweise Bruch einer Leitung oder Reibungseffekte). Diese Vereinfachung ist aber im Sinne der notwendigen Abstraktion bei der Modellbildung durchaus vertretbar. Zur Abbildung stochastischer Einflüsse im Rahmen des Softwaretests wird der Eingriff der Anwenderinnen und Anwender in Form einer Variation der Modellparameter oder des Signalflusses im physikalischen Teilmodell als hinreichend und zweckmäßig erachtet.

[7] Ergebnis einer Untersuchung bei mehreren Herstellern von automatisierten Fertigungssystemen *(Pörnbacher & Wünsch 2004, S. 67)*.

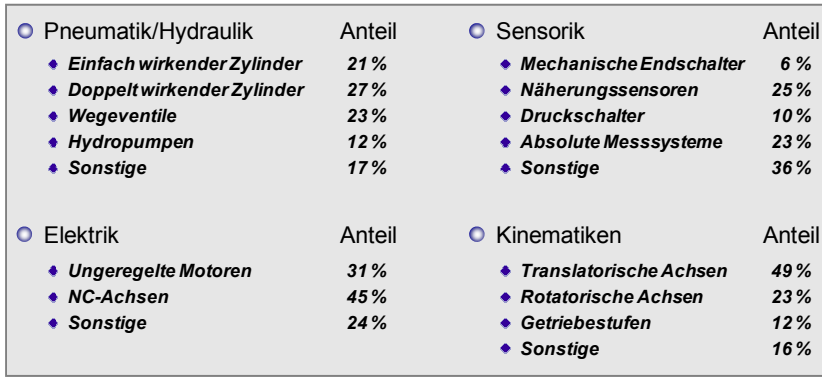


Abbildung 5.10: Komponenten in automatisierten Fertigungssystemen

In Bezug auf den Aufbau des physikalischen Teilsystems ist nicht notwendigerweise von einem strukturinvarianten[8] System auszugehen. Strukturelle Änderungen können beispielsweise durch das dynamische Bestücken eines Palettenwechslers mit Paletten, die über integrierte Spannsysteme und Sensoren verfügen, oder das prozessbedingte Abkoppeln einzelner Teilsysteme auftreten. Derartige Änderungen der Maschinen- und folglich der Modellstruktur sind aber selten und beziehen sich üblicherweise auf den Austausch durch gleichartige oder ähnliche Teilsysteme. Die Betrachtung des physikalischen Teilsystems als strukturinvariant ist daher zulässig, wenn die dynamische Veränderung im physikalischen Modell durch eine entsprechende Beschreibung des Signalflusses in vereinfachter Form berücksichtigt oder im Sinne der Einschränkung des Aufwandes zur Modellbildung bewusst vernachlässigt wird.

5.5.3 Bewertung bestehender Beschreibungsmittel

5.5.3.1 Anforderungen an eine DSL

Für die Evaluierung bestehender Beschreibungsmittel als Basis zur Auswahl bzw. Entwicklung einer domänenspezifischen Modellierungssprache sind die in Abschnitt 3.6.3 beschriebenen technologischen Anforderungen und die dargestellte systemtechnische Klassifikation der Fertigungssysteme als Grundlage zu verwenden. Weitere Voraussetzungen, die vor allem die Einsatztauglichkeit in der industriellen Praxis berücksichtigen, wurden in Zusammenarbeit mit diversen Herstellern automatisierter Fertigungssysteme *(Pörnbacher & Wünsch 2004)* ermittelt.

[8] Unter einem strukturinvarianten System wird ein System verstanden, bei dem einzelne Teilsysteme nicht dynamisch hinzugefügt, entfernt oder ausgetauscht werden können.

Die weitgehende Erfüllung dieser Randbedingungen ist als Prämisse für die prinzipielle Anwendbarkeit eines Beschreibungsmittels zu betrachten. Darüber hinaus gilt es jedoch auch, eine Evaluierung im Hinblick auf deren Eignung für die unterschiedlichen Entwicklungsphasen und dementsprechend die definierten Abstraktionsebenen bei der Modellbildung vorzunehmen. Abbildung 5.11 zeigt eine zusammenfassende Darstellung der wesentlichen zu beachtenden Bewertungskriterien.

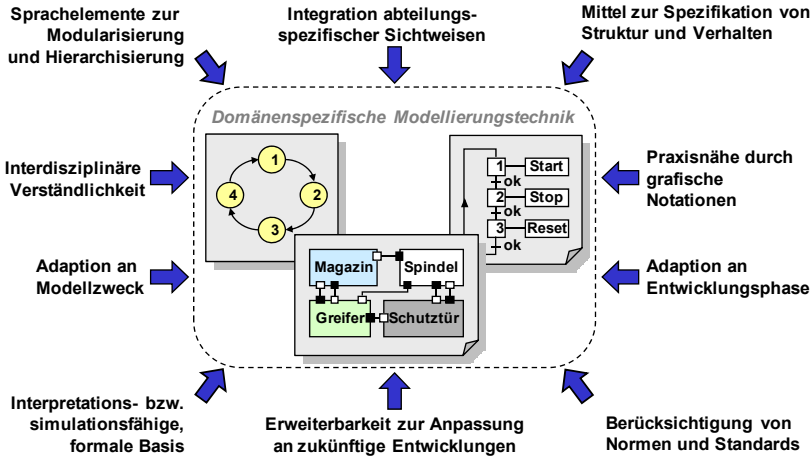


Abbildung 5.11: Anforderungen an eine DSL

Die nachfolgende Klassifikation unterscheidet Beschreibungsmittel zur Spezifikation des Verhaltens und der Struktur sowohl des Softwaresystems wie auch des physikalischen Teilsystems. Aufgrund der Vielfalt der in den letzten Jahren und Jahrzehnten entwickelten Ansätze ist eine vollständige Gegenüberstellung und Beurteilung nicht möglich. Die Auswahl der betrachteten Entwicklungen beruht daher auf einer Fokussierung auf für die Arbeit relevante und praxisnahe Ansätze, die standardisiert oder stark verbreitet sind. In Bezug auf geeignete Ansätze im Umfeld der Automatisierungstechnik werden auch Untersuchungen der GMA[9] *(VDI/VDE 3681)* als Grundlage verwendet. Programmiersprachen nach IEC 61131-3 sind ebenfalls Teil der Bewertung, da diese auch als Beschreibungsmittel verwendet werden können.

5.5.3.2 Bewertung von Beschreibungsmitteln zur Systemspezifikation

Abbildung 5.12 zeigt eine zusammenfassende Gegenüberstellung der ausgewählten Beschreibungsmittel zur Spezifikation technischer Systeme. Die Darstellung versteht sich als eine vergleichende Bewertung in Bezug auf die im Rahmen der Arbeit zu be-

[9] GMA = VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik.

rücksichtigenden Anforderungen und erhebt keinen Anspruch auf Vollständigkeit. Für nähere Ausführungen zu den einzelnen Beschreibungsmitteln wird auf den Abschnitt 2.3 bzw. den Anhang der Arbeit (Abschnitt 11.2) oder entsprechende Fachliteratur verwiesen.

Anforderungen / Beschreibungsmittel		Formale Basis			Verhalten				Struktur		Darstellung			Zeitbasis			Interdisziplinäre Verständlichkeit	Normierung bzw. Standardisierung	Erweiterbarkeit
		Formal	Semiformal	Informal	Deterministisch	Nicht deterministisch	Statisch	Dynamisch	Hierarchisierung	Modularisierung	Textbasiert	Mathematisch - Symbolisch	Grafisch	Ereignisdiskret	Kontinuierlich	Hybrid			
Endliche Automaten mit E/A		●	○	○	●	●	○	●	○	○	●	●	●	●	○	○	◑	○	○
Hybride Automaten		●	○	○	●	●	○	●	○	○	●	●	●	○	○	●	○	○	○
Zeitbewertete Automaten		●	○	○	●	●	○	●	○	○	●	●	●	○	○	●	○	○	○
Message Sequence Charts		○	●	○	●	●	○	●	○	○	●	○	●	●	○	○	◑	●	○
Signalinterpretierte Petri-Netze		●	○	○	●	●	○	●	●	○	○	○	●	●	○	○	○	●	○
Funktionsblockdiagramme		○	●	○	●	○	●	●	●	●	○	○	●	●	○	○	○	●	○
GRAFCET		○	●	○	●	●	○	●	●	○	○	○	●	●	○	○	◑	●	○
IEC 61131-3	Ablaufsprache	◑	○	○	●	○	○	●	●	●	●	○	●	●	○	○	○	●	○
	Funktionsbausteinsprache	●	○	○	●	○	●	●	●	●	○	○	●	●	○	○	○	●	○
	Anweisungsliste	●	○	○	●	○	●	●	●	●	●	○	○	●	○	○	○	●	○
	Kontaktplan	●	○	○	●	○	●	●	●	○	○	○	●	●	○	○	○	●	○
	Strukturierter Text	●	○	○	●	○	●	●	●	○	●	○	○	●	○	○	○	●	○
UML 2.0	Komponentendiagramm	○	●	○	○	○	○	○	●	●	○	○	●	○	○	○	◑	●	●
	Sequenzdiagramm	○	●	○	●	●	○	●	●	●	○	○	●	●	○	◑	◑	●	●
	Zustandsdiagramm	○	●	○	●	●	○	●	●	●	○	○	●	●	○	◑	◑	●	●
	Aktivitätsdiagramm	○	●	○	●	●	●	●	●	●	○	○	●	●	○	◑	◑	●	●
SysML	Anforderungsdiagramm	○	●	○	○	○	○	○	●	●	○	○	●	○	○	○	◑	○	●
	Blockdiagramm	○	●	○	○	○	○	○	●	●	○	○	●	○	○	◑	◑	○	●
HybridUML		○	●	○	●	○	○	●	●	●	○	○	●	○	○	●	○	○	●
Hybrides Dynamisches Netz		○	●	○	●	○	○	●	○	○	○	○	●	○	○	●	○	○	○
Blockschaltbilder		●	○	○	●	○	●	●	●	●	●	●	●	◑	●	●	◑	○	●
Modelica		●	○	○	●	○	●	●	●	●	●	●	●	◑	●	●	●	●	●

● erfüllt ◑ teilweise erfüllt ○ nicht erfüllt

Abbildung 5.12: Bewertung von Beschreibungsmitteln

Es ist festzustellen, dass keiner der betrachteten Ansätze die Anforderungen an eine domänenspezifische Modellierungssprache zur Gänze erfüllt. Während beispielsweise

Programmiersprachen nach IEC 61131-3 über den notwendigen Formalitätsgrad verfügen, haben diese Schwächen in Bezug auf die interdisziplinäre Verständlichkeit und die Abbildung hybriden Systemverhaltens. Des Weiteren definieren sie keine Mechanismen zur Erweiterung bzw. Anpassung des Sprachumfangs an das jeweilige Aufgabengebiet. Auf endlichen Automaten basierende Methoden hingegen sind mathematisch sehr genau spezifiziert und besitzen dem entsprechend eine formale Basis. Nachteilig sind insbesondere ihre Komplexität und damit die mangelnde Praxisnähe sowie die fehlenden Möglichkeiten zur Strukturierung der Modelle. Die in der UML 2.0 definierten Diagrammtypen beispielsweise haben wiederum den Vorteil, dass sie über mächtige Mechanismen zur Adaption des Sprachumfangs verfügen, in der konventionellen Ausführung aber zu umfangreich sind, keine formale Basis besitzen und zur Modellierung hybrider und kontinuierlicher Systeme nur eingeschränkt einsetzbar sind.

Bei einer Beurteilung der Beschreibungsmittel nach ihrer Eignung in Bezug auf unterschiedliche Entwicklungsphasen und folglich als Mittel zur Modellbildung für die in Abschnitt 5.4 definierten Abstraktionsebenen (siehe Abbildung 5.13), ist ebenfalls festzustellen, dass keines für den durchgängigen Einsatz prädestiniert ist.

Beschreibungsmittel / Entwicklungsphase / Abstraktionslevel	Endliche Automaten mit E/A	Hybride Automaten	Zeitbewertete Automaten	Message Sequence Charts	Signalinterpretierte Petri-Netze	Funktionsblockdiagramme	GRAFCET	IEC 61131-3: Ablaufsprache	IEC 61131-3: Funktionsbausteinsprache	IEC 61131-3: Anweisungsliste	IEC 61131-3: Kontaktplan	IEC 61131-3: Strukturierter Text	UML 2.0: Komponentendiagramm	UML 2.0: Sequenzdiagramm	UML 2.0: Zustandsdiagramm	UML 2.0: Aktivitätsdiagramm	SysML: Anforderungsdiagramm	SysML: Blockdiagramm	HybridUML	Hybrides Dynamisches Netz	Blockschaltbilder	Modelica
Aufgabenklärung	○	○	○	○	○	○	◑	○	○	○	○	○	○	◑	○	◑	●	○	○	○	○	○
Systemkonzeption	◑	○	○	◑	○	○	◑	◑	○	○	○	○	◑	●	◑	◑	◑	◑	◑	○	◑	◑
Systementwurf	●	●	●	○	◑	◑	◑	◑	◑	○	○	◑	●	●	●	●	○	◑	●	◑	◑	●
Implementierung	○	○	○	○	●	◑	○	●	●	●	◑	●	◑	○	◑	◑	○	○	◑	○	●	●
Integration und Inbetriebnahme	○	○	○	○	○	◑	◑	●	●	◑	◑	◑	◑	◑	◑	○	○	○	◑	○	◑	◑
Verifikation und Validierung	○	◑	◑	◑	◑	○	○	◑	◑	○	◑	○	○	◑	○	○	○	○	○	○	◑	◑
Präskriptive Modelle	◑	○	○	○	○	○	◑	◑	○	○	○	○	◑	●	◑	◑	◑	◑	◑	○	○	○
Deskriptive Modelle	●	●	●	◑	●	◑	◑	◑	◑	○	○	◑	●	◑	●	●	○	◑	◑	◑	●	●
Implementierungsmodelle	○	○	○	○	●	○	○	●	●	●	◑	◑	○	○	○	○	○	○	◑	○	◑	●

● geeignet ◑ teilweise geeignet ○ nicht geeignet

Abbildung 5.13: Einordnung von Beschreibungsmitteln in die Entwicklung

Einige Ansätze, wie beispielsweise die UML, haben ihre Stärken in den Bereichen der Systemkonzeption und des Systementwurfs. Aufgrund des semiformalen Charakters der einzelnen Diagrammtypen sind diese jedoch für die Implementierungs- und Verifikationsphase nur bedingt geeignet. Andere Methoden verfügen wiederum über Stärken in den späteren Entwicklungsphasen, während für vorhergehende Engineeringschritte nur eine eingeschränkte Einsatzfähigkeit gegeben ist.

Zusammenfassend kann festgestellt werden, dass partikuläre Ansätze als Basis für eine domänenspezifische Modellierungssprache im betrachteten Einsatzbereich keine hinreichende Eignung aufweisen. Eine adäquate Spezifikations- und Modellierungstechnik ist daher nur durch die Kombination vorhandener Methoden zu erreichen. Aufgabe dabei ist zum einen die Auswahl, Ergänzung und Weiterentwicklung entsprechender Techniken. Zum anderen sind die Übergänge zwischen den für die jeweiligen Entwicklungsphasen geeigneten Abstraktionsebenen zu gestalten.

5.5.4 Auswahl von Beschreibungsmitteln für eine Modellierungssprache

5.5.4.1 Übersicht

Im Folgenden wird die Auswahl einzelner Beschreibungsmittel als Basis für eine domänenspezifische Modellierungssprache erklärt und begründet. Des Weiteren erfolgt eine prinzipielle Zuordnung der einzelnen Methoden zu den definierten Abstraktionsebenen bei der Modellbildung und zu den beschriebenen Teilsystemen.

5.5.4.2 Übergeordnete Spezifikation des Systemverhaltens

Für eine relativ abstrakte, dem physikalischen Teilsystem und dem Steuerungssystem übergeordnete Spezifikation des Verhaltens eines Fertigungssystems werden Sequenzdiagramme der UML sowie eine hierarchische Funktionsstruktur als adäquat befunden. Beide sind besonders zur präskriptiven Modellierung der umzusetzenden Funktionen geeignet. Während eine hierarchische Funktionsstrukturierung primär einer Gliederung der Gesamtfunktionalität dient, liegt der Schwerpunkt der Sequenzdiagramme auf der Darstellung der Interaktion zwischen verschiedenen Komponenten, die zur Erfüllung spezifischer Aufgaben notwendig ist. Dezidierte Funktionen der Funktionshierarchie können bestimmten Systemkomponenten zugeordnet werden. Sequenzdiagramme hingegen weisen einen zu speziellen Systembausteinen übergeordneten Charakter auf. Sie eignen sich insbesondere zur Definition von Abläufen, indem sie festlegen, in welcher Reihenfolge bestimmte Funktionen einzelner Komponenten auszuführen sind. Funktionen können auf der präskriptiven Ebene durch zusätzliche Modellelemente, die primär der beschreibenden Modellierung dienen (z. B. Aktivitäten oder Zuweisungen) unterstützt werden. Diese im Folgenden beschriebenen Techniken erlauben eine weitere Präzisierung der geforderten Funktionalität und schlagen somit eine Brücke zur nachfolgenden Ebene der deskriptiven Modelle.

5.5.4.3 Spezifikation des Verhaltens des Steuerungssystems

Für die Spezifikation des Verhaltens des Steuerungssystems werden die Zustands- und Aktivitätsdiagramme der UML (siehe Abschnitt 2.3.2) eingesetzt. Wesentliche Vorteile gegenüber anderen Beschreibungsmitteln sind die ausgeprägten Möglichkeiten zur Einschränkung und Präzisierung des Sprachumfangs im Hinblick auf das jeweilige Einsatzgebiet und somit auch die Anpassbarkeit an zukünftige Entwicklungen in der Automatisierungstechnik. Darüber hinaus sind die graphischen Darstellungen für einen praxisnahen Einsatz von Vorteil. Obwohl sich die genannten Diagramme vor allem für die Spezifikation des Verhaltens ereignisdiskreter Systeme eignen, definieren sie in eingeschränktem Umfang auch Sprachelemente zur Abbildung zeitabhängiger Verhaltensaspekte und verfügen über Konstrukte zur hierarchischen, modularen Gestaltung der Modelle. Die Unzulänglichkeiten in Bezug auf die Formalität und die Spezifikation deterministischen Verhaltens sind durch entsprechende Konkretisierungen des Sprachumfangs zu eliminieren.

Während Sequenzdiagramme das Verhalten einzelner Systemkomponenten nur in Ausschnitten darstellen und sich auf das Notwendige beschränken, eignen sich Zustandsdiagramme und Aktivitätsdiagramme zur vollständigen Definition des Softwareverhaltens. Erstere haben ihre Vorzüge bei der Modellierung von Softwarefunktionalität mit koordinativem, dynamischem Charakter. Aktivitätsdiagramme hingegen besitzen zum einen den Vorteil, dass sich die Funktionalität, die einzelnen Zuständen bzw. Transitionen im Zustandsdiagramm zugeordnet wird, detailliert spezifizieren lässt. Zum anderen weisen Aktivitätsdiagramme auch eine grundsätzliche Eignung zur Definition statischer Softwarefunktionalität auf. Aufgrund der genannten Eigenschaften liegt der Fokus der beiden Diagrammtypen auf der deskriptiven Modellierung. Sie sind somit weitgehend der gestaltenden Phase im Rahmen der Konzeption und dem Systementwurf zuzuordnen.

5.5.4.4 Spezifikation des Verhaltens des physikalischen Teilsystems

Zur Modellierung des Verhaltens des physikalischen Teilsystems werden mathematische Beschreibungsmittel mit Elementen zur Abbildung von Systemeigenschaften mit kontinuierlichem bzw. hybridem Charakter verwendet. Als besonders prädestiniert wird Modelica, als standardisierte Sprache zur Modellierung multiphysikalischer Systeme, erachtet. Diese zeichnet sich gegenüber alternativen Beschreibungsmitteln zum einen durch ihre Unabhängigkeit aufgrund der offengelegten Spezifikation aus, wodurch sich erhebliche Potentiale zur Weiterverwendung der Modelle, beispielsweise zur Erstellung der Elektrodokumentation oder für Wartungs- und Schulungszwecke, ergeben. Darüber hinaus erlaubt Modelica sowohl eine kausale wie auch eine akausale (siehe Abschnitt 11.2.1), objektorientierte Modellbildung, womit eine praxisnahe, an die jeweilige Abstraktionsebene angepasste Erstellung der physikalischen Teilmodelle ermöglicht wird.

Für die funktionale, präskriptive Modellierung ist eine abstrahierte Abbildung der Systemzusammenhänge als adäquat zu betrachten. Die Spezifikation der Funktionen des physikalischen Teilsystems als logische Folge von Ursachen und Wirkungen, ohne eine explizite Betrachtung der physikalischen Hintergründe, ist für diese frühe Phase der Systemkonzeption hinreichend genau und mit geringem Aufwand verbunden. Eine umfassende, an der technologischen Umsetzung orientierte Modellierung kann aufgrund der üblicherweise noch unvollständigen Engineeringdaten in dieser Entwicklungsphase in der Regel auch nicht sinnvoll durchgeführt werden.

Auf der Ebene der deskriptiven Modelle hingegen steht eine Detaillierung und Konkretisierung der entwickelten Prinziplösungen im Vordergrund. Das physikalische Teilsystem ist damit in seiner Funktion grundsätzlich bekannt und kann mathematisch mit der in Bezug auf den Modellzweck notwendigen und hinreichenden Genauigkeit beschrieben werden. Für die Abbildung physikalischer Systeme sind kausale Beschreibungsformen jedoch nur bedingt einsetzbar. Dies ist damit zu begründen, dass derartige Systeme meist nicht als rückwirkungsarm[10] betrachtet werden können und folglich eine praxisnahe, einfache mathematische Beschreibung der Wirkzusammenhänge in gerichteter Form (siehe Abschnitt 11.2.1) nur bedingt möglich ist. Daher wird für diese Phase der Entwicklung eine komponentenorientierte Modellierung des Verhaltens in akausaler Form als geeignet betrachtet. Ist aufgrund des Modellzwecks eine Abstraktion des Verhaltens soweit möglich, dass Rückwirkungseffekte vernachlässigt oder vereinfacht abgebildet werden können, so kann auch eine Verfeinerung der Systemzusammenhänge auf der Grundlage der präskriptiven Modelle hinreichend sein. Die geeignete Form der Modellierung ist von der technologischen Realisierung abhängig, weshalb diesbezüglich keine allgemeingültige Aussage getroffen werden kann.

Die im Rahmen des Systementwurfs erstellten deskriptiven Modelle sind auch für die nachfolgenden Entwicklungsphasen weiter zu verwenden. In Bezug auf die Implementierung betreffen notwendige Verfeinerungen im Wesentlichen die Anpassung von Modellparametern, um das Verhalten der eingesetzten Systemkomponenten an den speziellen Anwendungsfall zu adaptieren. Weitere Präzisierungen können notwendig sein, um das Modell des physikalischen Teilsystems soweit aufzubereiten, dass eine simulationsgestützte Inbetriebnahme des Fertigungssystems oder einzelner Baugruppen möglich ist.

5.5.4.5 Spezifikation der Systemstruktur

Aufgrund des in Abschnitt 5.2 dargestellten systemtheoretischen Ansatzes und der anzuwendenden Prinzipien zur Modellbildung ist für die Abbildung der Systemstruktur ein gemeinsames, für beide Teilmodelle gültiges, übergeordnetes Konzept notwendig.

[10] Als Beispiel sei ein hydraulisches System genannt, bei dem eine Änderung der Belastung an einem Verbraucher Auswirkungen auf alle anderen Verbraucher im System haben kann.

Eine nähere Betrachtung der ausgewählten Beschreibungsmittel zeigt, dass sowohl die UML wie auch Modelica über ähnliche Strukturierungskonzepte verfügen. Daher wird zur Modellierung der Systemstruktur das Komponentendiagramm der UML als prinzipiell geeignet befunden. Trotz der Möglichkeit einer gemeinsamen Notation gilt es jedoch, die fundamental unterschiedlichen Ansätze zur Modellierung des Verhaltens des Steuerungssystems und des physikalischen Teilsystems zu integrieren. Wird beispielsweise aufgrund der Änderung eines Zustandes im Modell der Steuerung ein Ereignis generiert, so ist eindeutig festzulegen, welche Auswirkungen dieses auf das kontinuierliche oder hybride Verhalten des physikalischen Teilsystems hat. Umgekehrt ist klar zu spezifizieren, welche Änderungen im Verhalten des physikalischen Teilsystems zur Generierung von Ereignissen führen, die Einfluss auf das Steuerungsmodell haben. In der konkreten Umsetzung bedeutet dies, dass die aus der Theorie hybrider Systeme bekannten Konzepte von Quantisierer und Injektor (siehe Abschnitt 11.2.3) bei der Detaillierung einer domänenspezifischen Modellierungssprache berücksichtigt werden müssen. Im Hinblick auf einen praxisnahen und verständlichen Ansatz sollte sich deren Abbildung jedoch nicht in zusätzlichen Modellkomponenten bemerkbar machen, sondern durch bereits bestehende, geeignete Sprachelemente zur Verhaltensmodellierung abgedeckt werden.

5.5.5 Unterstützung durch Visualisierung

Mit den dargestellten Modellierungsmethoden kann die Struktur und das Verhalten automatisierter Fertigungssysteme hinreichend genau spezifiziert werden. Dies allein reicht jedoch für einen interdisziplinären Entwicklungsansatz nicht aus. Modelle müssen verständlich, einleuchtend und erlebbar sein und die interaktive Auseinandersetzung ermöglichen *(Geisberger 2005, S. 35)*. Trotz der graphischen Darstellungsform der ausgewählten Beschreibungsmittel fällt es dem Entwicklungspersonal häufig schwer, ein korrektes Verständnis für das Verhalten und insbesondere die Dynamik des spezifizierten Systems zu entwickeln. Dieses Defizit ist vor allem auf den statischen Charakter der Diagrammtypen zurückzuführen *(Bock & Zühlke 2006)* und wird durch die unterschiedliche Sichtweise und den jeweiligen Ausbildungshintergrund der beteiligten Personen verstärkt.

Als besonders geeignet für die Unterstützung einer interdisziplinären Entwicklung automatisierter Fertigungssysteme wird die Integration der im Rahmen der mechanischen Konstruktion erstellten dreidimensionalen Darstellungen in die Modelle erachtet *(Pörnbacher & Wünsch 2004, S. 23)*. Durch die anschauliche grafische Repräsentation werden, im Gegensatz zu einer zweidimensionalen oder textbasierten Beschreibung, viele Zusammenhänge besser verständlich. Eine simulationsgestützte Ausführung des statischen und dynamischen Verhaltens der Modelle und dessen dreidimensionale Visualisierung erleichtert die Diskussion der Entwicklung im interdisziplinären Team

oder mit dem Kunden und trägt somit entscheidend zur Entwicklung eines gemeinsamen Systemverständnisses bei *(Schwindt & Landgraf 2003)*.

5.6 Simulationsmethoden zur Verifikation und Validierung

5.6.1 Übersicht

Die Integration von Simulationsmethoden in den Entwicklungsprozess ist als eine wesentliche Anforderung an einen Ansatz zur modellgetriebenen Softwareentwicklung zu betrachten. Im Folgenden werden daher unterschiedliche Methoden zur Verifikation und Validierung der Modelle und der entwickelten Steuerungssoftware dargestellt und in Bezug auf ihre Eignung für die definierten Abstraktionsebenen bei der Modellbildung bewertet.

5.6.2 Prinzipielle Verfahren zur Systemsimulation

Für die Simulation technischer Systeme existieren eine Vielzahl von Möglichkeiten und Konzepten, die an das jeweilige Anwendungsgebiet, die eingesetzte Technologie und den Simulationszweck angepasst sind. Aufgrund des Fokus der Arbeit und der zu berücksichtigenden Rahmenbedingungen sind die nachfolgend beschriebenen, fundamentalen Ansätze als relevant zu betrachten.

Model-in-the-Loop-Simulation

Bei der Model-in-the-Loop-Simulation (siehe Abbildung 5.14) werden das dynamische und statische Verhalten des Steuerungsmodells wie auch des physikalischen Modells durch einen geeigneten Simulator verarbeitet. Die Ausführung wird entsprechend dem Aufbau des Simulators durch unterschiedliche Methoden realisiert.

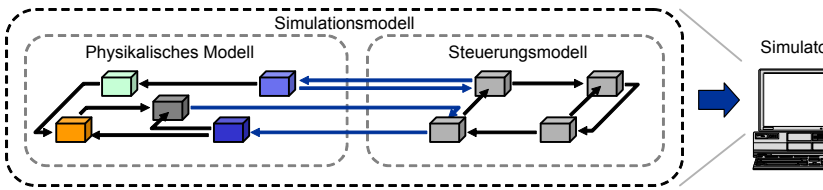


Abbildung 5.14: Model-in-the-Loop-Simulation

Beim Verfahren der *Modellinterpretation* wird das Modell zur Laufzeit durch den Simulator ausgewertet und interpretiert. Als nachteilig erweist sich dabei die Verarbeitung der Daten zur Laufzeit, wodurch ein hoher Rechenaufwand entsteht. Bei der *Modellcompilation* hingegen wird aus den Modellen in einem oder mehreren Schritten ein ablauffähiges Programm erzeugt. Hierzu muss entweder ein geeigneter Compiler entwickelt werden oder es erfolgt in einem Zwischenschritt die Transformation in eine

gebräuchliche Hochsprache, die eine problemlose Weiterverarbeitung der Modelldaten gestattet.

Software-in-the-Loop-Simulation

Die Software-in-the-Loop-Simulation (siehe Abbildung 5.15) verzichtet im Gegensatz zum zuvor genannten Verfahren auf eine Interpretation des Steuerungsmodells. Während das Modell des physikalischen Teilsystems weiterhin durch einen Simulator interpretiert oder in Form eines ablauffähigen Programms ausgeführt wird, wird das Steuerungsmodell durch die entwickelte Steuerungssoftware und eine Softwareemulation[11] der Hardware und des Betriebssystems der Steuerung ersetzt.

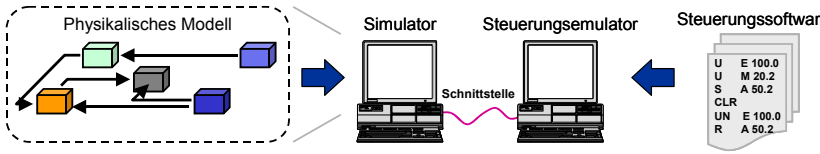


Abbildung 5.15: Software-in-the-Loop-Simulation

Beide Programme werden jedoch auf demselben oder miteinander gekoppelten Rechnern ausgeführt. Meist erfolgt die konkrete Umsetzung unter Nutzung des Verfahrens der Co-Simulation[12].

Hardware-in-the-Loop-Simulation

Bei der Hardware-in-the-Loop-Simulation (siehe Abbildung 5.16) wird das Steuerungsmodell durch die Steuerungshardware und die entwickelte Software substituiert. Teilweise werden auch Komponenten des physikalischen Teilmodells durch die entsprechende Maschinenhardware ersetzt.

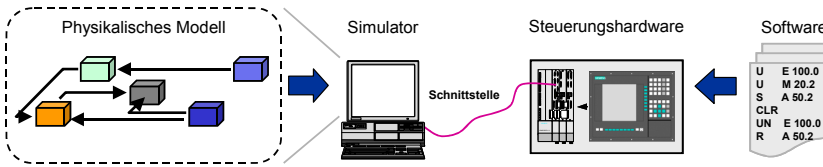


Abbildung 5.16: Hardware-in-the-Loop-Simulation

[11] Eine Emulation ist das funktionelle Nachbilden eines Systems durch ein Ersatzsystem. Ein Softwareemulator ist ein Programm, das die Funktionalität eines anderen Systems (Software, Hardware oder deren Kombination) nachbildet.

[12] Die Co-Simulation ist ein Verfahren, bei dem unterschiedliche Simulatoren/Emulatoren miteinander gekoppelt werden. Dies erfolgt unter Nutzung einer gemeinsamen, eindeutig definierten Schnittstelle und durch die Synchronisation der Simulatoren/Emulatoren zur Laufzeit.

Aufgrund der Notwendigkeit der Kommunikation zwischen der Steuerung und dem Simulator muss dieser mit geeigneter Zusatzhardware ausgestattet werden. Üblicherweise sind dies PC-Einsteckkarten, die sowohl über eine physikalisch vorhandene Schnittstelle (beispielsweise ein Feldbussystem) die Kommunikation mit der Steuerungshardware abwickeln als auch die interne Verbindung zum Simulator implementieren.

5.6.3 Bewertung und Einordnung der Verfahren

Abbildung 5.17 zeigt eine zusammenfassende Bewertung der dargestellten Ansätze zur Simulation in Bezug auf wirtschaftliche und technologische Faktoren.

Eigenschaften / Simulationsverfahren	Geringer Anpassungs- und Modellierungsaufwand	Abbildung des realen Zeitverhaltens	Integration der vollständigen Steuerungs- und Bedienfunktionalität	Einbindung realer Hardwarekomponenten	Modelle hoher Komplexität	Sicherstellung der Akzeptanz bei den Anwenderinnen bzw. Anwendern	Hohe Simulationsgüte	Geringer Investitionsaufwand	Unabhängigkeit vom Zielsystem	Entwicklungsphase: Systemkonzeption	Entwicklungsphase: Systementwurf	Entwicklungsphase: Implementierung	Entwicklungsphase: Integration und Inbetriebnahme
Model-in-the-Loop-Simulation	○	○	○	○	●	◑	◑	●	●	●	●	○	○
Software-in-the-Loop-Simulation	◑	○	◑	○	◑	◑	◑	●	◑	○	◑	●	◑
Hardware-in-the-Loop-Simulation	●	●	●	●	○	●	●	◑	◑	○	○	◑	●

● geeignet/erfüllt ◑ teilweise geeignet/erfüllt ○ nicht geeignet/erfüllt

Abbildung 5.17: Bewertung der Simulationsverfahren

Ein besonderer Vorteil des Hardware-in-the-Loop-Ansatzes ist zum einen der geringe Aufwand zur Anpassung und Erstellung der Modelle. Dies ist darauf zurückzuführen, dass bei alternativen Ansätzen auch das Verhalten der Steuerungshardware mit zu berücksichtigen ist und abgebildet werden muss. Wird hingegen die im Rahmen der Projektierung festgelegte Hardware verwendet, so fällt der entsprechende Aufwand nicht an. Des Weiteren wird es möglich, das Zeitverhalten der Steuerung direkt und realitätsnah umzusetzen und somit das effektive Verhalten der automatisierten Fertigungssysteme weitestgehend der Realität entsprechend abzubilden, wodurch eine hohe Simulationsgüte erreicht werden kann. Die Integration der Steuerungshardware, also der Speicherprogrammierbaren Steuerungen und entsprechender Schnittstellen zum Bedienpersonal, ermöglicht den Zugriff auf die volle Steuerungs- und Bedienfunktionalität, während bei Steuerungsemulatoren oder Simulatoren aufgrund des Aufwandes diesbezüglich häufig Abstriche gemacht werden müssen. Zudem besteht die Gefahr, dass das Verhalten entsprechender Baugruppen nicht hinreichend genau oder fehler-

haft implementiert wird. In der Folge wird auch eine wesentlich höhere Akzeptanz bei den Anwenderinnen und Anwendern eines entsprechenden Simulationssystems sichergestellt *(Pörnbacher & Wünsch 2004, S. 64)*, da diese die Schnittstelle benutzten, die im Anschluss an die Entwicklung auch an der realen Anlage zur Verfügung steht.

Ein weiterer Vorteil des Hardware-in-the-Loop-Ansatzes besteht in der Möglichkeit reale Komponenten in die Simulation mit einzubinden. Damit existiert die Option, Bauteile und Baugruppen, deren Modellierung aus wirtschaftlichen oder technologischen Gründen nicht umsetzbar ist, zu integrieren und Modellkomponenten im Zuge der Montage bzw. Inbetriebnahme schrittweise durch die realen Systemkomponenten zu ersetzen.

Als nachteilig erweist sich zum einen die Notwendigkeit, Speicherprogrammierbare Steuerungen und Benutzerschnittstellen zur Verfügung zu stellen, womit ein gewisser Investitionsaufwand verbunden ist. Da die Hersteller der Maschinen und Anlagen meist nur bestimmte, ausgewählte Komponenten in ihren Produkten einsetzen und somit üblicherweise vorrätig haben und da die verwendete Steuerungshardware später direkt in die Fertigungssysteme integriert werden kann, ist dieser Mehraufwand nur bedingt von Bedeutung. Zum anderen wird die mögliche Komplexität der Simulationsmodelle durch den beschriebenen Ansatz eingeschränkt. Die Kommunikation zwischen Steuerung und Simulator macht es notwendig, das Verhalten des physikalischen Teilsystems mit einer hinreichenden Geschwindigkeit zu simulieren. Aufgrund begrenzter Rechnerleistungen kann diese Anforderung nur dann erfüllt werden, wenn die Simulationsmodelle eine eingeschränkte Komplexität aufweisen.

Der Model-in-the-Loop- bzw. der Software-in-the-Loop-Ansatz besitzen als wesentlichen Vorteil die weitgehende Unabhängigkeit von spezifischen Hardwarekomponenten und die Möglichkeit zur Simulation komplexer Systemmodelle. Entsprechende Simulatoren oder Emulatoren sind üblicherweise generisch aufgebaut und unabhängig von spezifischen Eigenschaften und Sonderfunktionen dezidierter Hardwarekomponenten. Dies schränkt zwar die Simulationsgüte ein und kann auch Anpassungen an der Steuerungssoftware oder den Modellen (beispielsweise eine Adaption der Adressbereiche oder Ausblenden bestimmter hardwarenaher Funktionalitäten) zur Folge haben, ist jedoch bezüglich des Investitionsaufwands als günstiger zu bewerten. Insbesondere der möglichen Komplexität der einsetzbaren Modelle des physikalischen Teilsystems sind bei diesen Ansätzen nur eingeschränkt Grenzen gesetzt, wenn die Simulationsdurchführung unter Anwendung der Methode der „virtuellen Echtzeit“ erfolgt. Diese beruht darauf, dass die verwendeten Simulatoren bzw. Emulatoren nicht auf die reale Zeit als Grundlage zur Simulation und Synchronisation zurückgreifen, sondern auf einer virtuellen Zeit basieren, die entsprechend der benötigten Rechenleistung beliebig angepasst werden kann. Als nachteilig sind hingegen die geringere Akzeptanz bei den Anwenderinnen bzw. Anwendern und die fehlenden Möglichkeiten zur Integration realer Maschinenkomponenten zu betrachten.

Unter Berücksichtigung der genannten Eigenschaften wird die Model-in-the-Loop-Simulation für die frühen Entwicklungsphasen als adäquat bewertet. Zur Simulation der präskriptiven Modelle ist angesichts des Reifegrades der Entwicklung ein alternativer Ansatz nicht sinnvoll möglich. Die Einschränkungen bezüglich der erreichbaren Simulationsgüte sind aufgrund des Modellzweckes (Spezifikation und Diskussion der wesentlichen Systemfunktionalität auf abstraktem Niveau und Festlegung logischer Abläufe zu deren Umsetzung) nicht als relevant zu betrachten, zumal auch detaillierte Informationen zur technologischen Umsetzung der Steuerungsaufgaben und der Maschinenfunktionen nicht verfügbar sind und im Rahmen der Entwicklung erst noch festgelegt werden müssen. Weitestgehend ist der genannte Simulationsansatz auch für die deskriptive Modellbildung als prädestiniert zu betrachten.

Die Methode der Software-in-the-Loop-Simulation ist für einen Test der generierten Software unter Nutzung des physikalischen Teilmodells von Bedeutung. Dies gilt insbesondere für einen Vorab-Test einzelner Softwaremodule, die unabhängig von spezifischer Funktionalität der Steuerungshardware sind. Der Vorteil liegt dabei in den besseren Möglichkeiten zum Freischneiden einzelner Module und der guten Eignung für die Anwendung einer Bottom-up Teststrategie, wie sie in der industriellen Praxis üblich ist.

Die Hardware-in-the-Loop-Simulation ist für die (Vorab-)Inbetriebnahme als besonders geeignet zu bewerten, da die Simulationsgüte sehr gut an das Verhalten der realen Fertigungssysteme angenähert werden kann. Durch die Verwendung der eingesetzten Steuerungen und der Mensch-Maschine-Schnittstellen wird ein hoher Immersionsgrad für das Test- und Inbetriebnahmepersonal sichergestellt. Die genannten Rahmenbedingungen erlauben es weiterhin, einen Großteil der Systemparameter (z. B. Laufzeitüberwachungen) einzustellen. Die Integration von realen Systemkomponenten und Simulationsmodellen ist insbesondere für die Einbindung des Verhaltens nicht zur Verfügung stehender Zulieferkomponenten oder nicht simulierbarer Bauteile und Baugruppen von Vorteil.

5.7 Integration der Modelldaten in den Entwicklungsprozess

Für eine effiziente Umsetzung der modellgetriebenen Softwareentwicklung ist deren Integration in die weiteren Engineeringprozesse zwingend erforderlich. Nur durch den Zugriff auf aktuelle Entwicklungsdaten der verschiedenen Fachbereiche und die Nutzung dieser Informationen kann eine zielorientierte Systemmodellierung sichergestellt werden. Gleichzeitig ist es erforderlich, die im Rahmen der Softwareentwicklung neu generierten Informationen für andere Aufgaben zur Verfügung zu stellen. Derartige Anforderungen lassen sich mit bislang bestehenden Engineeringlösungen und Entwicklungswerkzeugen jedoch nur beschränkt umsetzen. Bedingt wird dies zum einen durch die organisatorischen Defizite der Geschäftsprozesse bei der Maschinenentwicklung (siehe Abschnitt 3.5.2), zum anderen aber auch durch proprietäre Datenformate,

die einen Informationsaustausch und eine Parallelisierung der Entwicklung im Sinne des Concurrent Engineering wesentlich erschweren. Die daraus resultierende Mehrfachgenerierung von Entwicklungsdaten ist mit hohem Zusatzaufwand verbunden, sie ist fehleranfällig und sie führt zu Missverständnissen, Fehlinterpretationen und Inkonsistenzen. Notwendig sind daher neue Konzepte zum Management der Entwicklungsdaten und zu deren Integration.

Ein für die modellgetriebene Entwicklung automatisierter Fertigungssysteme geeigneter Ansatz wird in *(Englberger u. a. 2003; Zäh & Lercher 2004; Zäh & Lercher 2006)* vorgestellt. Die unter dem Begriff der „Virtuellen Werkzeugmaschine“ beschriebene Engineeringplattform (siehe Abbildung 5.18) ermöglicht, basierend auf einem gemeinsamen Datenmodell, die unternehmensweite Integration und konsistente Verwaltung der Entwicklungsinformationen.

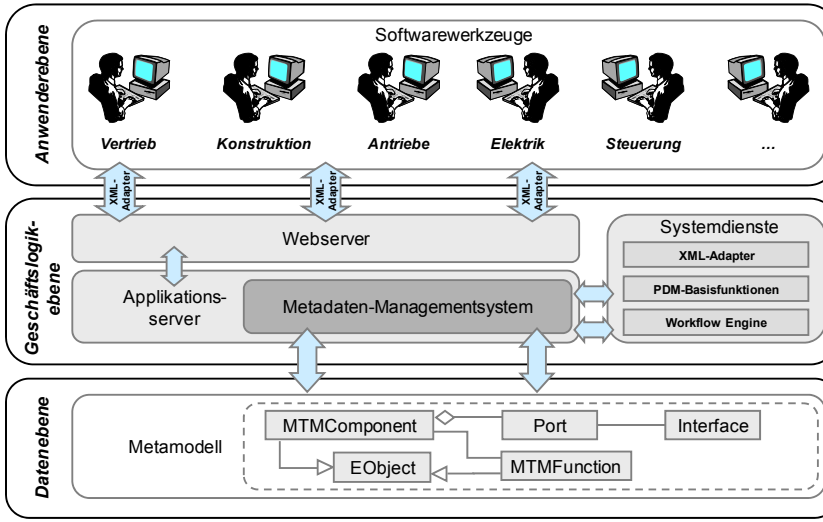


Abbildung 5.18: Die „Virtuelle Werkzeugmaschine“ als Entwicklungsplattform

Im Gegensatz zu herkömmlichen Produktdaten-Management-Systemen (PDM-Systemen) und interdisziplinären Datenmodellen, wie beispielsweise MechaSTEP[13] *(PAS 1013)*, beruht der Ansatz auf einer Fokussierung auf fachbereichsübergreifend relevante Zusammenhänge und Modellinformationen. Diese werden formal anhand eines Metamodells beschrieben und bilden auf der Datenebene den zentralen Bestandteil der in Form einer Client-Server-Architektur ausgeführten Engineeringplattform. Auf der übergeordneten Geschäftslogikebene erfolgt die Anbindung der verschiede-

[13] MechaSTEP = STEP data model for simulation data of mechatronic systems.

nen, auf der Anwenderebene eingesetzten Softwarewerkzeuge. Hierzu werden sogenannte XML-Adapter verwendet, deren Funktion darin besteht, die unterschiedlichen Datenformate der einzelnen Engineeringwerkzeuge zu analysieren und in eine einheitliche Form zu übersetzen, damit diese durch das zentrale Datenmodell verarbeitet werden können.

Zusätzlich verfügt das System über Dienste, die PDM-Basisfunktionalitäten, wie beispielsweise die Definition unternehmensspezifischer Standardabläufe, eine Versionsverwaltung oder das Ein- und Auschecken von Dokumenten, implementieren. Die Einbindung neuer Softwarewerkzeuge auf Anwenderebene und die Spezifikation der Abhängigkeiten zwischen den Daten erfolgt durch ein Metadaten-Management-System. Nähere Ausführungen hierzu finden sich in *(Lercher 2008)*.

5.8 Zusammenfassung

In Kapitel 5 wurde ein Rahmenkonzept zur modellgetriebenen Softwareentwicklung für automatisierte Fertigungssysteme beschrieben. Die Ausführungen umfassen sowohl die Darstellung eines hierzu geeigneten, prinzipiellen Ansatzes und die Vorstellung einer für das Fachgebiet adäquaten Vorgehensweise. Darüber hinaus wurden mehrere, an die jeweilige Entwicklungsphase angepasste, Abstraktionsebenen zur Modellbildung definiert und hierzu prinzipiell geeignete Beschreibungsmittel ausgewählt. Die Ausführungen schließen mit der Erläuterung geeigneter Simulationsmethoden und deren Einordnung in den Prozess der Softwareentwicklung sowie einer Diskussion bezüglich der Integration des Ansatzes in weitere Geschäftsprozesse der Entwicklung. Eine detaillierte Vorstellung der zur Modellbildung notwendigen domänenspezifischen Sprache ist folgendem Kapitel zu entnehmen.

6 Domänenspezifische Technik zur Systemmodellierung

6.1 Allgemeine Rahmenbedingungen zur Sprachdefinition

In den folgenden Abschnitten wird auf der Grundlage der in Abschnitt 5.5.4 ausgewählten Beschreibungsmittel eine domänenspezifische Modellierungssprache vorgeschlagen. Konkret bedeutet dies, dass die einzelnen Notationselemente in ihrer Syntax und Semantik im Detail beschrieben werden. Darüber hinaus wird, entsprechend dem Aufbau einer Modellierungssprache (siehe Abbildung 5.8), auch der semantische Zusammenhang zwischen den Modellkonstrukten in Form eines Metamodells spezifiziert. Dabei sind neben den in Abschnitt 5.5.3.1 genannten Anforderungen an eine DSL folgende Randbedingungen zu beachten:

- Im Sinne eines praxistauglichen Ansatzes ist die Vielfalt der möglichen Modellelemente einzuschränken. Gerade bei der UML besteht durch die Fülle der verfügbaren Sprachelemente und Diagrammtypen die Gefahr, dass ein Einsatz im betrachteten Umfeld an der Komplexität scheitert. Diese wird daher auch als ein wesentlicher Nachteil der UML betrachtet *(Broy & Rumpe 2007)*. Dennoch muss der Sprachumfang die für das Anwendungsgebiet notwendigen Modellierungsmittel umfassen.
- Dabei ist auch auf die erforderliche Formalität zu achten. Informale Modelle sind zwar zur Unterstützung der Systementwicklung durchaus als geeignet anzusehen *(Vogel-Heuser & Friedrich 2006)*. Sie erlauben aber keine eindeutige Interpretation und dem entsprechend keine rechnergestützte Auswertung und Weiterverarbeitung.
- Die vorteilhaften Konzepte der Objektorientierung, wie beispielsweise die Datenkapselung und die Methodendefinition, sind nicht Bestandteil der IEC 61131-3 *(Hess 2005)*. Folglich kann die Steuerungssoftware auch nicht objektorientiert erstellt werden. Dies schließt jedoch nicht die Modelle mit ein. Bei der Sprachspezifikation sind daher objektorientierte Techniken mit zu berücksichtigen.
- Im Rahmen der Sprachdefinition werden die in Abschnitt 2.3.2 beschriebenen Mechanismen (Stereotypen, Tagged Values, Object Constraint Language) zur Adaption an das Anwendungsgebiet (Tailoring) genutzt. Entsprechende Modellelemente werden mit dem Präfix[1] „DSL“ gekennzeichnet. Sind diese nur für das Steuerungsmodell oder das physikalische Teilmodell relevant, erfolgt eine Bezeichnung mit den Präfixen „SM“ bzw. „PM“.

Als Basis für eine effiziente und zielgerichtete Modellbildung und eine stringente Modellverfeinerung ist weiterhin eine Orientierung am domänenspezifischen Problemraum notwendig. Modellgetriebene Entwicklungsansätze fordern daher als Grundlage für die Erarbeitung einer DSL eine ausführliche Analyse einer Referenzimplementie-

[1] Die Präfixe werden lediglich im Rahmen der Sprachdefinition verwendet.

rung *(Stahl & Voelter 2005, S. 16)*. Aufgrund der Individualität automatisierter Fertigungssysteme und ihrer Steuerungsprogramme ist diese Vorgehensweise für die betrachtete Domäne nicht zwingend als adäquat zu betrachten. Vielmehr wird die Verwendung der in Abschnitt 5.5.2 dargestellten systemtechnischen Einordnung solcher Systeme, eine darüber hinausgehende Analyse der prinzipiellen Sprachkonstrukte der IEC 61131-3 sowie eine problemorientierte Untersuchung des Aufbaus der Maschinen und Anlagen als sinnvoll angesehen.

In Bezug auf die nachfolgenden Ausführungen zur domänenspezifischen Modellierungssprache wird des Weiteren darauf hingewiesen, dass diese keinen Anspruch auf Vollständigkeit erhebt. Die Beschränkung auf die dargelegten Sprachkonstrukte beruht auf einer Analyse des Aufbaus und der SPS-Software mehrerer automatisierter Fertigungssysteme und auf Erfahrungen und Untersuchungen bei der Entwicklung neuer Anlagen beziehungsweise dezidierter Teilsysteme. Die in Abschnitt 2.3.2 beschriebenen Erweiterungsmechanismen der UML erlauben jedoch jederzeit eine Erweiterung oder Ergänzung des Sprachumfangs.

Im Hinblick auf das physikalische Teilsystem ist zu erwähnen, dass Modelica bereits alle zur mathematischen und damit formalen Beschreibung notwendigen Konstrukte zur Verfügung stellt. Die Aufgabe besteht daher darin, den Umfang der Sprachelemente einzuschränken, um eine kompakte Spezifikation des Systemverhaltens mit geringem Aufwand, jedoch hinreichender Genauigkeit, zu ermöglichen. Weiterhin ist die Modellbildung durch die Einführung graphischer Hilfsmittel zu vereinfachen. Darüber hinaus gilt es, die verfügbaren Modellelemente beider Sprachen zu vergleichen und soweit möglich auf ein gemeinsames, generisches Sprachkonstrukt abzubilden.

6.2 Grundlegende Sprachelemente

6.2.1 Übersicht

Für die Beschreibung der Struktur und des Verhaltens automatisierter Fertigungssysteme werden mehrere fundamentale Basiskonstrukte benötigt. Von Relevanz sind Datentypen, Elemente zur Formulierung von Ausdrücken und Einschränkungen sowie informale Sprachelemente.

6.2.2 Datentypen

Obwohl die UML bereits einige einfache Datentypen (Boolean, Integer, String) festlegt, sind diese für die vorliegende Aufgabenstellung unzureichend. Zum einen werden nichtordinale Datentypen (Real) durch die Sprache nicht unterstützt. Zum anderen definiert die IEC 61131-3 aufgrund ihres hardwarenahen Anwendungsbereichs eine Reihe zusätzlicher und problemspezifischer Datentypen, die bei der Modellbildung beziehungsweise der Modellverfeinerung zu verwenden sind. Abbildung 6.1 zeigt den Aus-

schnitt eines Klassendiagramms[2], das die für die DSL erlaubten Ausprägungen spezifiziert. Bezüglich einer vollständigen Darstellung wird auf Abschnitt 11.4.1 verwiesen.

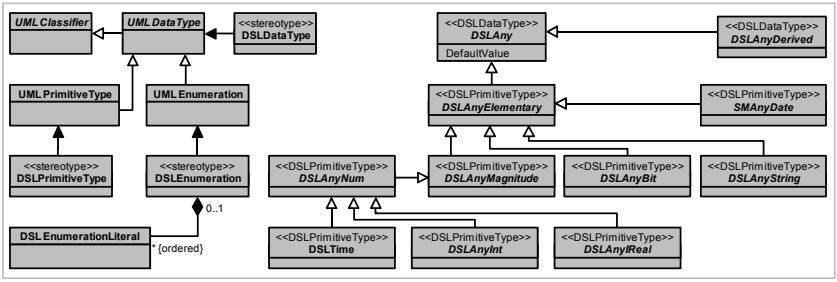


Abbildung 6.1: Ausschnitt aus dem Metamodell der Datentypen

Prinzipiell werden elementare und abgeleitete Datentypen unterschieden. Letztgenannte können dabei eine Kombination der Ersteren sein oder diese einschränken (z. B. den Wertebereich). Die DSL umfasst alle in *(DIN EN 61131-3)* festgelegten elementaren Datentypen, spezifiziert einen sprachspezifischen Default-Initialisierungwert *„Default-Value“* und erlaubt deren Kombination zu komplexeren Strukturen. Implizit definiert sind damit auch die für die Modellierungssprache Modelica *(Modelica 2005)* festgelegten Elementartypen (Real, Integer, Boolean, String, Enumeration) und deren Aggregation zu komplexen Objekten. Als eine Besonderheit des beschriebenen Ansatzes ist der Datentyp *„DSLTime“* zu betrachten, der die Basis für die Modellierung zeitabhängiger Systeme darstellt.

6.2.3 Ausdrücke und Einschränkungen

Ausdrücke (Expressions) sind zur Beschreibung mathematischer Zusammenhänge und zur Auswertung von Variablen notwendig. Sie werden in Form einer Zeichenkette dargestellt, müssen jedoch so formuliert sein, dass sie eindeutig interpretierbar sind. Ausdrücke setzen sich üblicherweise aus Operatoren und Operanden zusammen. Während Operanden feste Werte oder Variablen eines bestimmten Datentyps sein können, definieren Operatoren die Regeln zum Auswerten der Operanden. Zu berücksichtigen ist dabei, dass die Semantik eines Ausdruckes nicht eindeutig ist, weshalb eine weitere Präzisierung notwendig ist.

Die für die domänenspezifische Modellierungssprache relevanten Klassen von Ausdrücken können Abbildung 6.2 entnommen werden. Demnach wird als Erweiterung

[2] Ein Klassendiagramm ist ebenfalls ein Diagrammtyp der UML, dessen Fokus auf der Darstellung prinzipieller Abhängigkeiten in der Struktur eines Systems und der Beziehungen einzelner Bausteine zueinander liegt. Im Gegensatz zum Komponentendiagramm beschreibt es keine konkreten Ausprägungen, sondern wesentliche Zusammenhänge auf einem abstrakten Niveau.

der UML die Unterklasse *„DSLExpression“* eingeführt. Eine weitere Präzisierung definiert die Stereotypen *„PMConditionalExpression“* und *„DSLMathExpression“*. Ersterer fungiert als Grundlage zur Definition von Konstrukten zur Ausführungssteuerung oder zur Formulierung eingeschränkt gültiger Modellbestandteile. Darunter sind insbesondere bedingt auszuwertende oder alternativ gültige Verhaltensbeschreibungen bei der Modellbildung des physikalischen Teilsystems zu verstehen.

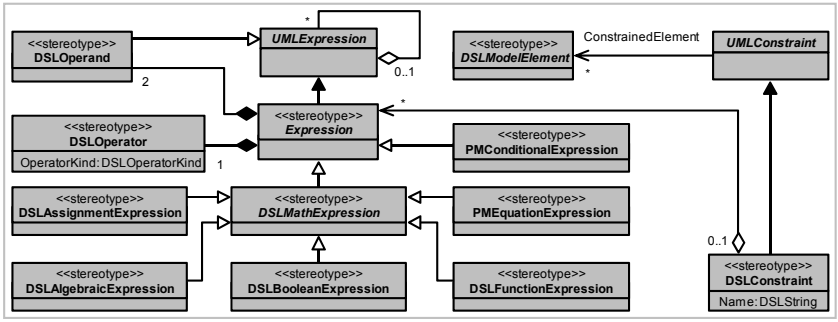


Abbildung 6.2: Metamodell der Expressions und Constraints

Für den abstrakten Stereotyp *„DSLMathExpression“* werden mehrere Verfeinerungen definiert. Ausdrücke vom Typ *„DSLAlgebraicExpression“* werden zur Spezifikation einfacher algebraischer Zusammenhänge benutzt. Damit beschränken sich die verwendbaren Operatoren auf die Grundrechenarten $(-,+,\times,\div)$. Als Operanden sind lediglich Zahlenwerte oder Variablen eines geeigneten Datentyps möglich. Komplexere funktionale Zusammenhänge definiert der Stereotyp *„DSLFunctionalExpression“*. Er erlaubt als Operatoren die in Modelica bzw. der IEC 61131-3 festgelegten Möglichkeiten. Als Beispiele seien die grundlegenden trigonometrischen Funktionen oder Exponentialfunktionen genannt. Eine Darstellung der für das Steuerungsmodell bzw. das physikalische Teilmodell zulässigen Funktionen ist Abschnitt 11.4.2 zu entnehmen. Wie bei allen mathematischen Ausdrücken ist auch bei den funktionalen Ausdrücken bezüglich der Operanden auf Typkonformität zu achten. Die entsprechende Überprüfung ist jedoch durch das verwendete Entwicklungswerkzeug sicherzustellen.

Das Modellelement *„DSLBooleanExpression“* hingegen bildet die Basis zum Aufbau logischer Ausdrücke. Derartige Konstrukte werden beispielsweise zur Formulierung von Bedingungen für die Ausführungssteuerung benötigt. Operatoren sind daher logische Basisfunktionen $(\wedge,\vee,\neg)$ und Relationsoperatoren $(\leq,<,\equiv,\neq,>,\geq)$.

Als gesonderte Ausdrücke mit einem speziellen, festgelegten Operator werden die Stereotypen *„DSLAssignmentExpression“* und *„PMEquationExpression“* eingeführt. Ersterer ist für beide Teilmodelle gültig. Er definiert mit dem Zuweisungsoperator, dass das Ergebnis einer Berechnung, ein fester Zahlenwert oder der Wert einer Variab-

len, einer anderen Variablen zugewiesen wird. Der zweitgenannte Ausdruck dient zur Modellierung von Gleichungen. Der entsprechende Operator legt nicht fest, dass ein Berechnungsergebnis einer bestimmten Variablen zugeordnet werden muss, sondern formuliert als Randbedingung lediglich, dass der Ausdruck der linken Seite dem Ausdruck der rechten Seite entsprechen muss. Da Speicherprogrammierbare Steuerungen in Ermangelung geeigneter Lösungsroutinen Gleichungssysteme nicht verarbeiten können, besteht die Restriktion, dass diese Art von Ausdrücken nur auf das physikalische Teilsystem anwendbar ist.

Einschränkungen (Constraints) sind den Ausdrücken syntaktisch sehr ähnlich. Der Unterschied liegt in der Semantik begründet. Sie werden bestimmten Modellelementen zugeordnet und formulieren Randbedingungen, die für diese gelten bzw. durch diese erfüllt werden müssen. Zur Beschreibung dieser Restriktionen können (Boolesche) Ausdrücke verwendet werden. Es ist prinzipiell auch eine Beschreibung in natürlichsprachiger Form möglich, aus Formalitätsgründen sollten hierzu jedoch speziell vorgesehene, informale Sprachelemente genutzt werden.

6.2.4 Informale Sprachelemente

Trotz der Notwendigkeit einer (semi-)formalen Modellspezifikation sind ergänzende informale Sprachelemente (siehe Abbildung 6.3) sinnvoll, um die Verständlichkeit zu erhöhen und formal nur aufwendig spezifizierbare, weiterführende und erklärende Zusatzinformationen in die Modelle mit einzubinden. Der Sprachumfang umfasst daher den abstrakten Stereotyp *„DSLAnnotation“* mit den konkreten Ausprägungen *„DSLComment“* und *„DSLLink“*.

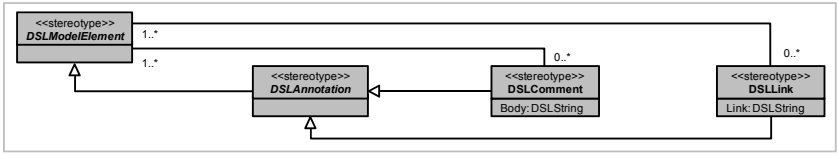


Abbildung 6.3: Metamodell informaler Sprachelemente

Beide Elemente verfügen über ein Attribut in Form eines Strings und können ein oder mehrere Modellelemente referenzieren. Ersterer dient zur näheren Erläuterung von Sachverhalten in Bezug auf das verwiesene Modellelement in Form von Kommentaren. Der Stereotyp *„DSLLink“* hingegen definiert einen Verweis auf ergänzende Informationen, die außerhalb des Modells dokumentiert sind. Beispiele für derartige, supplementäre Angaben sind Bilder, technische Zeichnungen und Skizzen, Handbücher, Webseiten oder Pflichtenhefte.

Bezüglich der Semantik der genannten Sprachelemente ist festzuhalten, dass diese keine zusätzlichen Einschränkungen zu den referenzierten Modellelementen darstellen.

Da die UML für die Stereotypisierung die Definition eigener Notationen erlaubt *(Jeckle u. a. 2004, S. 94)*, wird für die Darstellung eine reine Textform vorgesehen. Weitere toolspezifische Repräsentationsoptionen werden zugelassen, sind jedoch nicht Teil der Sprachdefinition.

6.3 Sprachelemente zur Spezifikation der Systemstruktur

6.3.1 Übersicht

Aufgrund seiner generischen Eigenschaften wird zur Modellierung der Struktur der Fertigungssysteme das Komponentendiagramm der UML als prädestiniert betrachtet. Demzufolge basieren die anzuwendenden Sprachelemente im Wesentlichen auf den im Standard *(OMG 2005a)* spezifizierten Konstrukten. Als Erweiterungen im Hinblick auf den hybriden Charakter der Teilsysteme werden jedoch Ergänzungen auf der Grundlage von Bestandteilen der Systems Modeling Language *(OMG 2007)* hinzugefügt. Ferner sind Sprachelemente vorgesehen, die speziell auf die Anforderungen der IEC 61131-3 und den Aufbau des physikalischen Teilsystems ausgerichtet sind. Nicht als zielführend bewertete Elemente der genannten Beschreibungsmittel sind mit Bezug auf eine kompakte Definition explizit nicht Bestandteil der domänenspezifischen Modellierungssprache.

Die Sprachkonstrukte und Notationen zur Beschreibung der Struktur automatisierter Fertigungssysteme können im Wesentlichen in Elemente zur Spezifikation des hierarchischen Aufbaus (Komponenten, Variablen, Ports, Schnittstellen) sowie Wirkbeziehungen (Konnektoren) unterteilt werden. Eine weitere Differenzierung betrifft generisch anzuwendende Bestandteile und speziell für das physikalische Teilsystem oder das Steuerungssystem geeignete Sprachelemente.

6.3.2 Variablen

Variablen (siehe Abbildung 6.4) werden syntaktisch in Form einer Zeichenkette deklariert und dienen der Beschreibung der Eigenschaften einzelner Bausteine des modellierten Systems. Die häufig auch als Attribute bezeichneten Elemente entsprechen damit aus systemtechnischer Sicht den Zustandsgrößen und verfügen über eine eindeutige Bezeichnung *„VariableName“* und einen für die DSL definierten Datentyp.

Darüber hinaus können noch einige zusätzliche Eigenschaften deklariert werden. Zum einen ist die Sichtbarkeit festzulegen. Ist eine Variable vom Typ *„DSLPRIVATE“*, so ist ein Zugriff auf diese nur durch den Systembaustein möglich, dem sie zugeordnet wurde. Attribute vom Typ *„DSLPUBLIC“* (Default-Wert) hingegen können auch von anderen Modellbausteinen referenziert werden. Eine weitere Konkretisierung betrifft die Einschränkung der Änderbarkeit. Wird zusätzlich zur Sichtbarkeit der Parameter *„DSLCONSTANT“* angegeben, so ist die Variable zwar lesbar, darf aber nicht verändert werden. Der alternative Wert *„PMPARAMETER“* schränkt weniger stark ein und

betrifft nur das physikalische Teilsystem. Er legt fest, dass die entsprechende Variable zur Laufzeit des Systems nicht modifizierbar ist. Wird zusätzlich zur Deklaration noch ein Initialisierungswert *„InitialValue“* angegeben, so gilt dieser unabhängig von dem für den jeweiligen Datentyp vorgegebenen Default-Wert.

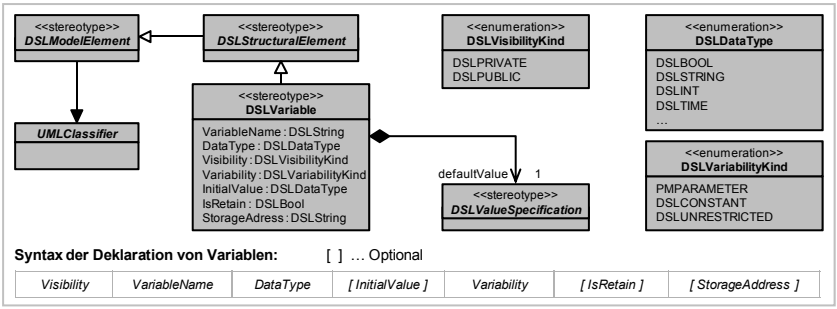


Abbildung 6.4: Metamodell und Syntax der Variablen

Weitere optionale Angaben für das Steuerungssystem sind die Eigenschaften *„IsRetain“* und *„StorageAddress“*. Ersterer ist für sicherheitskritische Bausteineigenschaften relevant. Wird als Wert *„TRUE“* angegeben, so bleibt der aktuelle Wert auch nach einem Ausfall der Stromversorgung der Steuerung erhalten. Dies wird üblicherweise dadurch sichergestellt, dass die entsprechende Variable in einem (hardwarespezifisch) festgelegten, nichtflüchtigen Speicherbereich abgelegt wird. Letzteres Attribut erlaubt die Vorgabe bestimmter Speicheradressen innerhalb der SPS. Für beide Teilsysteme wird weiterhin mit der impliziten Variablen *„Time“* vom Typ *„DSLTIME“* eine globale Zeitdefinition festgelegt, die bereits Teil des Sprachumfangs von Modelica ist *(Modelica 2005)*.

6.3.3 Komponenten und Konnektoren

Komponenten sind die zentralen Bausteine zur Modellierung der Struktur automatisierter Fertigungssysteme. Sie kapseln die Zustandsgrößen und das Verhalten partikulärer Subsysteme und entsprechen damit einzelnen Baueinheiten bzw. Baugruppen oder Softwarebausteinen des automatisierten Fertigungssystems. Konnektoren hingegen verbinden die Ein- und Ausgangsgrößen der Modellbausteine und bilden aus systemtechnischer Sicht die Wirkverbindungen zwischen den einzelnen Komponenten. Sie dienen folglich zum Aufbau der Wirkstruktur im modellierten System.

Abbildung 6.5 zeigt das Metamodell der für die domänenspezifische Modellierungssprache definierten Konstrukte. Demnach wird eine Komponente *„DSLComponent“* als Stereotyp einer UML-Komponente definiert. Sie verfügt über eine eindeutige Bezeichnung *„ComponentName“* und einen eindeutigen Namensraum *„NameSpace“*,

der auch für interne Elemente gültig ist. Dadurch ist es möglich, Komponenten mit der gleichen Bezeichnung mehrfach zu verwenden.

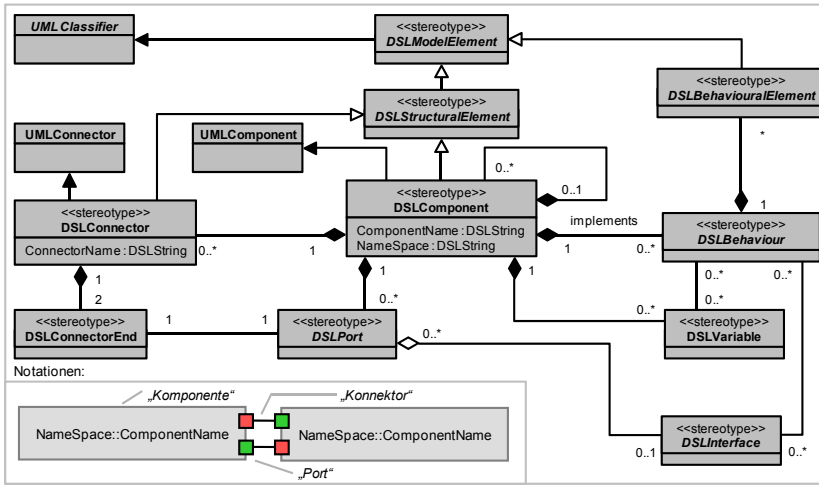


Abbildung 6.5: Metamodell der Komponenten und Konnektoren

Der Namensraum dient sinnvollerweise zur Aggregation zusammengehörender Einheiten des physikalischen Teilmodells oder des Steuerungsmodells. Im ersteren können dadurch mehrere Teilsysteme des gleichen Typs modelliert werden, im Steuerungsmodell legt der Namensraum die Verteilung der Software auf bestimmte Ressourcen[3] fest. Die interne Struktur einer Komponente besteht aus einer endlichen Menge von Variablen *„DSLVariable“*, Konnektoren *„DSLConnector“* und Ports *„DSLPort“*. Sie kann zusätzlich wiederum Komponenten beinhalten, wodurch der Aufbau hierarchischer Systemmodelle (Komposition von Subsystemen) ermöglicht wird. Eine Komponente implementiert ein Verhalten *„DSLBehavior“*, das Bestandteil der Komponente selbst ist und mit den Variablen und Ports (über Schnittstellen) verknüpft ist. Die Spezifikation des Verhaltens erfolgt unter Nutzung hierfür geeigneter Modellelemente (siehe Abschnitt 6.4). Die Kommunikation zwischen einer Komponente und ihrer Umgebung findet ausschließlich über Ports statt. Diese stellen somit definierte Interaktionspunkte zwischen einer Komponente und ihrer Umgebung bzw. ihrem Verhalten dar. Durch diese Kapselung wird eine modulare Einheit gebildet, die es erlaubt, Teilmodelle durch andere Teilmodelle mit gleichen Schnittstellen und gleicher oder erweiterter Funktionalität zu ersetzen.

[3] Unter einer Ressource ist nach IEC 61131-3 eine Speicherprogrammierbare Steuerung zu verstehen. Eine Ressource kann hardwarespezifisch mehrere Tasks bearbeiten. Diese sind als ein inhärentes Element einer SPS zu betrachten, das einzelne Programm-Organisationseinheiten gruppiert und deren Ausführung steuert.

Die Notation einer Komponente geschieht in graphischer Form als ein Rechteck mit durchgezogener Linie, das sowohl den Komponentennamen wie auch den Namensraum beinhaltet (siehe Abbildung 6.5). Weitere, hierarchisch untergeordnete Komponenten werden im Sinne einer „White-Box"-Darstellung graphisch ineinander verschachtelt abgebildet. Optional ist es zulässig, ein Symbol darzustellen, das die Zugehörigkeit der Komponente zum physikalischen Teilmodell oder zum Steuerungsmodell kennzeichnet oder ihren Zweck plastisch darstellt.

Speziell die IEC 61131-3 definiert einige elementare Strukturbausteine, die aus systemtechnischer Sicht eine vordefinierte Struktur, einen internen Zustand und ein festgelegtes Verhalten besitzen. Beispiele sind Zählerbausteine, Zeitglieder (Timer), Speicherbausteine (FlipFlops) oder Flankenerkennungen. Da die graphische Darstellung durch derartige Komponenten unnötig unübersichtlich würde und diese nur für die Spezifikation des Verhaltens komplexerer, übergeordneter Systembausteine relevant sind, kann auf deren explizite graphische Notation im Rahmen der Strukturspezifikation verzichtet werden. Die Verwendung derartiger Elemente zum Zwecke der Verhaltensmodellierung ist Abschnitt 6.4 zu entnehmen.

Konnektoren werden als durchgezogene Linien dargestellt (siehe Abbildung 6.5) und verbinden die Ports einzelner Komponenten. Eine optionale Angabe eines Namens *„ConnectorName"*, der die Rolle der Verbindung näher spezifiziert, ist möglich. Im Gegensatz zur UML wird jedoch eine Verknüpfung zwischen Komponenten und Konnektoren nicht erlaubt. Die Anbindung an die Ports erfolgt indirekt über ein Element des Typs *„DSLConnectorEnd"*, das keine graphische Repräsentation besitzt.

6.3.4 Ports und Schnittstellen

Ein Port definiert einen Interaktionspunkt einer Komponente mit ihrer Umgebung oder zwischen einer Komponente und ihrem Verhalten bzw. internen Bestandteilen. Über diesen werden festgelegte Dienste oder Informationen zur Verfügung gestellt oder von anderen Komponenten angefordert. Als Beispiele seien der Aufruf bestimmter Funktionen eines Softwarebausteins oder der Zugriff auf Variablen dieses Bausteins genannt. Die Art der Kommunikation wird anhand einer oder mehrerer Schnittstellen (Interfaces) definiert, die dem Port zugeordnet sind. Ein Interface stellt eine Beschreibungsform dar, die im Detail festlegt, welche Form die Interaktion hat. Deren Implementierung ist jedoch durch das den Komponenten zugeordnete Verhalten *„DSLBehavior"* zu realisieren. Betrachtet man Schnittstellen und Ports aus systemtechnischer Sicht, so beschreiben diese die Ein- und Ausgangsgrößen der einzelnen Bausteine des modellierten Systems.

Aufgrund der unterschiedlichen Eigenschaften der abzubildenden Teilsysteme ist eine Spezialisierung der genannten Sprachelemente notwendig. Abbildung 6.6 zeigt explizit

die zur Beschreibung von Ports und Schnittstellen erforderlichen Modellkonstrukte. Demnach werden zwei unterschiedliche Subtypen von Schnittstellen unterschieden.

Der Typ *„DSLSignalInterface“* dient zur Spezifikation des zeitkontinuierlichen Energie-, Stoff- und Datenflusses (Signalfluss) im modellierten System. Die Festlegung erfolgt ausschließlich durch die Angabe einer endlichen Menge von Eigenschaften *„DSLSignal“*, die den zur mathematisch-physikalischen Beschreibung des Signalflusses hinreichenden Größen (Signale[4]) entsprechen. Beispielsweise kann anhand dieser Form der Schnittstellenspezifikation ein hydraulischer Anschluss eines Schaltventils durch die Angabe der Signale „Systemdruck“ und „Volumenstrom“ in vereinfachter Form deklariert werden. Zur Festlegung der Kausalität wird jede Größe durch den Tagged Value *„Causality“* ergänzt. Eingehende Signale erhalten das Schlüsselwort *„DSLIN“*, ausgehende das Schlüsselwort *„DSLOUT“*. Signale, die im Sinne der Erfüllung physikalischer Gesetzmäßigkeiten in beide Richtungen wirken und dem entsprechend keine Kausalität definieren[5], wird das Schlüsselwort *„DSLACAUSAL“* zugeordnet. Das Attribut *„IsFlow“* legt weiterhin fest, ob ein Signal eine physikalische Größe beschreibt, die durch eine Flussrichtung gekennzeichnet ist[6].

Der Typ *„SMOperationInterface“* wird hingegen zur Beschreibung komplexerer ereignisdiskreter Kommunikationsfunktionen[7] genutzt. Dazu werden Operationen *„SMOperation“* deklariert, welche die entsprechende Funktionalität formal spezifizieren. Dies geschieht anhand der Vergabe eines aussagekräftigen Namens *„OperationName“* sowie einer geordneten Menge von kausalen Parametern *„DSLParameter“*, die als Übergabe- bzw. Rückgabewerte dienen. In Konformität zum objektorientierten Prinzip der Datenkapselung ist damit nur ein indirekter Zugriff auf die internen Größen einer Komponente und deren Manipulation möglich. Zur Festlegung, ob eine Operation einer Funktion entspricht, die von der implementierenden Komponente angeboten oder benötigt wird, dient das Attribut *„InterfaceKind“*. Verfügt eine Schnittstelle über das Schlüsselwort *„SMPROVIDED“*, so realisiert die Komponente die spezifizierten Operationen und stellt sie anderen Komponenten über den besitzenden Port zur Verfügung. Im alternativen Fall werden die Operationen durch die Komponente zur Erfüllung ihrer Funktion benötigt. Eine Sonderstellung nehmen aufgrund der häufigen

[4] Signale sind im Sinne der Schnittstellendefinition als Verweise auf Variablen der Komponente vom Typ *„DSLPUBLIC“* zu betrachten. Die referenzierten Variablen der über Konnektoren verbundenen Signale weisen zur selben Zeit jeweils den selben Wert auf. Die Definition eines Signals entspricht damit nicht der Signaldefinition der UML.

[5] Derartige physikalische Größen sind beispielsweise Potentialgrößen, wie etwa der Öldruck in einem hydraulischen System.

[6] Größen mit einer Flussrichtung stellen im Allgemeinen Ströme dar, wie beispielsweise der Volumenstrom im genannten Beispiel.

[7] Eine elementare Beschreibung mit Signalen ist für die Spezifikation komplexer Kommunikationsfunktionen, wie sie durch Softwaresysteme realisiert werden, ungeeignet.

Anwendung Operationen zum Lesen und Schreiben von Variablen ein. Bei deren Notation wird anstelle des Operationsnamens ein vordefiniertes Präfix (z. B. R bzw. S) oder Symbol verwendet. Als Parametername wird der Name der referenzierten Variablen angegeben.

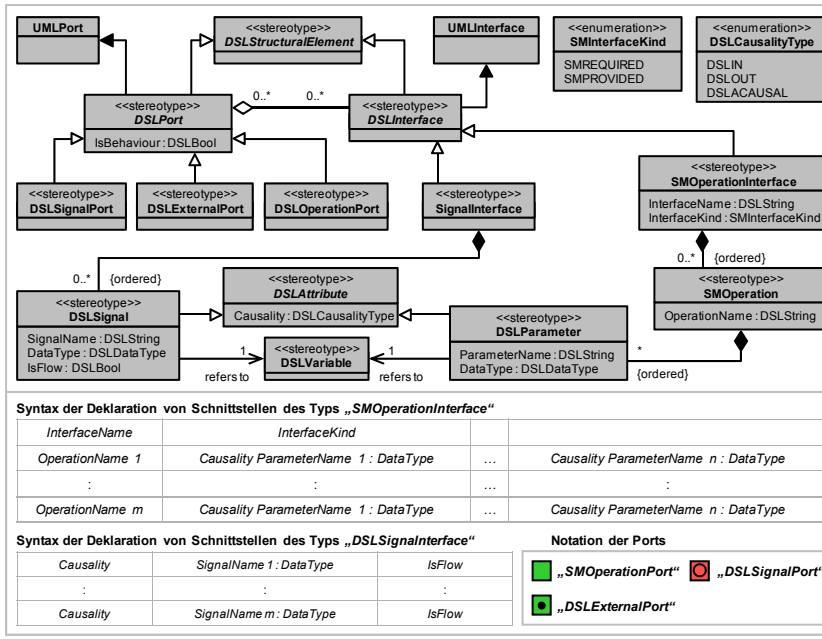


Abbildung 6.6: Metamodell, Syntax und Notation der Ports und Schnittstellen

Die Angabe der Kausalität des einzig zulässigen Parameters ist davon abhängig, ob die Schnittstelle dem Port der aufrufenden oder aufgerufenen Komponente zugeordnet ist. Für die aufgerufene Komponente (deren Teil auch die referenzierte Variable ist) wird die Kausalität, als ausgehend festgelegt, sofern lesender Zugriff[8] erfolgt. Dagegen wird im Falle des schreibenden Zugriffs eine eingehende Kausalität definiert.

Ein Port verfügt mit dem Attribut *„IsBehavior“* über eine Eigenschaft, die seine Aufgabe näher spezifiziert. Hat dieses den Wert *„TRUE“*, so interagiert der Port mit dem Verhalten der Komponente, deren Teil er ist. Im Alternativfall wird der Port lediglich als Durchgangspunkt (Durchgangsport) zur Verknüpfung von internen mit übergeord-

[8] Beim lesenden Zugriff wird aus semantischer Sicht der aktuelle Wert durch die Komponente angeboten. Im Alternativfall wird eine Funktion benötigt, die eine Änderung des Wertes vornimmt.

neten Komponenten genutzt. Weiterhin werden in Analogie zu den Schnittstellen mehrere Subtypen unterschieden.

Die Ausprägung *„DSLSignalPort“* dient zur Modellierung zeitkontinuierlicher Signalschnittstellen. Daher müssen die diesem Porttyp zugeordneten Interfaces vom Typ *„DSLSignalInterface“* sein. Einsatz finden Signalports vor allem bei der Modellierung des physikalischen Teilsystems. Dies ist darin begründet, dass Modelica die Kommunikation zwischen einzelnen Modellkomponenten auf Variablen, die den Energie-, Stoff- und Datenfluss im System abbilden, beschränkt. Die explizite Ausführung einzelner Verhaltensbestandteile einer Komponente des Modells durch andere Komponenten im Sinne ereignisdiskreter Kommunikationsmechanismen zur Laufzeit ist aufgrund der üblichen Überführung des Modells in eine Zustandsraumdarstellung und dessen Lösung mithilfe numerischer Verfahren (siehe Abschnitt 11.2.1) auch nicht möglich.

Im Steuerungsmodell wird die Anwendung der Signalports auf die Abbildung der Schnittstelle zum physikalischen Teilsystem und die Verknüpfung einzelner Ressourcen beschränkt. Damit wird der Tatsache Rechnung getragen, dass eine Speicherprogrammierbare Steuerung die Kommunikation mit anderen Systemen über ein Ein- und Ausgangsabbild (siehe Abschnitt 3.2.6.2), die Nutzung von Kommunikationsdiensten (beispielsweise einen Feldbus oder den steuerungsinternen Bus) und elektronische Baugruppen (beispielsweise digitale oder analoge Ein- und Ausgangsbaugruppen) realisiert. Beim Einlesen des Eingangsabbildes werden die als zeitkontinuierlich anzunehmenden Eingangsgrößen in definierten Speicherbereichen abgelegt. Nach dem Ende der zyklischen Datenverarbeitung erfolgt das Schreiben des Ausgangsabbildes. Dazu werden die entsprechenden Speicherbereiche ausgelesen und dem physikalischen Teilsystem bzw. anderen Ressourcen als zeitkontinuierliche Größen zur Verfügung gestellt[9]. Die Menge aller Schnittstellen der Ports vom Typ *„DSLSignalPort“* im Steuerungsmodell beschreibt folglich das Ein- und Ausgangsabbild der Speicherprogrammierbaren Steuerungen im modellierten System. Weiterhin ist festzustellen, dass aufgrund der technischen Ausführung sowie der klaren Trennung von Ein- und Ausgangssignalen das Schlüsselwort *„DSLACAUSAL“* für Signale bei der beschriebenen Art der Anwendung nicht zulässig ist. Vielmehr dürfen Ports zwischen dem physikalischen Teilmodell und dem Steuerungsmodell sowie verschiedenen Ressourcen nur mit Signalen eines Typs eindeutiger Kausalität (*DSLIN, DSLOUT*) spezifiziert werden.

Mit dem Subtyp *„SMOperationPort“* werden ereignisdiskrete Kommunikationsschnittstellen abgebildet. Zugeordnete Interfaces sind auf den Typ *„SMOperationInterface“* beschränkt. Die Anwendung dieser Art von Ports ist aufgrund der prozedu-

[9] Bussysteme in Kombination mit den elektronischen Baugruppen können aus Steuerungssicht aufgrund der beschriebenen zyklischen Arbeitsweise vereinfacht als wertdiskret, jedoch zeitkontinuierlich arbeitende Systeme betrachtet werden.

ralen Funktionsweise Speicherprogrammierbarer Steuerungen zur Modellierung der Kommunikation einzelner Subsysteme des Steuerungsmodells geeignet.

Eine spezielle Art von Port stellt der Typ *„DSLExternalPort“* dar. Er dient zur Abbildung der durch das Modell selbst festgelegten Systemgrenze und ermöglicht es, Modellinformationen über diese hinaus zur Verfügung zu stellen oder mit dem Modell des Fertigungssystems zu interagieren. Beispiele hierfür sind die Einbringung von Benutzereingaben zur Laufzeit, um die Interaktion zwischen dem Bedienpersonal und dem Fertigungssystem zu simulieren, und der Zugriff auf aktuelle Zustandsgrößen zum Zwecke einer dreidimensionalen Visualisierung. Als Beschränkung gilt, dass für diesen Subtyp nur Schnittstellen vom Typ *„DSLSignalInterface“* und kausale Signale zulässig sind.

Die Notation von Ports erfolgt anhand eines Quadrates, das mit einer durchgezogenen Linie umrandet ist und auf dem Rand der zugeordneten Komponente platziert wird. Durch zusätzliche, toolspezifische graphische Ergänzungen wird eine Unterscheidung der definierten Subtypen ermöglicht (siehe Abbildung 6.6). Operationports werden beispielsweise mit einer festzulegenden Farbe ausgefüllt, um die Eigenschaften der verwendeten Schnittstelle hervorzuheben. Die Darstellung des Typs *„DSLExternalPort“* geschieht mittels einer zusätzlichen Markierung durch einen schwarzen Punkt, während Signalports ergänzend mit einem Kreis versehen werden. Durchgangsports werden hingegen durch eine verkleinerte Darstellung gekennzeichnet.

Die Festlegung der Schnittstellen erfolgt in Textform gemäß der in Abbildung 6.6 dargestellten Syntax. Zum Zwecke der graphischen Hervorhebung kann optional die sogenannte *„Buchse-Stecker-Notation“* (siehe Abbildung 6.7) verwendet werden. Diese stellt benötigte Schnittstellen oder solche mit eingehenden Signalen als Halbkreis und verfügbare Schnittstellen sowie Interfaces mit ausgehenden Signalen als Kreis dar. Beide Symbole werden mit dem besitzenden Port durch eine Linie verbunden. Werden Ports mit kompatiblen Schnittstellen durch einen Konnektor miteinander verknüpft, so werden beide Symbole nebeneinander dargestellt. Die *„Stecker-Buchse-Notation“* hat keine semantische Bedeutung für das Modell, erleichtert aber dessen Interpretation. Beispielsweise ist es sinnvoll, die Schnittstelle zwischen dem physikalischen Teilmodell und dem Steuerungsmodell in dieser Form abzubilden, um die Trennung der beiden Subsysteme deutlich zu machen.

6.3.5 Anwendung der Sprachelemente zur Strukturmodellierung

Die Anwendung der beschriebenen Sprachelemente zur Modellierung der Struktur automatisierter Fertigungssysteme erfolgt anhand von Komponentendiagrammen. Da die UML einen sichtenorientierten Modellbegriff unterstützt, wird auf die vollständige Visualisierung aller strukturellen Elemente im Diagramm verzichtet. Wie in Abbildung 6.7 exemplarisch dargestellt, werden lediglich Komponenten mit ihrem Namen

und Namensraum sowie Ports, Konnektoren und optional die Verbindung zwischen dem physikalischen Teilmodell und dem Steuerungsmodell in Form der *„Stecker-Buchse-Notation“* abgebildet. Variablen, elementare Strukturbausteine, die vollständige Spezifikation der verwendeten Schnittstellen sowie zusätzliche Eigenschaften der abgebildeten Strukturelemente sind toolspezifisch außerhalb des Diagramms zur Verfügung zu stellen. Im Hinblick auf den Modellaufbau ist deren Integration durch geeignete Interaktionsmöglichkeiten zu unterstützen.

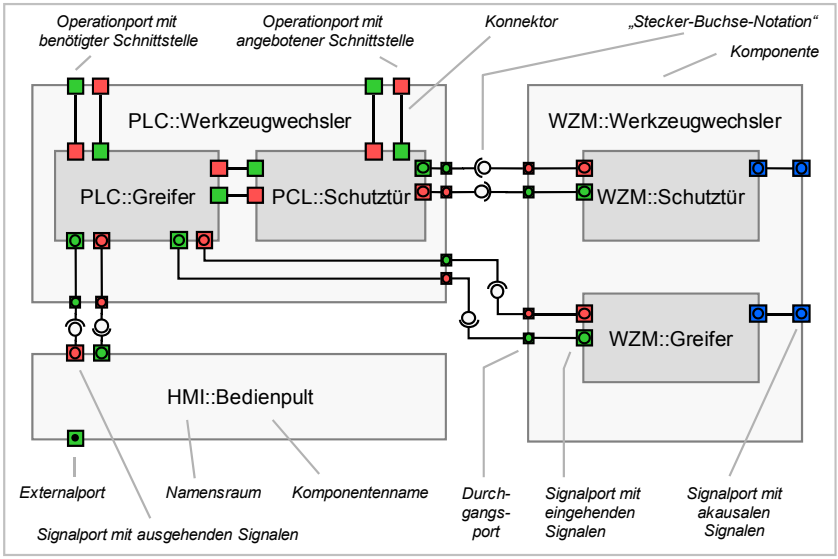


Abbildung 6.7: Beispiel eines Komponentendiagramms

Weiterhin sollte es zur Verbesserung der Übersichtlichkeit möglich sein, im Sinne einer „Black-Box-Darstellung“ verschachtelte Komponenten, Ports und Konnektoren auszublenden oder in separaten Diagrammen zu visualisieren.

6.4 Sprachelemente zur Spezifikation des Systemverhaltens

6.4.1 Übersicht

Die erforderlichen Modellelemente zur Spezifikation des Systemverhaltens sind relativ umfangreich und bauen teilweise aufeinander auf. Um eine einfache Verständlichkeit zu gewährleisten, weicht ihre nachfolgende Beschreibung partiell von der Reihenfolge ihrer Anwendung bei der Entwicklung ab. Zunächst werden Funktionen als Sprachelemente zum Aufbau eines hierarchischen Funktionsmodells eingeführt. Anschließend erfolgt in Abschnitt 6.4.3 die Darstellung der Modellkonstrukte zur Formulierung

des Verhaltens des Steuerungssystems. Abschnitt 6.4.4 hingegen erweitert die domänenspezifische Modellierungssprache um eine graphische Möglichkeit zur Abbildung des Verhaltens des physikalischen Teilsystems. Die abschließende Definition von Elementen zur Spezifikation des komponentenübergreifenden Informationsaustauschs komplettiert die Vorstellung der notwendigen Sprachkonstrukte.

6.4.2 Funktionen als teilsystemübergreifende, abstrakte Sprachelemente

Funktionen definieren in einer abstrahierten und nicht näher beschriebenen Form einen Ausschnitt des Verhaltens, das durch einzelne Systemkomponenten – unabhängig von deren Zuordnung zum jeweiligen Teilsystem – zu implementieren ist. Sie werden analog zu Variablen und Schnittstellen durch eine Zeichenkette syntaktisch dargestellt (siehe Abbildung 6.8).

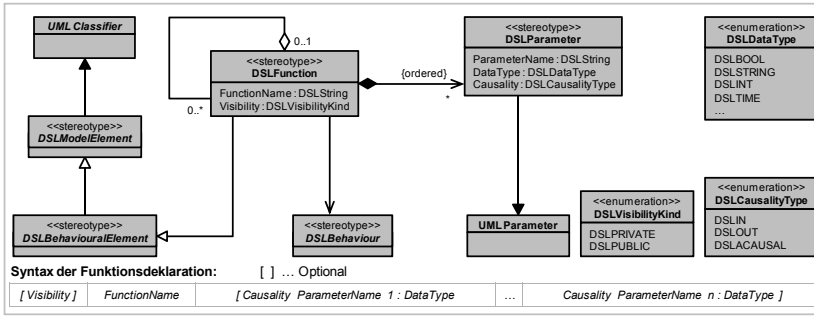


Abbildung 6.8: Metamodell und Syntax der Funktionen

Neben einem aussagekräftigen Namen *„FunctionName“* kann die Angabe optional um das Attribut Sichtbarkeit *„Visibility“* sowie eine endliche Menge von Übergabe- und Rückgabeparametern *„DSLParameter“* erweitert werden. Diese zusätzlichen Angaben erlauben eine Detaillierung der Eigenschaften einer Funktion. Die Sichtbarkeit gibt Auskunft darüber, ob eine Funktion ein Verhalten implementiert, das nur für die Komponente selbst nutzbar ist, oder ob dieses auch anderen Subsystemen zur Verfügung gestellt wird. Die Parameter hingegen sind als Verweise auf Systemgrößen zu betrachten, die zur Erfüllung einer Funktion benötigt oder von dieser bereitgestellt werden.

Durch die beschriebenen Möglichkeiten zur Präzisierung der Funktionen wird die Modelliererin bzw. der Modellierer beim Aufbau der Systemarchitektur unterstützt. Zum einen helfen sie, die Eigenschaften der einzelnen Komponenten festzulegen. Zum anderen wird die Modellierung kompakter Schnittstellen unterstützt. Benötigt beispielsweise eine Funktion, die durch eine bestimmte Komponente umgesetzt werden soll, viele Eigenschaften anderer Systembausteine, so stellt sich die Frage, ob die Zuordnung und die Implementierung dieser Funktion durch die Komponente sinnvoll ist.

Funktionen können im Sinne einer übersichtlicheren Modellgestaltung auch als Referenz auf komplementäre Elemente zur Verhaltensmodellierung dienen, die deren Umsetzung im Detail beschreiben. So kann beispielsweise das Verhalten, das zur Umsetzung einer bestimmten Aufgabe notwendig ist, durch eine Funktion zusammengefasst und strukturiert werden. Die hierzu im Detail notwendigen Sprachkonstrukte werden in den nachfolgenden Abschnitten vorgestellt.

6.4.3 Modellierung des Verhaltens des Steuerungssystems

6.4.3.1 Allgemeines

Bei der Spezifikation des Verhaltens einzelner Komponenten des Steuerungssystems werden prinzipiell zwei aufeinander aufbauende Modellierungsebenen unterschieden. Auf der prozedural-funktionalen Ebene werden einzelne Funktionalitäten unter Nutzung atomarer Sprachelemente im Detail ausgearbeitet und modelliert. Für diese Aufgabe werden Aktivitäten eingesetzt, da sich diese für die Spezifikation der Verarbeitung von Daten auf einem feingranularen Niveau besonders eignen und zudem eine hohe Affinität zu den graphischen Sprachen der IEC 61131-3 (Funktionsbausteinsprache, Ablaufsprache) aufweisen. Die übergeordnete Ebene hat hingegen eine koordinative Funktion. Ihre Aufgabe ist es, die Funktionen der untergeordneten Ebene anzustoßen und entsprechend dem Zustand und der geforderten Gesamtfunktionalität auf einer abstrakteren Ebene zu steuern. Hierzu sind neben koordinierenden Aktivitäten insbesondere Zustandsautomaten und deren Darstellung in Form von Zustandsdiagrammen prinzipiell geeignet. Die Sprachelemente auf beiden genannten Ebenen sind jedoch an den Anwendungsbereich der Softwareentwicklung für Speicherprogrammierbare Steuerungen anzupassen und zu formalisieren.

6.4.3.2 Aktivitäten und Aktionen

Aktivitäten sind analog zu den Petri-Netzen gerichtete Graphen und bestehen aus einer endlichen Menge von Knoten und Kanten. Aktivitätsknoten definieren nicht weiter zerlegbare Verhaltensbestandteile einer Aktivität und umfassen Kontrollkonstrukte (Kontrollknoten), Datenobjekte (Objektknoten) und Aktionen, die auch als Aktionsknoten bezeichnet werden. Die Spezifikation der Ausführungsreihenfolge der einzelnen Aktionen erfolgt anhand von Aktivitätskanten. Diese verbinden die einzelnen Knoten miteinander und definieren zusammen mit den Kontroll- und Objektknoten im Rahmen eines inhärenten Koordinationskonzepts durch die Weitergabe von Daten bzw. durch die Weitergabe der Ausführungskontrolle das Gesamtverhalten.

Aktionen werden als Rechtecke mit abgerundeten Ecken dargestellt und repräsentieren vordefinierte Funktionen, die Eingabedaten empfangen, verarbeiten und als Ausgaben zur Verfügung stellen können. Während die UML 44 derartige atomare Elemente festlegt, sind für Speicherprogrammierbare Steuerungen aufgrund der Strukturinvarianz

der Programme sowie eingeschränkter Möglichkeiten zur Datenverarbeitung nur ausgewählte Aktionen notwendig. Gleichzeitig erfordert die unabdingbare Orientierung an der IEC 61131-3 ergänzende Subtypen, die speziell die Funktionsweise der SPS berücksichtigen. Abbildung 6.9 zeigt die für die domänenspezifische Modellierungssprache definierten Typen von Aktionen.

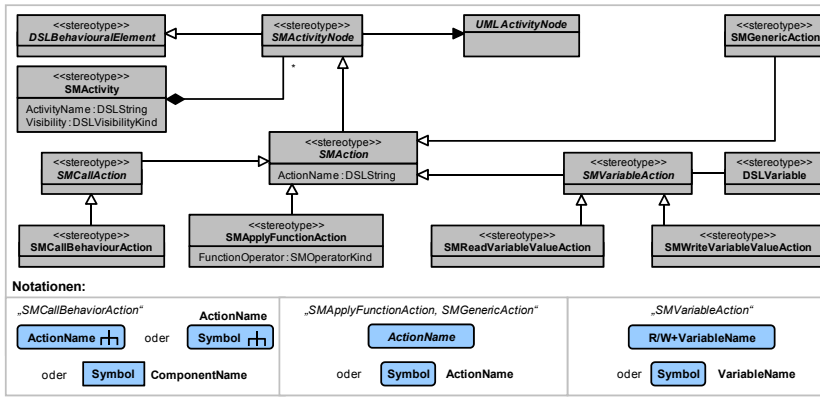


Abbildung 6.9: Metamodell und Notation der Aktionen

Prinzipiell werden drei Klassen von Aktionen unterschieden. Kommunikationsbezogene Aktionen *„SMCallAction“* umfassen Sprachelemente zur Modellierung der Kommunikation zwischen einzelnen Komponenten des Steuerungssystems. Der einzig existierende Subtyp *„SMCallBehaviorAction“* wird dazu verwendet, das Verhalten einer anderen Komponente oder von Teilen davon (Funktionen, Aktivitäten) aufzurufen. Da die Kommunikation ausschließlich über Ports und Schnittstellen zulässig ist, muss das abzuarbeitende Verhalten indirekt bestimmt werden. Diese geschieht, indem die Komponente eine entsprechende Operation *„SMOperation“* über eine Schnittstelle vom Typ *„SMREQUIRED“* zur Verfügung stellt, welche die aufrufende Aktion referenziert. Die aufzurufenden Verhaltenskonstrukte hingegen werden mit einer Operation gleichen Namens aber Parametern inverser Kausalität über eine komplementäre Schnittstelle vom Typ *„SMPROVIDED“* verknüpft. Nach der Modellierung eines Konnektors zwischen den Ports, denen die jeweiligen Schnittstellen zugeordnet sind, ist die Verbindung vollständig definiert. Um die semantische Eindeutigkeit sicherzustellen und die Interpretation einer derartigen Aktion zu erleichtern, ist es zweckmäßig, dass das auszuführende Verhalten einen innerhalb der besitzenden Komponente eindeutigen und aussagekräftigen Namen aufweist. Dieser sollte dann sowohl zur Schnittstellendefinition *(„OperationName“)* als auch für die aufrufende Aktion (*„ActionName“*) verwendet werden. Alternativ ist ergänzend zum Namen ein geeignetes Symbol zur Visualisierung zu verwenden. Eine besondere Berücksichtigung findet die

Kommunikation mit elementaren, im Komponentendiagramm nicht dargestellten Strukturbausteinen mit vordefiniertem, einfachen Verhalten (Zählerbausteine, Zeitglieder, Speicherbausteine, Flankenerkennungen). Diese werden analog zu Komponenten als Rechteck dargestellt und durch ein geeignetes Symbol gekennzeichnet. Der Name der Komponente wird zusätzlich neben der graphischen Darstellung annotiert. Eine Abbildung aller zulässigen elementaren Strukturbausteine zeigt Abschnitt 11.4.2.

Des Weiteren ist auch die Ausführung anderer Verhaltensbestandteile (Aktivitäten) der besitzenden Komponente möglich. Da alle Aktionen innerhalb des Steuerungssystems aufgrund der streng sequentiellen Arbeitsweise Speicherprogrammierbarer Steuerungen als synchron zu betrachten sind[10], ist die Rekursionsfreiheit der Verhaltenssteuerung sicherzustellen. Diese schließt beispielsweise aus, dass eine Aktion die Aktivität aufruft, innerhalb derer sie verwendet wird, oder dass Aktivitäten sich gegenseitig aufrufen können.

Variablenbezogene Aktionen dienen dem direkten Lesen *(„SMReadVariableValueAction")* und Schreiben *(„SMWriteVariableValueAction")* der Variablen einer Komponente und sind somit aus systemtheoretischer Sicht als fundamentale Elemente zum Zugriff auf die Eingangs-, Ausgangs- und Zustandsgrößen im System zu betrachten. Beide Subtypen erlauben analog zu den kommunikationsbezogenen Aktionen einen komponentenübergreifenden Zugriff über Ports und Schnittstellen, wobei dieser auf Variablen mit dem Schlüsselwort *„DSLPUBLIC"* beschränkt ist. Innerhalb einer Komponente können jedoch auch nicht öffentlich zugängliche Größen (Schlüsselwort *„DSLPRIVATE"*) sowie Signale referenziert werden. Der Aktionsname hat dem Namen der gelesenen oder geschriebenen Variable *„VariableName"* bzw. dem Namen des Signals *„SignalName"* inklusive eines Präfixes zur Kennzeichnung der Art des Zugriffs zu entsprechen. Bezüglich der Signale ist anzumerken, dass durch deren Verwendung die Modellierung der Kommunikation zwischen einzelnen Ressourcen und mit dem physikalischen Teilsystem ermöglicht wird. Sie erlauben den indirekten Zugriff auf die kontinuierlichen Größen im physikalischen Teilsystem und implementieren damit das aus der Theorie hybrider Systeme (siehe Abschnitt 11.2.3) bekannte Konzept des Injektors. Das komplementäre Konstrukt des Quantisierers wird hingegen durch die im Folgenden dargestellten Aktionsarten umgesetzt, die sowohl Variablen als auch Signale auswerten und entsprechend den Aktualwerten Änderungen im Steuerungsmodell induzieren.

[10] Tatsächlich definiert die IEC 61131-3 auch ein Taskkonzept, das analog zu PC-basierten Systemen eine quasiparallele Abarbeitung einzelner Programm-Organisationseinheiten zulässt. Da dieses aber die Verhaltensausführung eindeutig regelt *(DIN EN 61131-3, S. 112-113),* ist diese Form der Betrachtung zulässig. Aufgrund der hardwarespezifischen Umsetzung wird ferner die Verteilung der Komponenten des Steuerungssystems auf einzelne Tasks im Rahmen der Modellbildung nicht festgelegt. Im Zuge der Transformation der Modelle in Steuerungscode sind derartige Informationen zu ergänzen (siehe Abschnitt 6.5), wodurch entsprechende Zusammenhänge automatisiert identifiziert und berücksichtigt werden können.

Die Klasse der primitiven Funktionen *„SMApplyFunctionAction“* ist zur Festlegung elementarer Konstrukte zur Datenverarbeitung notwendig. Beispiele sind mathematische Funktionen, Vergleichsoperationen oder grundlegende Möglichkeiten zur Verarbeitung von Zeichenketten. Die UML lässt eine genauere Spezifikation offen, für eine formale Systemmodellierung ist eine weitere Präzisierung jedoch zwingend erforderlich. Dies geschieht im Rahmen der DSL durch den Eigenschaftswert (Tagged Value) *„FunctionOperator“*, der die Funktion im Detail charakterisiert. Die Menge der erlaubten Funktionen *„SMOperatorKind“* umfasst alle in *(DIN EN 61131-3)* definierten elementaren Standardfunktionen. Eine vollständige Auflistung der möglichen Funktionen ist Abschnitt 11.4.2 zu entnehmen.

Des Weiteren wird eine generische Aktion *„SMGenericAction“* eingeführt. Dieser Aktionstyp kann verwendet werden, um Aufgaben zu beschreiben, deren Umsetzung noch nicht im Detail spezifiziert werden kann oder deren Modellierung mit den bereits aufgeführten Aktionen nicht möglich ist. Ihre besondere Fähigkeit besteht darin, Daten bereitzustellen, die durch weitere Aktionen verarbeitet oder über Parameter an die aufrufenden Verhaltenselemente zurückgegeben werden können.

Kontrollknoten sind Elemente zur Ausführungssteuerung und ermöglichen die Festlegung des Start- und Endpunktes einer Aktivität. Weiterhin erlauben sie die Modellierung alternativer Ausführungen durch Verzweigungen und Vereinigungen sowie die Spezifikation von nebenläufigem Systemverhalten durch Elemente zur Parallelisierung und Synchronisation einzelner Aufgaben. Aufgrund der direkten Übernahme dieser Elemente aus der UML wird auf eine formale Darstellung verzichtet. Abbildung 6.10 zeigt die Notation der definierten Elemente, bezüglich der formalen Spezifikation wird auf *(OMG 2005a, S. 270-272)* verwiesen.

Starten und Beenden von Aktivitäten		Alternative Ausführungen		Nebenläufiges Verhalten	
Initialknoten		Entscheidungsknoten		Parallelisierungsknoten	
Aktivitätsendknoten		Vereinigungsknoten		Synchronisationsknoten	
Ablaufendknoten					

Abbildung 6.10: Arten von Kontrollknoten und ihre Notation

Objektknoten stellen das Bindeglied zwischen der Verhaltensmodellierung in Form von Aktivitäten und der Struktur einer Komponente dar. Sie können Daten aufnehmen, wobei im Gegensatz zur UML eine Beschränkung auf Variablen und die Über- und Rückgabeparameter der aufrufenden Operation erfolgt. Sie dienen damit zum einen als die formalen Ein- und Ausgabeparameter einer Aktivität, können aber auch eine Pufferfunktion während deren Ausführung übernehmen. Eine detailliertere Klassifizierung

unterscheidet Pins *„SMPin“*, Parameterknoten *„SMParameterNode“* und Pufferknoten[11] *„SMBufferNode“* (siehe Abbildung 6.11). Alle Subtypen verfügen über einen Datentyp, der die Art der Daten definiert, die durch den jeweiligen Objektknoten aufgenommen werden können. Pins sind als die formalen Parameter von Aktionen zu betrachten und entsprechen damit den Ein- und Ausgangsgrößen einer Aktion. Sie sind Teil der Aktion selbst und werden als kleine Rechtecke direkt an die Aktionsknoten angefügt. Die Datentypen der jeweiligen Pins müssen konform zu den Aktionen sein, der Objektname ist optional und kann das aufzunehmende Objekt näher beschreiben. Durch eine weitere Präzisierung werden eingehende Größen *„SMInputPin“* und ausgehende Größen *„SMOutputPin“* unterschieden. Der ergänzende Subtyp *„SMValuePin“* definiert feste Werte, die durch den Eigenschaftswert *„Causality“* als Ein- oder Ausgangsgrößen der besitzenden Aktion festgelegt sind. Zur Vereinfachung der Modellbildung verfügen Pins darüber hinaus über eine implizite Funktion zur Negation der durch sie repräsentieren Größen. Diese wird aktiviert, wenn der Tagged Value *„IsNegation“* den Wert *„TRUE“* annimmt. Die Notation eines Pins wird in diesem Fall durch einen kleinen Kreis ergänzt. Parameter- und Pufferknoten werden ebenfalls als Rechtecke dargestellt, im Gegensatz zu Pins aber nicht direkt an Aktionen gebunden. Erstere sind überlappend auf dem umschließenden Rahmen der Aktivität zu positionieren.

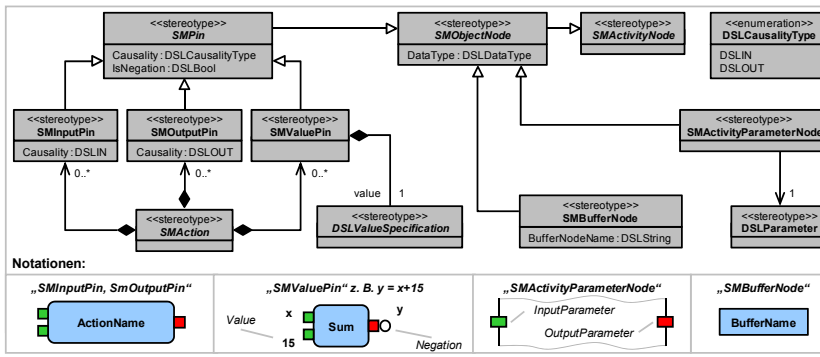


Abbildung 6.11: Metamodell und Notation der Objektknoten

Allen Arten von Knoten ist gemeinsam, dass sie durch Aktivitätskanten *„SMActivityEdge“* miteinander verknüpft werden. Diese bestimmen zusammen mit den Kontrollknoten die zeitliche und logische Reihenfolge der Ausführung der einzelnen Aktionen. Es erfolgt eine prinzipielle Unterscheidung in Objekt- und Kontrollflusskanten (siehe Abbildung 6.12). Erstere modellieren den Datenfluss, werden als

[11] Derartige Objektknoten sind mit Bezug auf die IEC 61131-3 als temporäre Variablen zu betrachten und werden durch einen eindeutigen Namen *„BufferName“* gekennzeichnet.

durchgehende Linien mit offener Pfeilspitze dargestellt und verbinden Objektknoten innerhalb der Aktivität.

Es gilt die Regel, dass der Datentyp der verbundenen Elemente kompatibel sein muss. Ein Objektknoten mit Booleschem Datentyp kann folglich nicht mit Objektknoten anderer Datentypen verbunden werden. Ausnahmen gelten bezüglich der Datentypen, für die in der IEC 61131-3 geeignete Funktionen zur Datentypumwandlung festgelegt sind. Diese müssen jedoch nicht explizit modelliert werden, da im Rahmen der Transformation der Modelle in Steuerungscode entsprechende Ergänzungen automatisiert vorgenommen werden können.

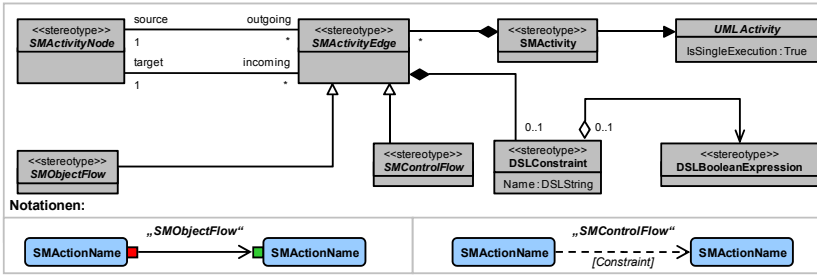


Abbildung 6.12: Metamodell und Notation der Aktivitätskanten

Kontrollflusskanten werden in Form von Pfeilen mit Strichlinien abgebildet. Sie drücken eine Abhängigkeit zwischen zwei Aktionsknoten aus. Ein nachfolgender Knoten (Target) kann nur dann verarbeitet werden, wenn der direkt vorhergehende Knoten (Source) die Kontrolle weitergibt. Die Verbindung der Aktionsknoten durch Kontrollflusskanten erfolgt direkt oder über entsprechende Kontrollknoten. Für Kontrollflusskanten können weiterhin Überwachungsbedingungen *„DSLConstraint“* in Form Boolescher Ausdrücke *„DSLBooleanExpression“* definiert werden. Eine Weitergabe der Kontrolle erfordert als ergänzende Randbedingung, dass der entsprechende Ausdruck als „wahr“ ausgewertet werden kann, wobei dessen Operanden Variablen innerhalb der Komponente, Signale oder konkrete Werte referenzieren müssen.

Zur formalen Modellierung mit Aktivitäten ist ein Koordinationskonzept notwendig, das den Datenfluss und die Reihenfolge der Datenverarbeitung regelt. Dies geschieht in Analogie zu den Petri-Netzen (siehe Abschnitt 11.2.2) anhand der Erzeugung und Weitergabe von Token[12], die übereinstimmend mit der UML in Datentoken[13] und

[12] Ein Token ist im Rahmen der Graphentheorie als ein Element zur Steuerung der Ausführungskontrolle und/oder zum Transport von Daten zu betrachten. Dieses kann nach definierten Regeln im Graphen weitergegeben werden. Die durch ein Token gekennzeichneten Aktionen sind als aktiv zu betrachten und werden ausgeführt. Damit entspricht ein Token den für Petri-Netze definierten Marken.

[13] Datentoken werden auch als Objekttoken bezeichnet.

Kontrolltoken unterschieden werden. Erstere sind als eine Befähigung zum Ausführen von Aktionen zu betrachten, während Datentoken zusätzlich einen Datenwert mittransportieren. Diese Möglichkeit impliziert, dass Datentoken nur an Objektknoten weitergegeben werden dürfen und Kontrolltoken auf die Weiterleitung über Kontrollflusskanten beschränkt sind. Für die Erzeugung, Verteilung, Zusammenführung und Weitergabe von Token sind für die einzelnen Modellkonstrukte der Aktivitäten eine Reihe weiterer Regeln zu beachten, die zusammenfassend in Abbildung 6.13 dargestellt sind. Diese sind teilweise nicht konform zum Tokenkonzept der UML. Deren Anwendung ist jedoch erforderlich, um eine Überführung der Modelle in Steuerungscode im Zuge der Entwicklung zu gewährleisten. Weitere Randbedingungen, die aus logischer Sicht zu problematischen Modellen führen würden (beispielsweise die direkte Verknüpfung eines Entscheidungsknoten mit einem Synchronisierungsknoten), sind durch entsprechende Entwicklungswerkzeuge zu prüfen und gegebenenfalls zu unterbinden.

Modellelement	Regeln
Initialknoten	• Erzeugt beim ersten Aufruf der Aktivität an jeder ausgehenden Kante ein Kontrolltoken. • Eine Aktivität kann mehrere aber auch keinen Initialknoten besitzen. • Alle Knoten ohne eingehende Kanten werden beim ersten Aufruf mit einem Token belegt. • Alle Knoten ohne eingehende Kanten und ausgehenden Datenflusskanten werden bei jedem Aufruf mit einem Token belegt. • Parameterknoten stellen ebenfalls Token zur Verfügung.
Aktivitätsendknoten	• Eine Aktivität kann einen oder mehrere Aktivitätsendknoten besitzen. • Erreicht ein Token einen Aktivitätsendknoten wird die ganze Aktivität beendet und alle Kontrolltoken werden gelöscht. Datentoken die an den Parametern anliegen werden an das aufrufende Verhaltenskonstrukt zurückgegeben.
Ablaufendknoten	• Erreicht ein Token einen Ablaufendknoten wird der entsprechende Ablauf beendet. Weitere Token bzw. Abläufe sind davon nicht betroffen.
Entscheidungsknoten	• Ein Token, das einen Entscheidungsknoten über die eingehende Kante erreicht, wird nur über eine ausgehende Kante an einen anderen Knoten weitergegeben. Durch entsprechende Überwachungsbedingungen oder weitere Kontrollknoten ist ein deterministisches Verhalten sicherzustellen. • Es sind nur Aktivitätskanten desselben Typs zu verwenden.
Vereinigungsknoten	• Ein Token, das einen Vereinigungsknoten über eine der eingehenden Kanten erreicht, wird an den nachfolgenden Knoten weitergegeben, sobald dieser zur Aufnahme bereit ist und vorhandene Überwachungsbedingungen erfüllt sind. • Es sind nur Aktivitätskanten desselben Typs zu verwenden.
Parallelisierungs-knoten	• Liegt an der eingehenden Kante ein Token an, wird an allen ausgehenden Kanten ein Token angeboten. • Ausgehende Kanten sind nicht mit Überwachungsbedingungen zu versehen.
Synchronisations-knoten	• Liegt an allen eingehenden Kanten ein Token an, wird an der ausgehenden Kante ein Token angeboten. • Eingehende Kanten können Kontrollflusskanten oder Objektflusskanten sein, die Daten des Typs „*DSLBool*“ transportieren. • Liegen unterschiedliche Arten von Token an, wird ein Kontrolltoken weitergegeben.
Action Aktionsknoten	• Es muss an allen eingehenden Kanten ein Token angeboten werden, damit eine Aktion ausgeführt werden kann. • Sobald alle erforderlichen Token vorhanden sind werden diese vereint, fließen in den Knoten ein und dieser kann ausgeführt werden. Die Token bleiben bei diesem Knoten, solange dessen Ausführung dauert bzw. der Folgeknoten die Token akzeptiert und eventuelle Überwachungsbedingungen erfüllt sind. • Ein Knoten kann jeweils nur ein Token aufnehmen (Kapazität = 1). • Nach der Ausführung werden Token an den ausgehenden Kanten gleichzeitig angeboten. • Nach dem Ausführen der Aktion wird an allen ausgehenden Objektflusskanten ein Token angeboten.
Buffer Pufferknoten	• Ein Pufferknoten stellt das eingehende Objekttoken direkt wieder zur Verfügung.
Aktivitätskanten	• Eine Aktivitätskante kann jeweils nur ein Token weiterleiten (Gewicht = 1). • Ein Kontrolltoken kann bei jedem Aktivitätsaufruf maximal zum nächsten Aktionsknoten weitergeleitet werden. • Ein Token wird über die Kante weitergeleitet, sobald der nachfolgende Objekt- oder Aktionsknoten das Token akzeptiert und die Überwachungsbedingungen erfüllt sind.
Aktivitäten	• Die Reihenfolge der Abarbeitung mehrerer aktiver Knoten erfolgt in Analogie zur IEC 61131-3 von links oben nach rechts unten. Mehrere nicht über Aktivitätskanten verknüpfte Teilaktivitäten werden sequentiell von oben nach unten abgearbeitet. • Können keine Token mehr weitergereicht werden, so wird die Kontrolle an das aufrufende Verhaltenselement zurückgegeben. Anschließend werden alle Objekttoken gelöscht.

Abbildung 6.13: Regeln für das Tokenkonzept der Aktivitäten

Die hohe Affinität der Aktivitäten und ihrer Darstellung in Form von Aktivitätsdiagrammen zu einigen graphischen Sprachen der IEC 61131-3 lässt es zu, sowohl Verknüpfungsteuerungen als auch Ablaufsteuerungen zu modellieren. Verknüpfungsteuerungen zeichnen sich dadurch aus, dass sie datenflussdominiert sind und abhängig von eingehenden Größen und internen Zuständen nach einer unveränderlichen Berech-

nungsvorschrift weitere Zustandsgrößen und Ausgangsgrößen berechnen. Ablaufsteuerungen sind hingegen kontrollflussdominiert. Die Berechungsvorschrift ist vom aktuellen Zustand des Systems abhängig. Zur Verdeutlichung zeigt Abbildung 6.14 anhand eines Beispiels eine Gegenüberstellung beider Steuerungsarten in graphischen Sprachen der IEC 61131-3 sowie deren Umsetzung anhand von Aktivitätsdiagrammen.

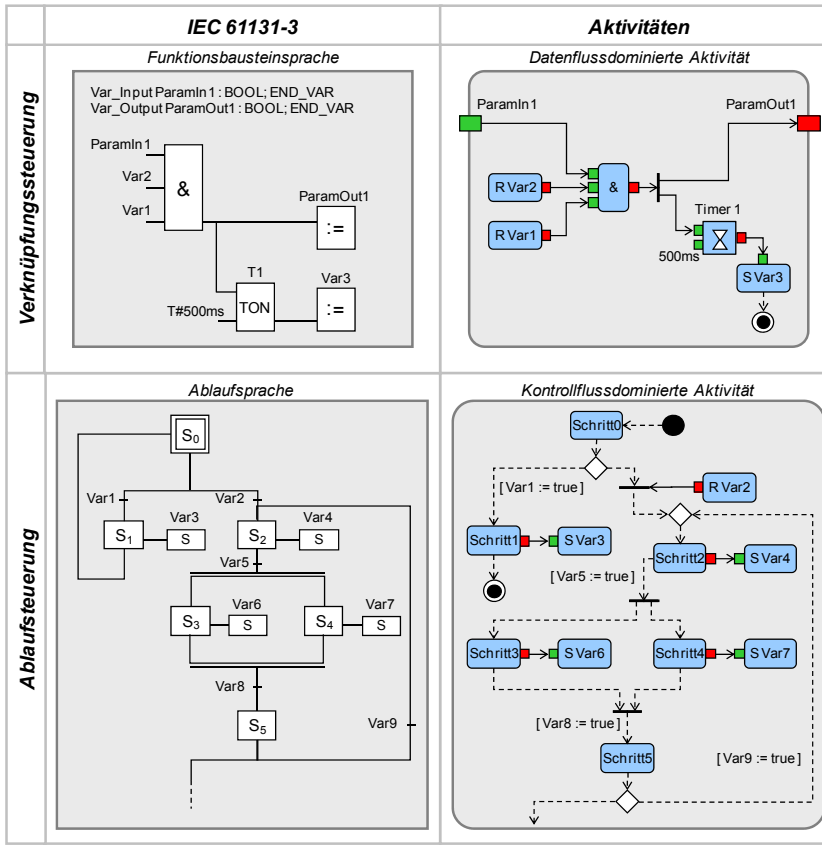


Abbildung 6.14: Aktivitäten und graphische Sprachen der IEC 61131-3

Die dargestellten Formen der Verhaltensmodellierung erfüllen auf der einen Seite die Anforderungen an einen praxisnahen und effizienten Ansatz. Auf der anderen Seite bieten die Aktivitätsdiagramme über die Modellierung von reinen Verknüpfungs- oder Ablaufsteuerungen hinausgehende Möglichkeiten. Hierbei ergibt sich jedoch die Herausforderung, die Transformation der Modelle in Steuerungscode zu realisieren. Entsprechende Transformationsregeln müssen die eindeutige Auflösbarkeit der eingesetz-

ten Modellelemente gewährleisten, so dass eine Überführung in zur IEC 61131-3 konforme Programmbausteine ermöglicht wird. Dies hat zur Folge, dass zum einen die notwendigen Generatoren wesentlich komplexer werden und zum anderen die Lesbarkeit des generierten Codes darunter leidet. Eine Orientierung an der dargestellten Art der Modellierung mit Aktivitäten ist daher gerade für die Detaillierung der Modelle im Rahmen des Systementwurfs zu empfehlen.

6.4.3.3 Zustandsautomaten der UML

Die Zustandsautomaten der UML sind als eine objektorientierte Variante der endlichen Zustandsautomaten nach Harel (siehe Abschnitt 11.2.2) zu betrachten. Sie sind analog zu den Aktivitäten gerichtete Graphen, die aus einer endlichen Menge von Knoten und Kanten bestehen. Die Knoten des Graphen stellen die Zustände dar, die von dem Objekt, dessen Verhalten abgebildet wird, eingenommen werden können. Die Kanten hingegen legen die möglichen Zustandsübergänge fest und werden als Transitionen bezeichnet. Für den Entwurf der Steuerungssoftware stellt sich die Frage, wodurch die Zustände definiert werden. Aus rein systemtheoretischer Sicht sind dies für eine Softwarekomponente alle möglichen Wertkombinationen der Zustandsgrößen, wodurch sich bei nicht trivialen Bausteinen sehr schnell eine erhebliche Menge von Zuständen ergibt. In der Praxis ist ein derartiger Ansatz aufgrund der resultierenden Komplexität nicht einsetzbar. Vielmehr ist es Aufgabe der Systementwicklerinnen und -entwickler, durch die Auswahl geeigneter Kombinationen von Zustandsgrößen und die Spezifikation eindeutiger Ausgangs- und Zustandsübergangsbedingungen ein sinnvolles und deterministisches Systemverhalten zu modellieren. Dabei ist der Determinismus als notwendige, aber nicht hinreichende Bedingung zu betrachten. Die Präzision der Verhaltensspezifikation muss auch durch das Ausschließen aller (oder von möglichst vielen) unzulässigen Systemzuständen erreicht werden.

Für die domänenspezifische Modellierungssprache wird eine vereinfachte Form der Zustandsautomaten der UML als geeignet angesehen. Gerade die Möglichkeiten zur Zustandsverfeinerung sind für die koordinative Aufgabe dieses Mittels zur Verhaltensbeschreibung als nicht notwendig zu betrachten und werden im Sinne einer übersichtlichen Modellgestaltung nicht eingesetzt. Somit reduzieren sich die erforderlichen Sprachkonstrukte auf die in Abbildung 6.15 dargestellten Elemente.

Demnach wird ein Zustandsgraph *„SMStateMachine“* in eine oder mehrere Regionen *„SMRegion“* unterteilt. Regionen sind orthogonale[14] Teile eines Zustandsgraphen und bestehen wiederum aus einer Menge von Zuständen *„SMState“* und verbindenden Transitionen *„SMTransition“*. Letztere verfügen genauso wie die Zustände selbst über

[14] Unter Orthogonalität ist in der Informationstechnik die freie Kombinierbarkeit unabhängiger Konzepte zu verstehen. Damit sind orthogonale Regionen als unabhängig voneinander zu betrachten, wodurch die Modellierung von nebenläufigem Verhalten möglich wird.

Aktivitäten, die ausgeführt werden, wenn der jeweilige Zustand aktiv[15] ist bzw. ein Zustandsübergang stattfindet. Eine Ausnahme gilt für den Initalzustand, der Teil eines jeden Subgraphen ist und keine Aktivitäten besitzt.

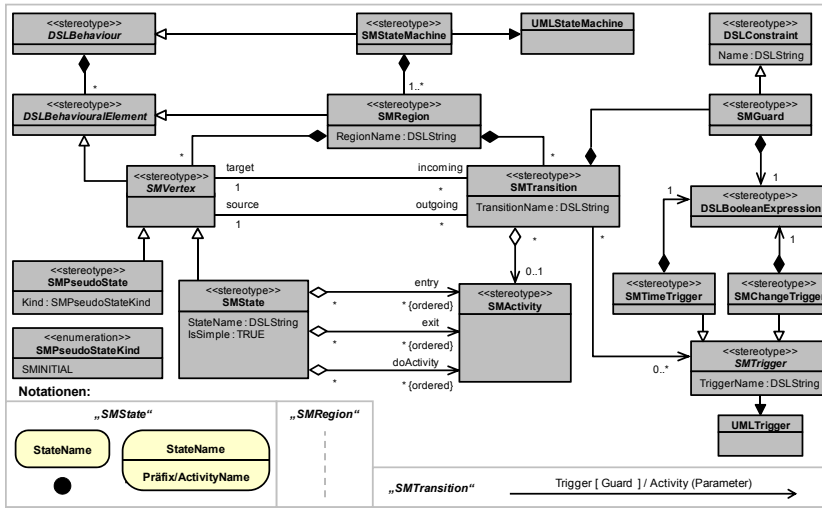


Abbildung 6.15: Metamodell und Notation der Zustandsautomaten

Aktivitäten, die Zuständen zugeordnet sind, werden durch die Präfixe *„entry“*, *„exit“* und *„doActivity“* in drei Klassen eingeteilt. Die beiden erstgenannten kennzeichnen Aktivitäten, die nur dann ausgeführt werden, wenn ein Zustand aktiviert bzw. ein aktiver Zustand verlassen wird. Aktivitäten mit dem Präfix *„doActivity“* werden hingegen abgearbeitet, solange der besitzende Zustand aktiv ist. Transitionsbezogene Aktivitäten sind demgegenüber nur einmalig beim Zustandsübergang ausführbar. Um die Modellbildung zu vereinfachen, können den Zuständen auch Aktionen zugeordnet werden. Dies ist jedoch nur dann zulässig, wenn diese nicht sinnvoll zu einer komplexeren Aktivität aggregierbar sind.

Zur formalen Spezifikation des Übergangs von einem Ausgangs- in einen Zielzustand sind die Sprachelemente *„SMTransition“*, *„SMTrigger“* und *„SMGuard“* notwendig. Während Transitionen lediglich definieren, zwischen welchen Zuständen gewechselt werden kann, beschreiben Trigger die hierzu erforderlichen Ereignisse. Diese können beispielsweise das Ändern des Wertes einer Variablen bzw. eines Signals *(„SMChangeTrigger“)* oder der Ablauf eines Timers *(„SMTimeTrigger“)* sein, der beim Aktivieren des Zustands gestartet wurde. Guards sind analog zu den Überwa-

[15] Jeder Subgraph kann nur genau einen aktiven Zustand besitzen.

chungsbedingungen bei Aktivitäten als zusätzliche Einschränkung in Form Boolescher Ausdrücke *„DSLBooleanExpression“* zu verstehen. Ein Zustandsübergang erfordert als ergänzende Randbedingung, dass der entsprechende Ausdruck als „wahr“ ausgewertet werden kann. Dessen Operanden müssen Variablen, Signale oder konkrete Werte referenzieren. Ist für eine Transition kein Ereignis definiert, so ist nur die Erfüllung der angegebenen Randbedingungen zum Zustandsübergang notwendig.

Die Notation eines Zustandes erfolgt anhand eines Rechtecks mit abgerundeten Ecken, das neben dem Zustandsnamen optional die auszuführenden Aktivitäten mit dem kennzeichnenden Präfix zur Art der Ausführung enthält (siehe Abbildung 6.15). Davon abweichend ist der erforderliche Initialzustand als schwarzer Punkt darzustellen. Eine Transition hingegen wird als beschrifteter Pfeil modelliert, der einen Zustand mit dessen Folgezustand verbindet. Die Beschriftung enthält den Namen des auslösenden Triggers (Variablenname, Timerbezeichnung), der durch zusätzliche Argumente näher beschrieben werden kann, und ein optionales Symbol. Weiterhin sind eine eventuell vorhandene Überwachungsbedingung sowie die auszuführenden Aktivitäten anzufügen. Die Aufteilung in einzelne Regionen wird durch eine senkrechte Strichlinie gekennzeichnet, welche die zugehörigen Subgraphen voneinander trennt.

Bezüglich der Ausführung der Zustandsgraphen ist anzumerken, dass ein Zustandsübergang nur dann stattfindet, wenn das notwendige Ereignis eintritt und gleichzeitig alle hierzu erforderlichen Randbedingungen erfüllt sind. Dieser als „Schalten“ oder „Feuern“ der Transition bezeichnete Vorgang bleibt in alternativen Fällen aus und das aufrufende Ereignis geht verloren, auch wenn die notwendigen Bedingungen zu einem späteren Zeitpunkt erfüllt werden. Ist kein spezielles Ereignis für eine Transition definiert, so sind der übergeordnete Aufruf des Zustandsgraphen und die gleichzeitige Erfüllung der einschränkenden Bedingungen zum Zustandsübergang erforderlich. Obwohl eine Aufteilung des Zustandsgraphen in mehrere nebeneinander stehende Regionen als eine Spezifikation nebenläufigen Verhaltens zu betrachten ist, muss aufgrund der sequentiellen Arbeitsweise Speicherprogrammierbarer Steuerungen semantisch eine fortlaufende Abarbeitung von links nach rechts impliziert werden.

6.4.4 Modellierung des Verhaltens des physikalischen Teilsystems

Die Modellierung des Verhaltens einer Komponente des physikalischen Teilsystems mit Modelica ist prinzipiell in reiner Textform, ohne graphische Unterstützung, realisierbar. Indem die Verhaltensbeschreibung anhand eines Editors und unter Berücksichtigung der in der Sprachdefinition *(Modelica 2005)* festgelegten Grammatik und Semantik eingegeben wird, ist die vollständige Spezifikation des Systemverhaltens möglich. Diese Form der Modellierung entspricht jedoch nicht den in Abschnitt 5.5.3.1 dargelegten Anforderungen an eine domänenspezifische Modellierungssprache. Weiterhin erfordert eine solche Art der Modellbildung ein hohes Maß an Expertenwissen und eine exakte Kenntnis der Sprachdefinition. Daher und aufgrund der Tatsache, dass

für eine hinreichend präzise Verhaltensmodellierung nicht alle im Standard definierten Sprachelemente benötigt werden, wird der Einsatz eines geeigneten graphischen Editors und die Eingabe des Verhaltens in Form eines entsprechenden Diagramms vorgeschlagen. Die Beschreibung der hierzu notwendigen konkreten und abstrakten Syntax sowie die Erläuterung der semantischen Zusammenhänge zwischen den erforderlichen Sprachelementen ist den folgenden Abschnitten zu entnehmen.

Prinzipiell erfolgt die Modellierung des Verhaltens einer Komponente in einem oder mehreren Abschnitten *„PMSection“* (siehe Abbildung 6.16). Das Attribut *„SectionKind“* bestimmt, ob die untergeordneten Elemente dieses in deklarativer Form anhand einer Menge von Gleichungen *„PMEquationStatement“* oder alternativ als Algorithmen mittels einer Folge von Zuweisungen *„PMAssignmentStatement“* festlegen. Erstere sind flexibler, da sie keine Datenflussrichtung definieren[16], und daher insbesondere dann zweckmäßig, wenn das Verhalten mittels physikalischer Gesetzmäßigkeiten beschreibbar ist. Zuweisungen hingegen bestimmen sowohl die Auswertungsvorschrift als auch die Auswertungsreihenfolge eindeutig. Sie kommen deshalb vorwiegend dann zum Einsatz, wenn das Systemverhalten durch feste Abläufe definiert werden kann oder nicht durch Naturgesetze zu beschreiben ist. Dies ist beispielsweise bei Feldkomponenten mit einer immanenten Intelligenz der Fall, die in Abhängigkeit bestimmter Eingangsgrößen die Berechnung der Ausgangsgrößen nach einer definierten Berechnungsvorschrift vornehmen. Ergänzend ist für jeden Verhaltensabschnitt das Boolesche Attribut *„IsInitial“* zu definieren. Dieses erlaubt es, supplementäre Randbedingungen für das Verhalten einer Komponente festzulegen. Notwendig ist dies beispielsweise dann, wenn für die numerische Berechnung zusätzliche Anfangsbedingungen erforderlich sind. Die entsprechenden Abschnitte sind nur zu Beginn der Berechnung gültig und werden bei den nachfolgenden Berechnungsschritten nicht mehr berücksichtigt.

Neben Gleichungen oder Zuweisungen können die beschriebenen Verhaltensabschnitte analog zu vielen Programmiersprachen auch Kontrollstrukturen *„PMControlStructure“* enthalten. Diese erlauben es, anhand Boolescher Ausdrücke Randbedingungen zu definieren, die erfüllt werden müssen, damit die untergeordneten Gleichungen oder Zuweisungen als gültig zu betrachten sind. Welcher Art das entsprechende Konstrukt ist, wird durch das Attribut *„ControlOperator“* definiert. Gültig sind alle in der Sprachspezifikation festgelegten Typen. Es ist jedoch festzustellen, dass für den vorliegenden Ansatz hauptsächlich die konditionellen Elemente (PMIF, PMELSEIF, PMELSE) von Relevanz sind. Dies ist damit zu begründen, dass diese die wesentlichen Bausteine darstellen, um hybrides Systemverhalten zu spezifizieren. Damit lässt

[16] Die Modellierung mit Gleichungen (z. B. x + y = z) legt nicht fest, welche der Größen bekannt sind und welche berechnet werden müssen. Zuweisungen (z. B. x := z - y) hingegen definieren eindeutig, dass das Ergebnis der linken Seite durch Auswertung der rechten Seite zu bestimmen ist.

sich das Verhalten einer Komponente in einzelne, alternativ gültige Bereiche *„PMControlOperand“* unterteilen, wobei die durch Boolesche Ausdrücke zu definierenden Bedingungen bestimmen, welcher Bereich gerade gültig ist. Abbildung 6.18 zeigt dies beispielhaft anhand der vereinfachten Modellierung eines doppeltwirkenden Hydraulikzylinders, bei dem das Systemverhalten prinzipiell anders ist, wenn sich die Kolbenstange in einer der beiden Endlagen befindet oder an einer beliebigen Zwischenposition steht.

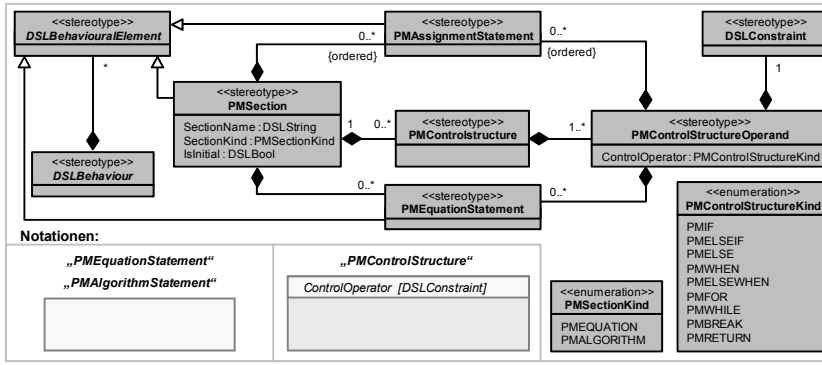


Abbildung 6.16: Metamodell und Notation des physikalischen Verhaltensdiagramms

Die konkrete Syntax zur Modellierung des Verhaltens des physikalischen Teilsystems orientiert sich im Sinne einer Vereinheitlichung der Modellierungstechnik stark an den in Abschnitt 6.4.3.2 beschriebenen Aktivitätsdiagrammen. Einzelne Abschnitte (Sections) werden in separaten Diagrammen visualisiert und optional mit einem aussagekräftigen Namen *„SectionName“* und dem Attribut *„IsInitial“* gekennzeichnet. Die Darstellung der Gleichungen oder Zuweisungen erfolgt durch untereinander angeordnete Rechtecke. Die gleiche Form der Notation gilt für Kontrollstrukturen mit ihren Operanden. Der Unterschied besteht darin, dass diese eine ergänzende Kopfzeile besitzen, die den Typ des Operators eindeutig ausweist und den Booleschen Ausdruck enthält, der zur Definition der Gültigkeit dient. Dessen Eingabe kann aber auch in graphischer Form geschehen (siehe Abbildung 6.18). Zur Erleichterung der Interpretation der Modelle und zur Verdeutlichung der physikalisch-logischen Hintergründe der Modellbildung ist es weiterhin sinnvoll, den unterschiedlichen Modellbestandteilen ergänzend informale Modellelemente zuzuordnen.

Für die Modellierung der Gleichungen und Zuweisungen[17] werden ungerichtete bzw. gerichtete Graphen als geeignet betrachtet. Deren Notation erfordert jedoch eine Reihe weiterer Modellelemente (siehe Abbildung 6.17).

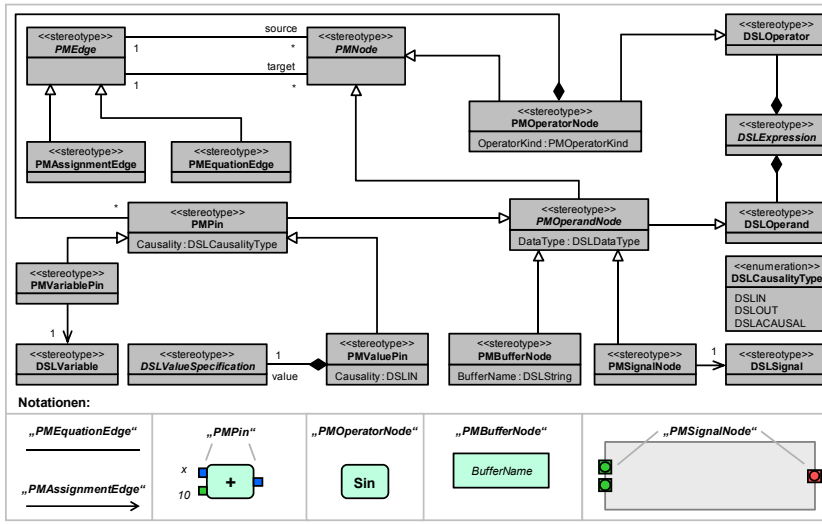


Abbildung 6.17: Metamodell und Notation von Gleichungen und Zuweisungen

Analog zu den Aktionen im Aktivitätsdiagramm wird das Konstrukt des Operatorknoten *„PMOperatorNode“* eingeführt, dessen Notation ebenfalls als Rechteck mit abgerundeten Ecken erfolgt. In Bezug auf die Semantik legt das Attribut *„OperatorKind“* (siehe Abschnitt 11.4.2) die Art des Operators fest, die der jeweilige Knoten beschreibt. Möglich sind typische arithmetische und Boolesche Operatoren, aber auch vordefinierte Funktionen (beispielsweise trigonometrische Funktionen), die Teil der Sprachdefinition von Modelica sind. Für die Kennzeichnung der Operatorart ist ein geeignetes Symbol zu verwenden.

Zur Modellierung der Operanden, die durch einen Operatorknoten verarbeitet werden können, wird das abstrakte Modellelement des Operandenknoten *„PMOperandNode“* definiert, das in Form mehrerer konkreter Ausführungen zur Modellierung von Gleichungen und Zuweisungen verwendet werden kann. Signalknoten *„PMSignalNode“* sind als Verweise auf die an den Ports der Komponente anliegenden Signale zu interpretieren. Um ihre Semantik auch optisch hervorzuheben, werden sie analog zu Signalports als Quadrate mit Kreis dargestellt und auf dem Rand der Zuweisung oder

[17] Prinzipiell sind Gleichungen und Zuweisungen als Einschränkungen zu betrachten, die durch Ausdrücke beschrieben werden.

Gleichung platziert, innerhalb derer sie als Operand benötigt werden. Zu ergänzen ist die Notation durch den Namen des Signals *„DSLSignalName“*, auf das verwiesen wird.

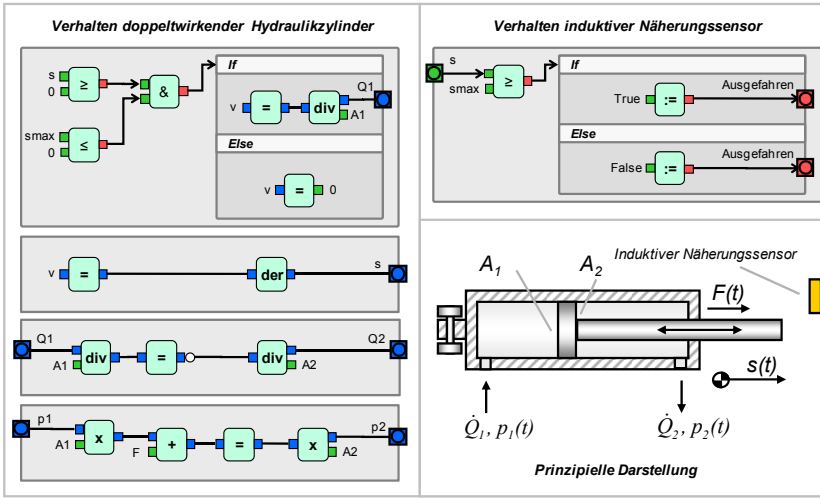


Abbildung 6.18: Beispiele eines physikalischen Verhaltensdiagramms

Pufferknoten *„PMBufferNode“* hingegen implementieren äquivalent zu den gleichnamigen Modellelementen von Aktivitäten temporäre Variablen, die nur während der Ausführung des Verhaltens gültig sind, dessen Teil sie sind. Aufgrund der Tatsache, dass Gleichungen keine temporären Größen enthalten können, ist diese spezielle Form der Operandenknoten nur zur Modellierung von Zuweisungen zu verwenden. Ihre Notation erfolgt genauso wie in Aktivitäten als Rechteck, das durch einen eindeutigen Namen gekennzeichnet ist.

Pins *„PMPin“* sind als graphische Repräsentation der zu verarbeitenden Größen und Ergebnisse der Operatorknoten zu betrachten. Sie verfügen wie die Pins in Aktionen über eine implizite Negationsfunktion (aktiv wenn Eigenschaftswert *„IsNegation“* den Wert *„TRUE“* annimmt, Notation als kleiner Kreis) und werden gleichermaßen als kleine Rechtecke direkt an die Operatorknoten angefügt. Eine weitere Verfeinerung definiert die beiden Subtypen *„PMVariablePin“* und *„PMValuePin“*. Während erstere direkt mit einer Variablen verknüpft werden, deklarieren zweitgenannte feste Zahlenwerte. Beiden Subtypen ist jedoch gemeinsam, dass der Name der Variablen oder der Zahlenwert direkt neben dem jeweiligen Pin annotiert wird. Ferner ist für jeden Pin der Tagged Value *„Causality“* anzugeben. Die zulässigen Werte sind zum einen davon abhängig, ob der besitzende Operatorknoten Teil einer Gleichung oder Zuweisung ist. Zum anderen sind aber auch die Art des Subtyps und die Eigenschaften des damit ver-

knüpften Modellelements von Relevanz. So können beispielsweise feste Werte, Konstanten oder Parameter nur den Kausalitätstyp *„DSLIN“* annehmen, da diese für die Auswertung der Gleichungen und Zuweisungen als bekannt vorausgesetzt werden. Abbildung 6.19 zeigt zusammenfassend eine Übersicht der möglichen Werte für alle definierten Arten von Pins.

		Gleichungen				Zuweisungen			
		PMVariablePin		PMValuePin	PMPin	PMVariablePin		PMValuePin	PMPin
		Variability = DSLCONSTANT PMPARAMETER	Variability = UNRESTRICTED			Variability = DSLCONSTANT PMPARAMETER	Variability = UNRESTRICTED		
Kausalität	DSLIN	✓	✗	✓	✗	✓	✓	✓	✓
	DSLOUT	✗	✗	✗	✗	✗	✓	✗	✓
	DSLACAUSAL	✗	✓	✗	✓	✗	✗	✗	✗

✓ möglich ✗ nicht möglich

Abbildung 6.19: Pinkausalität bei der Modellierung des physikalischen Verhaltens

Generell verfügt ein Operandenknoten über einen eindeutigen Datentyp *„DataType“*, der konform zum verwendeten Operatorknoten beziehungsweise dem Datentyp der referenzierten Signale, Variablen oder Werte sein muss. Beispielsweise kann ein Operatorknoten vom Typ *„PMDER“* (Ableitung nach der Zeit) lediglich kontinuierlich veränderbare Größen und damit nur solche des Datentyps Real verarbeiten.

Weiterhin ist allen Arten von Operandenknoten gemeinsam, dass sie durch Kanten *„PMEdge“* miteinander verknüpft werden. Diese formulieren das Zusammenwirken der einzelnen Operanden- und Operatorknoten. Es erfolgt generell eine Unterscheidung in Gleichungs- und Zuweisungskanten[18] *(„PMEquationEdge“* bzw. *„PMAssignmentEdge“*), wobei erstere aufgrund der prinzipiell deklarativen Form von Gleichungen keine Datenflussrichtung festlegen. Ihre Modellierung wird dem entsprechend in Form einer durchgehenden Linie umgesetzt, die einzelne Operandenknoten miteinander verbindet. Zweitgenannte hingegen definieren eine eindeutige Flussrichtung, die durch eine offene Pfeilspitze visualisiert wird. Für sie gilt analog zu den Aktivitäten ein Tokenkonzept, das die Ausführungsreihenfolge eindeutig festlegt. Zusätzlich zu den aufgezeigten semantischen und syntaktischen Rahmenbedingungen sind bei der Modellbildung und -interpretation ergänzende Regeln zu beachten, die durch die Sprachspezifikation von Modelica festgelegt werden. So ist beispielsweise die Reihen-

[18] Zuweisungskanten werden auch zur Modellierung der Gültigkeitsbedingungen von Kontrollstrukturen verwendet, da bei diesen analog den Zuweisungen die Auswertungsreihenfolge eindeutig definiert ist.

folge einzelner Gleichungen nicht relevant, während bei algorithmischen Modellabschnitten die Semantik aufgrund der kausalen Art von Zuweisungen von deren Ausführungssequenz abhängt. Deshalb ist bereits bei der Modellierung ein sequentielles Ausführen der geordnet untereinander dargestellten Zuweisungen zu implizieren. Eine tabellarische Zusammenfassung der wichtigsten supplementären Festlegungen ist Abschnitt 11.6 zu entnehmen. Bezüglich detaillierterer Ausführungen wird auf entsprechende Fachliteratur *(Modelica 2005; Fritzson 2004)* verwiesen. Die Umsetzung der erforderlichen Restriktionen ist soweit möglich durch entsprechende Engineeringwerkzeuge sicherzustellen bzw. zu unterstützen.

6.4.5 Interaktionen zur komponentenübergreifenden Modellierung

Während Aktivitäten, Zustandsautomaten, Gleichungen und Zuweisungen zur Abbildung des Verhaltens einzelner Komponenten eines Fertigungssystems eingesetzt werden, liegt der Fokus der Interaktionen auf der Spezifikation der logischen Reihenfolge und der Art des komponentenübergreifenden Informationsaustausches, der zur Erfüllung bestimmter Funktionen notwendig ist. Zu diesem Zweck wird eine angepasste Variante des Sequenzdiagramms der UML verwendet. Die zur Modellbildung notwendigen Sprachkonstrukte, deren Notation sowie die semantischen Grundlagen sind den folgenden Ausführungen zu entnehmen.

Ein Sequenzdiagramm (siehe Abschnitt 2.3.2) beschreibt eine Interaktion *„DSLInteraction“* (siehe Abbildung 6.20) anhand zweier zueinander orthogonaler Darstellungsdimensionen. Die vertikale Dimension entspricht einer Zeitachse, während die horizontale Dimension die Interaktionspartner abbildet, die an der Erfüllung der modellierten Funktion beteiligt sind. Deren Visualisierung erfolgt anhand sogenannter Lebenslinien *„DSLLifeline“*, die horizontal angeordnet und in Form eines Rechtecks (Lebenslinienkopf) sowie einer vertikal verlaufenden Linie notiert werden. Entgegen den in der UML festgelegten Konzepten repräsentieren Lebenslinien lediglich Komponenten *„DSLComponent“* des modellierten Systems und sind über die gesamte Zeitdauer der Interaktion gültig[19]. Der Name der Komponente, die durch eine Lebenslinie verkörpert wird, ist innerhalb des Kopfes der Lebenslinie zu notieren, wobei optional auch der zugeordnete Namensraum zu berücksichtigen ist.

Der Informationsaustausch zwischen einzelnen Komponenten wird anhand von Nachrichten *„DSLMessage“* modelliert, die in Form von Pfeilen zwischen den einzelnen Lebenslinien notiert werden. Eine Nachricht hat einen optional darstellbaren Namen *„MessageName“* und kann zusätzlich über eine endliche, geordnete Menge von Attributen *„DSLAttribute“* verfügen. Diese sind als eine Verallgemeinerung der in Ab-

[19] Aufgrund der in Abschnitt 5.5.2 beschriebenen Strukturinvarianz der Fertigungssysteme ist diese Vereinfachung als zulässig zu betrachten. Im Unterschied zur UML erfolgt daher die Modellierung auch nicht auf der Typ-, sondern der Objektebene.

schnitt 6.3.4 beschriebenen Signale und Parameter aufzufassen und können auf Variablen verweisen, die Bestandteil der sendenden oder empfangenden Lebenslinie bzw. der durch sie repräsentierten Komponente sind. Unabhängig vom Aktualwert der referenzierten Variablen ist für Attribute auch die Vorgabe von festen Werten möglich, die im Rahmen der Nachrichtenübermittlung eine höhere Priorität einnehmen. Die beschriebene Form der Kommunikation zwischen den einzelnen Komponenten des modellierten Systems ist damit als eine Abstraktion der bereits dargestellten Möglichkeiten zur Umsetzung der Wirkverbindungen anhand von Ports, Konnektoren, Signalen und Operationen zu betrachten.

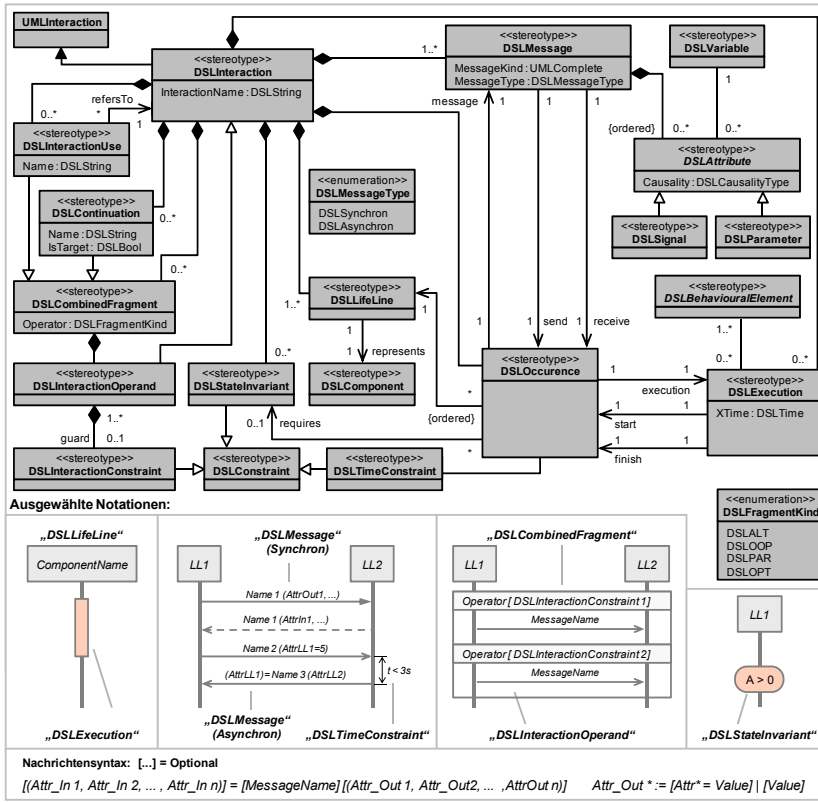


Abbildung 6.20: Metamodell und Notation von Interaktionen

Aufgrund der unterschiedlichen Eigenschaften der genannten Kommunikationsformen ist eine Klassifizierung in synchrone und asynchrone Nachrichten notwendig. Erstere sind dadurch gekennzeichnet, dass der Sender der Nachricht mit der weiteren Ausfüh-

rung des Verhaltens wartet, bis der Empfänger eine entsprechende Antwortnachricht zurückschickt. Folglich dienen synchrone Nachrichten zur Modellierung des Aufrufes komplexerer Verhaltenskonstrukte, die durch andere Interaktionspartner realisiert werden. Ihre Anwendung ist aufgrund der beschriebenen Eigenschaften der beiden Teilmodelle einzig und allein auf die Modellierung der Kommunikation zwischen Komponenten des Steuerungsmodells beschränkt. Obwohl auch der lesende und schreibende Zugriff auf Variablen im Steuerungsmodell aufgrund der prozeduralen Arbeitsweise der SPSen synchron erfolgt, wird hierfür im Sinne einer vereinfachten Modellierung eine asynchrone Kommunikationsform[20] postuliert. Die Darstellung synchroner Nachrichten erfolgt als Pfeile mit gefüllter Spitze, Antwortnachrichten und asynchrone Nachrichten werden hingegen konform zur UML als strichlierte bzw. durchgehende Linien mit offener Pfeilspitze notiert.

Für eine syntaktisch korrekte Modellierung komplexerer Nachrichten sind einige weitere Regeln zu beachten (siehe Abbildung 6.20). Ausgehende Attribute, die vom Sender zur Verfügung gestellt werden, sind geordnet innerhalb runder Klammern dem Namen der Nachricht nachzustellen. Die Namen der Parameter sollten hierbei den Namen der Variablen der sendenden Komponente entsprechen. Sollen diese im Rahmen der Nachrichtenübermittlung direkt Variablen des Nachrichtenempfängers zugewiesen werden, so sind letztere in Form entsprechend geordneter Attribute an der linken Seite des Nachrichtennamens darzustellen und von diesem durch ein Gleichheitszeichen zu trennen. Feste Vorgabewerte sind, ebenfalls durch einen Gleichheitsoperator getrennt, direkt dem Attribut folgend angebbar.

Das Senden oder Empfangen einer Nachricht auf einer Lebenslinie ist immer mit einem Sende- bzw. Empfangsereignis *„DSLOccurrence“* verbunden. Diese besitzen optional eine Notation in Form eines kleinen Quadrats[21] und stellen aus formaler Sicht das Bindeglied zwischen einer Nachricht und der sendenden/empfangenden Lebenslinie dar. Ferner sind Ereignisse auch mit sogenannten Ausführungsspezifikationen *„DSLExecution“* verknüpfbar. Eine Ausführungsspezifikation kann als eine Verbreiterung einer Lebenslinie annotiert werden und symbolisiert aus semantischer Sicht die Ausführung eines bestimmten Verhaltens durch die jeweilige Komponente. Auf das auszuführende Verhalten selbst wird lediglich verwiesen. Möglich sind Referenzen auf Funktionen, Sektionen, eine oder mehrere Zuweisungen oder übergeordnete Kontrollstrukturen, wenn die jeweilige Lebenslinie eine Komponente des physikalischen Teilsystems repräsentiert. Bei Komponenten des Steuerungssystems sind hingegen Verknüpfungen zu einzelnen Funktionen oder Aktivitäten möglich. Als ergänzendes Attri-

[20] Entsprechende Lese- bzw. Schreibzugriffe werden durch das Verhalten der aufrufenden Komponente selbst implementiert. Damit ist diese vereinfachte Form der Modellierung auch gerechtfertigt.

[21] Dadurch ist die Form des Informationsaustauschs über Ports und Schnittstellen optisch hervorhebbar.

but wird die aktuelle Zeitdauer *„XTime“* der Ausführung eingeführt, auf die auch die referenzierten Verhaltenskonstrukte Zugriff haben.

Zur Präzisierung und Formalisierung einer Interaktion kann eine Lebenslinie weiterhin über Zustandsinvarianten *„DSLStateInvariant“* verfügen. Diese stellen Zusicherungen dar, die zu einem bestimmten Zeitpunkt des Interaktionsablaufs erfüllt sein müssen, damit die weitere Ausführung überhaupt möglich ist. Solche Bedingungen können einen bestimmten Zustand eines oder auch mehrerer Interaktionspartner ausdrücken. Dem entsprechend wird eine Zustandsinvariante durch einen logischen Ausdruck beschrieben, der sich auf Variablen der besitzenden Lebenslinie[22] oder auf mittels Konnektoren erreichbare Variablen vom Typ *„DSLPUBLIC“* bezieht. Die Auswertung erfolgt zur Laufzeit vor dem Eintritt des folgenden Ereignisses. Zur Notation wird ein aussagekräftiger Name verwendet, den ein Rechteck mit abgerundeten Ecken umrandet. Die Platzierung der Zustandsinvariante erfolgt direkt auf der Lebenslinie oder sie wird alternativ über eine durchgehende Linie mit dieser verbunden.

Eine weitere Möglichkeit zur Spezifikation von Randbedingungen mit informalem Charakter für eine Interaktion existiert in Form von Zeitrestriktionen *„DSLTimeConstraint“* zwischen einzelnen Ereigniseintritten. Ihre Notation geschieht mittels kleinen horizontalen Linien, sogenannten Zeitmarken, und mittels vertikalen Doppelpfeilen mit offenen Spitzen. Der einschränkende Ausdruck wird an diese annotiert. Abbildung 6.21 zeigt eine beispielhafte Anwendung anhand eines Werkzeugwechsels. Hierbei wird die maximale Zeitdauer festgelegt, die einzelne Abschnitte des Ablaufs haben dürfen, ohne dass die im Pflichtenheft angegebene Gesamtdauer überschritten wird.

Zur Modellierung komplexer Kontrollstrukturen (Verzweigungen, Schleifen, Nebenläufigkeiten) und deren Steuerung wird das Konstrukt der kombinierten Fragmente *„DSLCombinedFragment“* definiert. Es wird dazu verwendet, um anhand eines speziellen Attributes, des sogenannten Interaktionsoperators *„Operator“*, alternative, parallele, optionale oder sich wiederholende Teile einer Interaktion zu spezifizieren. Je nach Art des Operators besteht ein kombiniertes Fragment aus einem oder mehreren Interaktionsoperanden *„DSLInteractionOperand“*, die als Rechtecke, die eine oder mehrere Lebenslinien überdecken, notiert werden. Der Typ des Operators wird üblicherweise durch eine entsprechende Kennzeichnung oberhalb des Operanden kenntlich gemacht (siehe Abbildung 6.20). Der Teil der Interaktion, der innerhalb eines Operanden steht, ist nur dann als gültig zu betrachten, wenn eine entsprechende Kontrollbedingung *„DSLInteractionConstraint“* erfüllt ist. Diese Bedingung wird in Form eines entsprechenden Booleschen Ausdruckes spezifiziert, wobei es Aufgabe der Modelliererin bzw. des Modellierers ist, entsprechend der Art des jeweiligen Operators ein de-

[22] Genauer die Variablen der Komponente, die durch die Lebenslinie repräsentiert wird.

terministisches Verhalten sicherzustellen. Die Notation des Ausdruckes erfolgt in Textform direkt neben dem Interaktionsoperator.

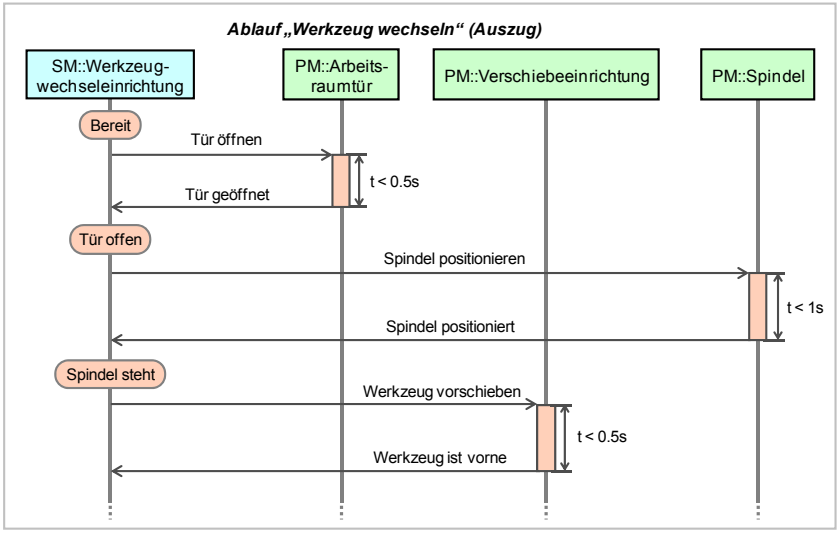


Abbildung 6.21: Beispiel eines Sequenzdiagramms

Da Sequenzdiagramme bei komplexen Abläufen relativ schnell unübersichtlich werden können, ist es sinnvoll, ergänzende Modellelemente zur Modularisierung einzuführen. Interaktionsreferenzen *„DSLInteractionUse“* dienen als Verweis auf weitere Interaktionen, die anhand eigenständiger Diagramme modelliert werden. Sie sind aus semantischer Sicht als ein eingebetteter Teil der Interaktion zu betrachten, der zur Laufzeit aufgelöst und abgearbeitet wird. Ihre Darstellung erfolgt analog zu den Interaktionsoperanden in Form eines Rechteckes, das eine oder mehrere Lebenslinien überdeckt und innerhalb dessen der Name der referenzierten Interaktion steht. Fortsetzungsmarken *„DSLContinuation“* hingegen erlauben die Modellierung von Sprüngen innerhalb eines Ablaufs anhand von Start- und Zielmarken. Zur Notation wird wie bei den Zustandsinvarianten ein Rechteck mit abgerundeten Ecken verwendet. Um eine eindeutige Unterscheidbarkeit sicherzustellen, empfiehlt sich eine toolspezifische, optische Kennzeichnung. Ergänzend zur in der UML festgelegten Differenzierung einzelner Marken anhand einer eindeutigen Bezeichnung wird das Attribut *„IsTarget“* definiert. Während Startmarken mehrfach vorkommen können, darf nur eine einzige Zielmarke gleichen Namens existieren, da ansonsten die Grundvoraussetzung zur Definition deterministischen Verhaltens fehlt.

6.5 Modelltransformationen

6.5.1 Allgemeines

Wie in Abschnitt 2.1 dargestellt, sind auch Transformationen, die Modelle oder deren Bausteine auf unterschiedlichen Abstraktionsebenen ineinander überführen, eine Stärke der modellgetriebenen Entwicklung. Inwieweit diese Strategie sinnvoll angewandt werden kann und wie eine konkrete Umsetzung im Detail zu bewerkstelligen ist, hängt wesentlich davon ab, welche Technologien im Anwendungsbereich eingesetzt werden, wie die domänenspezifische Modellierungssprache aufgebaut ist und welche weiteren Randbedingungen (z. B. Normen, Vorschriften, Ausbildung des Personals usw.) durch die spezifischen Gegebenheiten innerhalb einer bestimmten Domäne definiert werden. In den Abschnitten 2.1.4 bzw. 2.1.5 wurden mit dem Model-Driven-Architecture- und dem architekturzentrierten Ansatz exemplarisch zwei Methoden vorgestellt, die unterschiedlichen Kriterien gerecht werden, jedoch auch Schwächen aufweisen und nicht alle Fragestellungen der Entwicklung vollständig beantworten. Für den vorliegenden Anwendungsfall der Softwareerstellung für Fertigungssysteme wird daher ein eigener, pragmatischer Ansatz vorgestellt, der die spezifischen Rahmenbedingungen der Domäne berücksichtigt. Dieser verfügt zwar einerseits aus formaler Sicht durchaus über Verbesserungspotentiale, zeichnet sich andererseits jedoch durch seine praxisnahe Orientierung aus.

6.5.2 Eigenschaften des prinzipiellen Ansatzes

Eine nähere Betrachtung der in Abschnitt 5.4 definierten Ebenen der Modellbildung (präskriptive Modelle, deskriptive Modelle, Zielartefakte) zeigt, dass im Wesentlichen zwei Transformationen durchzuführen sind (siehe Abbildung 6.22).

Die erste Transformation hat die Aufgabe, die Informationen, die im Rahmen der präskriptiven Beschreibung der notwendigen Funktionalität anhand geeigneter Modellierungstechniken abgebildet wurden, in die Ebene der deskriptiven Modellbildung zu übertragen. Die zweite Stufe der Transformationen spezifiziert den Übergang von den deskriptiven Modellen zu den Zielartefakten. Sie definiert somit für das Steuerungsmodell den Übergang zu den Softwarebausteinen nach IEC 61131-3[23]. Für das physikalische Teilmodell ist eine Transformation in ein durch ein Simulationssystem verarbeitbares Modell[24] erforderlich.

[23] Alternativ auch in steuerungsspezifische Sprachen, falls diese von der IEC 61131-3 abweichen.

[24] Dies bedeutet eine Transformation des Modells in Modelica-Code. Alternativ ist prinzipiell bei kausalen Modellen auch eine Transformation in andere Zielformate möglich, sofern diese offengelegt sind.

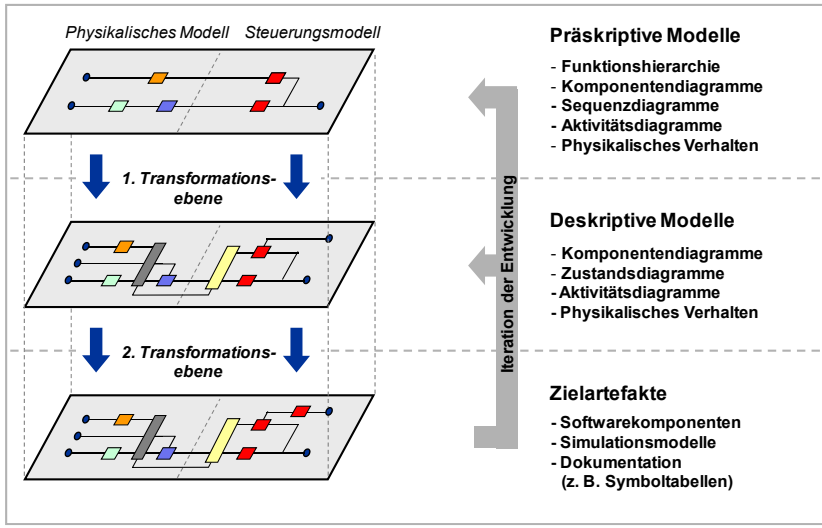


Abbildung 6.22: Transformationsebenen in der Entwicklung

Zu beachten ist, dass die Transformationen nicht bijektiv sind. Folglich ist eine Rückwärtstransformation im Allgemeinen nicht möglich. Werden daher im Sinne eines, bei der Entwicklung von Fertigungssystemen üblichen, iterativen Entwicklungsvorgehens Änderungen im Modell notwendig, so können diese nicht auf einer der untergeordneten Ebenen durchgeführt und anschließend automatisiert in die übergeordnete Ebene übertragen werden. Der Umstand, dass eine derartige Rückwärtsabbildung (Reverse Engineering) nicht möglich ist, wird häufig als Nachteil empfunden, da ein gewisser Mehraufwand notwendig ist, um die Modelle auf allen Ebenen konsistent zu halten. In Wirklichkeit ist diese Tatsache jedoch von Vorteil, da erstens eine Transformation eines Modells auf eine übergeordnete Ebene nicht zu einer Abstraktion der Funktionalität führt[25]. Zweitens wird das Entwicklungspersonal dazu gezwungen, sich intensiver mit den Modellen auseinanderzusetzen und nur teilweise durchdachte Erweiterungen durch Kopieren und Ändern bestehender Modellbestandteile werden erschwert.

Die Unterscheidung in Plattform-unabhängige Modelle (PIM) und Plattform-abhängige Modelle (PSM) sowie die Festlegung von Zeitpunkten im Entwicklungsablauf, an denen entsprechende Transformationen stattfinden, ist bei alternativen Anwendungsbereichen der modellgetriebenen Entwicklung üblicherweise einfacher als beim vorgestellten Ansatz. Dies ist darauf zurückzuführen, dass zum einen Speicher-

[25] Die Abstraktion ist eine kreative Aufgabe, die in sinnvoller Weise nur durch das Entwicklungspersonal und nicht durch vordefinierte Algorithmen durchführbar ist.

programmierbare Steuerungen als Zielsystem bereits vorausgesetzt werden und die Modellierung mittels Sprachelementen erfolgt, die bereits auf diese Rahmenbedingung zugeschnitten wurden. Andererseits können die Modellbausteine trotz der Beschränkung auf die IEC 61131-3 in verschiedene Zielsprachen (z. B. Funktionsbausteinsprache oder Anweisungsliste) transformiert werden. Weiterhin kann die Festlegung einer konkreten Programmiersprache und auch die technische Umsetzung der notwendigen Infrastruktur (Steuerung, Bussysteme, Feldkomponenten, usw.) im steuernden System bei Erfüllung der notwendigen Rahmenbedingungen (z. B. Kosten, Taktzeiten, Vorschriften) erst relativ spät im Entwicklungsprozess erfolgen. Dies ist beim gesteuerten System nicht der Fall. Bereits in der Konzeptphase wird festgelegt, wie die Umsetzung bestimmter Funktionen zu realisieren ist, indem entsprechende Prinziplösungen ausgewählt oder entwickelt werden (siehe Abschnitt 5.3). Dennoch konnte durch Untersuchungen gezeigt werden *(Herkommer 2003)*, dass im Hinblick auf die Implementierung steuerungstechnischer Aufgaben eine Betrachtung des Systems aus funktionaler Sicht ausreichend ist. Wird beispielsweise für eine Werkzeugwechselvorrichtung eine hydraulische Ansteuerung der einzelnen Bewegungsachsen als prinzipielle Lösung ausgewählt, so sind konkrete Details der Umsetzung (Abmessungen, Durchflussmengen, Systemdrücke, usw.) bei entsprechender Modellierung (parametrische Gestaltung der Bausteine des physikalischen Modells, konfigurierbare Softwarekomponenten) der notwendigen Funktionen durch die Modelle bereits abgedeckt. Die Unterscheidung in Plattform-spezifische und Plattform-unabhängige Modelle wird aufgrund der genannten Rahmenbedingungen nicht als zielführend angesehen. Vielmehr wird eine Unterscheidung in funktionale Modelle und Implementierungsmodelle, die sich durch eine konkrete Ausprägung eines funktionalen Modells auszeichnen, vorgeschlagen.

Für die Festlegung der Transformationen wird das Konzept der Markierung (siehe Abschnitt 2.1.4) als adäquat erachtet. Gegenüber der alternativen Möglichkeit zu deren Spezifikation anhand von Metamodellen ist diese Form der Umsetzung als wesentlich einfacher anwendbar und anpassbar sowie leichter verständlich und implementierbar zu betrachten. Prinzipiell werden bei der Markierung Modellelemente mit zusätzlichen Eigenschaften ausgezeichnet, die für die Transformatoren als Eingangsinformation dienen, um die anzuwendenden Transformationsregeln auszuwählen. Auf der Grundlage der vorhandenen Attribute entscheidet der jeweilige Transformationsalgorithmus zur Laufzeit, wie ein Modell oder ein Modellbestandteil zu interpretieren und für die nächste Modellebene aufzubereiten ist. Da noch keine allgemeingültige Sprache zur Beschreibung von Transformationen verfügbar ist *(Stahl & Voelter 2005, S. 23)*, ist eine Implementierung toolspezifisch durchzuführen. Hierbei ist es prinzipiell sinnvoll, eine zweistufige Vorgehensweise anzuwenden. Übergeordnet können Transformationen in Profilen zusammengefasst werden, welche die anzuwendenden Regeln unabhängig von bestimmten Modellen auf Typebene definieren. Damit muss nicht jeder Modellbaustein mit den entsprechenden Attributen versehen werden, sondern es wer-

den die generisch in einem bestimmten Profil definierten Regeln angewandt. Wird dagegen für explizite Modellbausteine die Anwendung anderer Vorschriften verlangt, so können den Modellelementen entsprechende Attribute zugewiesen werden, die mit einer erhöhten Priorität gehandhabt werden.

Grundsätzlich ist, genauso wie bei der Modellierung selbst, auch bei den Transformationen zwischen Struktur und Verhalten zu unterscheiden. Es gilt also zum einen Regeln zu definieren, die Transformationen auf der strukturellen Ebene beschreiben. Zum anderen sind die Möglichkeiten zur sinnvollen Verwertung der Verhaltenskonstrukte einzelner Modellbausteine zu spezifizieren. Die folgenden Ausführungen zu Transformationen zwischen den festgelegten Abstraktionsebenen beruhen auf Erfahrungen und Untersuchungen bei der Entwicklung von Fertigungssystemen beziehungsweise untergeordneten Teilsystemen. Alternativ sind sicherlich abgewandelte Möglichkeiten zur Modelltransformation anwendbar. Deshalb erheben die folgenden Betrachtungen auch keinen Anspruch auf Vollständigkeit oder eine allgemeine Gültigkeit, sondern beschreiben lediglich eine Variante einer Umsetzung und die dabei zu beachtenden Randbedingungen.

6.5.3 Transformation der präskriptiven Modelle

Die Ebene der präskriptiven Modelle (siehe Abschnitt 5.4) dient dazu, die umzusetzenden Funktionalitäten im interdisziplinären Team auf einem relativ abstrakten Niveau, ohne eine explizite, detaillierte Betrachtung der softwaretechnischen und technologischen Umsetzung, zu beschreiben. Im Fokus stehen die wesentliche Funktionalität und die Interaktion einzelner Bausteine des Systems. Dieser Teil der Systemkonzeption hat zum einen die Aufgabe, die im Pflichtenheft festgelegten Anforderungen auf eine Menge erforderlicher Funktionen abzubilden und diese in Form einer Funktionshierarchie (Funktionsgruppen, Funktionsuntergruppen und Funktionseinheiten) zu gliedern. Zum anderen erfolgt die Festlegung der wesentlichen Systemstruktur im interdisziplinären Team, wobei die in Abschnitt 5.2 beschriebenen Prinzipien der Modellbildung sowie übergeordnete Zielsetzungen (z. B. modulare Strukturierung in Form eines Baukastensystems, Restriktionen im Pflichtenheft) zu berücksichtigen sind. Die Funktionen können nun einzelnen Strukturbausteinen zugeordnet und durch Attribute, welche die Eigenschaften der jeweiligen Komponente kennzeichnen, ergänzt werden. Mit der Definition der signifikanten Abläufe anhand von Sequenzdiagrammen und der präziseren Beschreibung einzelner Funktionen mittels Zuweisungen, Aktivitäten usw. wird die präskriptive Modellierung abgeschlossen. Die Aufgabe der Transformation besteht somit darin, die in den entsprechenden Modellbausteinen enthaltenen Informationen in die Ebene der deskriptiven Modellierung zu übertragen und für die dort eingesetzten Modellierungstechniken nutzbar zu machen.

Auf der strukturellen Ebene ist diese Transformation nur eingeschränkt notwendig, da die aufgebaute Struktur vollständig übernommen werden kann. Das in der präskripti-

ven Phase in Form eines Komponentendiagramms aufgebaute Modell bleibt auch auf der Ebene der deskriptiven Modellbildung erhalten und wird im Rahmen der weiteren Entwicklung verfeinert. Somit können auch die den einzelnen Komponenten zugeordneten Funktionen und Attribute übernommen werden. Durch die Analyse der Kommunikationspfade innerhalb der modellierten Interaktionen kann weiterhin eine Ergänzung der Systemstruktur erfolgen. Die Prüfung umfasst sowohl die Untersuchung des Informationsaustausches mittels Nachrichten als auch der Verweise auf Attribute anderer Systemkomponenten innerhalb von Zustandsinvarianten. Ist für einen bestimmten Kommunikationspfad noch keine strukturelle Verbindung mittels Ports und Konnektoren definiert, so kann dieser Modellbildungsschritt automatisiert ablaufen. Existiert hingegen eine Verbindung, so gilt es, die den Ports zugeordneten Schnittstellen um konforme Signale bzw. Operationen zu ergänzen.

Die Transformation des Verhaltens gestaltet sich hingegen komplexer. Während Zuweisungen oder Aktivitäten, die im Rahmen der präskriptiven Modellierung eingesetzt werden, direkt in die deskriptive Ebene übernehmen werden können, existieren auf der koordinierenden Ebene der Interaktionen mehrere Möglichkeiten zur Interpretation bestimmter Modellkonstrukte. Zum einen kann das Verhalten eines Interaktionspartners sowohl in Elemente eines Zustandsautomaten oder einer koordinierenden Aktivität transformiert werden. Zum anderen sind die einzelnen Sprachbestandteile teilweise unterschiedlich interpretierbar. So kann etwa eine Zustandsinvariante als Zustand eines entsprechenden Diagramms oder aber auch als Überwachungsbedingung einer Transition übersetzt werden. Die Transformation von Nachrichten wiederum ist davon abhängig, zwischen welcher Art von Modellkomponenten (Steuerungsmodell, physikalisches Teilmodell) die Kommunikation stattfindet, da entsprechend unterschiedliche Rahmenbedingungen zu beachten sind.

In Abbildung 6.23 sind einige wesentliche, anwendbare Transformationen, die für Modellelemente von Sequenzdiagrammen möglich sind, beschrieben. Eine Auswahl weiterer möglicher Transformationen ist Abschnitt 11.5 zu entnehmen. Welche Form der Interpretation letztlich für die Transformation eines bestimmten Modellelements aktiviert wird, ist durch das Setzen entsprechender Markierungen festzulegen.

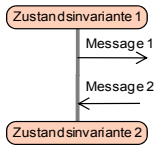
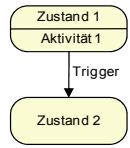
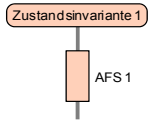
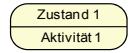
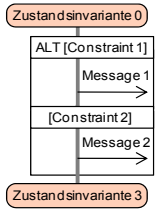
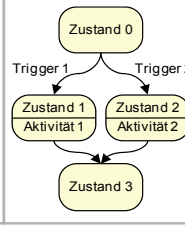
Ausgangselement	Zielelement	Beschreibung
		Die Zustandsinvarianten werden als Zustände interpretiert. Ausgehende Nachrichten, die einer Zustandsinvariante nachfolgen, werden als Aktivitätsaufrufe übersetzt und mit einer entsprechenden Operation der Schnittstelle verknüpft, falls das Ziel Teil des Steuerungsmodells ist. Im Alternativfall wird eine Verknüpfung mit einem Signal hergestellt, das den gleichen Namen wie die Nachricht trägt. Eingehende Nachrichten werden als Bedingung für eine ausgehende Transition interpretiert. Ein eventuell vorhandener Boolescher Ausdruck, der die Zustandsinvariante beschreibt, wird als Randbedingung (Guard) für das Feuern derjenigen Transitionen eingefügt, die zum Zustand hinführen.
		Ausführungsspezifikationen werden als Aktivitäten interpretiert. Existiert unmittelbar vorher in der Trace bereits eine Zustandsinvariante, so wird diese in einen Zustand transformiert, dem die Aktivität zugeordnet wird. Alternativ wird ein neuer Zustand erzeugt. Ist die Ausführungsspezifikation nicht bereits mit einer Aktivität verknüpft, wird ein entsprechendes Grundgerüst angelegt.
		Kombinierte Fragmente mit einem Operator vom Typ „DSLALT" (Alternative Verhaltensausführung) werden durch eine Verzweigung im Zustandsgraphen modelliert. Die einzelnen Operanden werden in Form separater Teilgraphen abgebildet. Die für den jeweiligen Operator gültige Kontrollbedingung wird in Form eines Guards und einer entsprechenden Transition übersetzt. Die abschließende Zusammenführung der alternativen Teilabläufe erfolgt durch Transitionen, die einen gemeinsamen Zielzustand aufweisen.

Abbildung 6.23: Möglichkeiten zur Transformation präskriptiver Modelle

Ergänzend ist festzustellen, dass nach der Transformation der Modelle auf die deskriptive Ebene das Systemverhalten nicht vollständig beschrieben wird und durchaus auch widersprüchliche und unvollständige Modelle existieren können. Einzelne Sequenzdiagramme und das ihnen inhärente Verhalten bestimmter Komponenten beschreiben lediglich Ausschnitte des Gesamtverhaltens, die mehr oder weniger eng miteinander verbunden sind. Die Aufgabe der Entwicklerinnen und Entwickler ist es daher, die generierten Modellkonstrukte in sinnvoller Weise so zu verknüpfen, dass das in der deskriptiven Phase definierte Verhalten erfüllt, Widersprüche aufgelöst, gültige Modelle gebildet und diese gegebenenfalls vervollständigt bzw. erweitert werden.

6.5.4 Transformation der deskriptiven Modelle

Auf der Ebene der deskriptiven Modelle werden die Funktionalitäten des Fertigungssystems im Detail beschrieben. Hierzu kommen für das Steuerungsmodell Aktivitäts- und Zustandsdiagramme zum Einsatz. Die Spezifikation des Verhaltens des physikalischen Teilsystems erfolgt mittels Gleichungen und Zuweisungen sowie übergeordneten Kontrollstrukturen. Ziel ist es, das Verhalten der einzelnen Komponenten eindeutig und so präzise wie notwendig zu beschreiben, dabei jedoch im Sinne einer funktiona-

len Systemmodellierung soweit möglich unabhängig von der konkreten Umsetzung zu bleiben.

In Bezug auf die Transformation der deskriptiven Modelle muss prinzipiell zwischen dem physikalischen Teilmodell und dem Steuerungsmodell unterschieden werden. Für ersteres gilt es, das abgebildete Verhalten und die Struktur in Modelica zu überführen. Da der zur Modellbildung verwendete Diagrammtyp auf dieser Sprache aufbaut, besteht die Aufgabe im Wesentlichen darin, die graphischen Konstrukte auszuwerten und in entsprechende Anweisungen zu übersetzen. Ergänzend ist aber auch eine Gültigkeitsprüfung des generierten Gesamtmodells durchzuführen. Dies ist erforderlich, da aufgrund der Konvertierung in eine Zustandsraumdarstellung, die für Simulationszwecke notwendig ist, insbesondere bei akausalen Modellen nicht zulässige Singularitäten auftreten können. Soll eine Transformation in zu Modelica alternative Modellierungssprachen erfolgen, so wird prinzipiell eine nachfolgende, sekundäre Transformation empfohlen. Manche kommerzielle Systeme, wie beispielsweise MATLAB/Simulink, bieten hierzu bereits entsprechende Schnittstellen an. Aufgrund der Tatsache, dass diese Transformationen die Kenntnis der den einzelnen Systemen inhärenten Modellformate erfordern, diese in der Praxis aber häufig proprietär und nicht zugänglich sind, wird auf diesbezüglich weiterführende Ausführungen im Rahmen der Arbeit verzichtet. Ein ausführlicher Vergleich von Modelica und MATLAB sowie eine detaillierte Beschreibung der Schnittstelle zwischen beiden genannten Systemen ist *(Richert u. a. 2003)* bzw. *(Martin u. a. 2005)* zu entnehmen.

Bei der Transformation des Steuerungsmodells in die Zielartefakte[26] der Softwareentwicklung sind technologische Rahmenbedingungen, wie beispielsweise die Verteilung der Ein- und Ausgänge auf entsprechende Module der Steuerungshardware, die dazu konforme Zuweisung von Datentypen oder die Verteilung der Softwarekomponenten auf unterschiedliche Steuerungen bzw. Tasks zu berücksichtigen. Obwohl die UML mit dem Verteilungsdiagramm prinzipiell eine Möglichkeit zur Beschreibung der Steuerungshardware und der Kommunikationsverbindungen (Bussysteme, dezentrale Komponenten, usw.) zur Verfügung stellt, wird eine praxisübliche Umsetzung mit herstellerspezifischen Softwarewerkzeugen zur Hardwareprojektierung als zielführend angesehen. Diese sind speziell auf die Funktionalität der eingesetzten Komponenten abgestimmt und bieten neben umfangreichen Bibliotheken auch einfache Möglichkeiten zum Setzen baugruppenspezifischer Parameter und Eigenschaften. Die weitgehende Entkoppelung von der Softwarefunktionalität und damit die Möglichkeit zur unabhängigen Projektierung unterstützt die beschriebene Vorgehensweise.

Zur Vermeidung der Mehrfachgenerierung von Entwicklungsdaten und Inkonsistenzen wird die Nutzung der verfügbaren Schnittstellen zum Datenaustausch als geeignet be-

[26] In der modellgetriebenen Softwareentwicklung wird üblicherweise auch der generierte Zielcode als ein Modell verstanden. Die Codegenerierung ist damit als eine Transformation zu betrachten.

trachtet. So kann beispielsweise die Zuordnung der Adressbereiche zu den eingesetzten Feldkomponenten oder Ein- bzw. Ausgabebaugruppen in den herstellerspezifischen Werkzeugen erfolgen. Durch den Export der Hardwarekonfiguration in Form eines offenen Beschreibungsformates[27] und das Verarbeiten durch einen geeigneten Parser können die projektierten Adressbereiche den Signalen zwischen dem physikalischen Modell und dem Steuerungsmodell zugeordnet und auf die Konformität mit den verwendeten Datentypen geprüft werden. Umgekehrt können anhand der Adresszuordnungen sowie der Signalnamen Symboltabellen automatisiert generiert werden.

In Bezug auf die Transformation des Steuerungsmodells in Softwarebausteine erweist sich vor allem die Anpassung an das gewünschte Zielsystem als problematisch. Es ist festzustellen, dass kommerziell verfügbare Steuerungssysteme nicht alle Eigenschaften und Möglichkeiten der IEC 61131-3 unterstützen[28], so dass diese Tatsache bei der Generierung der Zielartefakte ebenfalls zu berücksichtigen ist. Darüber hinaus sind bestimmte Namenskonventionen für eine einwandfreie Funktion erforderlich oder aufgrund vorgeschriebener Konventionen[29] notwendig. Als Beispiel für die genannten Einschränkungen seien zum einen die in der IEC 61131-3 definierten 64-Bit-Datentypen LWORD, LINT und LREAL genannt, die von manchen Steuerungsherstellern nicht unterstützt werden und somit den zulässigen Wertebereich von Variablen begrenzen. Eine weitere, in der Praxis häufig auftretende Problematik ist durch die Adressierung der höherwertigen Datentypen mit mehr als 8 Bit gegeben. Manche Steuerungen vertauschen den Adressbereich der zugehörigen Speicherbereiche[30], wodurch Zugriffsprobleme auftreten können.

Zusammenfassend kann festgestellt werden, dass Transformationen in Steuerungsbausteine trotz der Eindeutigkeit der IEC 61131-3 aufgrund der praktischen Randbedingungen an das jeweils verwendete Zielsystem anzupassen sind. Transformationen in Steuerungscode sind daher trotz der Normung der Programmiersprachen nicht generell beschreibbar. Aufgrund der damit verbundenen Komplexität wird diese Thematik im Rahmen der vorliegenden Arbeit nicht näher untersucht. Kapitel 7 geht jedoch anhand eines Anwendungsbeispiels auf eine exemplarische Umsetzung ein und verweist ergänzend auf Arbeiten, die Lösungen für diese Problemstellung bieten.

[27] Steuerungshersteller bieten hierzu meist entsprechende Textformate an. Teilweise existieren auch übergeordnete Beschreibungen (z. B. GSD-Dateien der Profibus-Nutzerorganisation zur Spezifikation von Teilnehmern in Profibus-Feldbussen).

[28] Die internationale Organisation PLCOpen definiert hierzu mehrere Konformitätslevels.

[29] Viele Hersteller von Fertigungssystemen definieren hierfür Regeln, um die Wartbarkeit der Software zu erleichtern und den Steuerungscode bestimmten Baugruppen, Komponenten oder Funktionen zuordnen zu können.

[30] So ist etwa die Aufteilung einer 16-Bit-Variablen in High Byte und Low Byte bei einer Siemens-SPS genau umgekehrt gegenüber einer Soft-SPS mit Intel 80x86 CPU.

6.6 Zusammenfassung

In Kapitel 6 wurde eine domänenspezifische Sprache zur modellgetriebenen Softwareentwicklung für automatisierte Fertigungssysteme vorgestellt. Diese umfasst grundlegende Sprachelemente sowie eine Reihe von Modellkonstrukten und Diagrammen zur Spezifikation der Struktur und des Verhaltens von automatisierten Fertigungssystemen. Die jeweils eingesetzten Bausteine der DSL wurden im Detail in ihrer syntaktischen Darstellung und ihrer Semantik erläutert. Darüber hinaus wurde die Problematik der Modelltransformationen diskutiert, ein geeignetes Konzept zu deren Umsetzung vorgeschlagen und auf die hierbei zu beachtenden Rahmenbedingungen eingegangen. Die konkrete Anwendung der Sprache sowie praxisbezogene Fragestellungen bezüglich des in Kapitel 5 beschriebenen Rahmenkonzeptes werden im Folgenden anhand von Beispielen im betrachteten Umfeld erläutert.

7 Anwendung der modellgetriebenen Softwareentwicklung

7.1 Übersicht

Der Einsatz der vorgestellten domänenspezifischen Modellierungssprache und der zugrunde gelegten Vorgehensweise wird im Folgenden anhand von Auszügen aus ausgewählten Praxisbeispielen erläutert. Dabei liegt der Schwerpunkt nicht nur auf der Anwendung der einzelnen Diagrammtypen und Sprachelemente. Im Besonderen wird der Fokus auf die Darstellung von Erfahrungen beim praxisnahen Einsatz gelegt. Die Gliederung erfolgt im Wesentlichen anhand der in Abschnitt 5.3 definierten Prozessbausteine bzw. der diesen jeweils zugrunde gelegten Abstraktionsebenen der Modellbildung und der hierfür vorgeschlagenen Modellierungstechniken. Da der schwerpunktmäßige Einsatz der einzelnen Techniken stark von der Art des Entwicklungsprojektes abhängt, wird die Anwendung zum einen anhand der Neukonstruktion einer Reibschweißanlage gezeigt. Zum anderen wird eine Variantenkonstruktion der Werkzeugwechseleinrichtung eines Fräsbearbeitungszentrums betrachtet.

Als problematisch für die praxisnahe Anwendung der modellgetriebenen Softwareentwicklung gestaltet sich die mangelnde Verfügbarkeit einer Entwicklungsumgebung, die den gestellten Anforderungen in hinreichender Weise genügt. UML-Modellierungswerkzeuge weisen hierfür lediglich ein eingeschränktes Potential auf, da sie nur bedingt für eine integrative Modellbildung des steuernden und des gesteuerten Systems geeignet sind. Ferner ist nur eine beschränkte Anpassbarkeit der graphischen Modellkonstrukte gegeben. Die Fähigkeit zur interaktiven Simulation des abgebildeten Systemverhaltens ist ebenfalls noch nicht in hinreichender Qualität verfügbar. Kommerzielle Simulationssysteme zur Abbildung des Systemverhaltens im Zeitbereich haben hingegen nur unzureichende Möglichkeiten zur Anpassung der graphischen Beschreibungsmittel, weisen darüber hinaus Schwächen im Hinblick auf die Verwendung allgemeingültiger und offener Beschreibungssprachen auf und stellen in der Regel nur Teile der prinzipiell notwendigen Modellkonstrukte bereit. Daher wurde im Rahmen der Arbeit ein Prototyp einer integrierten Entwicklungsumgebung für die modellgetriebene Softwareentwicklung konzipiert und umgesetzt. Dieser bietet alle notwendigen Editoren zur Modellierung und verfügt über Möglichkeiten zur Simulation des Systemverhaltens präskriptiver und eingeschränkt deskriptiver Modelle. Der Aufwand zur Umsetzung der Anforderungen im Hinblick auf die Simulation des Systemverhaltens in späteren Entwicklungsphasen, insbesondere der Inbetriebnahmephase, erfordert den ergänzenden Einsatz hierfür geeigneter Systeme. Daher wurden auch ausgewählte, kommerziell verfügbare Simulationssysteme an das vorgeschlagene Konzept adaptiert und in den Entwicklungsprozess integriert. Die entstehenden Brüche in der Durchgängigkeit der Entwicklungsdaten wurden soweit als möglich vermieden. Dennoch sind diesbezüglich und im Hinblick auf einen praxisnahen Einsatz noch weitere Optimie-

rungen, insbesondere in Bezug auf die Handhabung und ergonomische Gestaltung der Modellbildung notwendig.

7.2 Beschreibung der Anwendungsbeispiele

Die Schaufelräder moderner Flugtriebwerke werden aufgrund der steigenden Anforderungen in Bezug auf das Schub/Masse-Verhältnis und die Herstellkosten in „Blisk[1]-Bauweise“ gefertigt. Das Zusammensetzen einzelner Schaufelräder zu Triebwerksstufen erfolgt mittels Reibschweißen. Aufgrund der hohen Anforderungen an die Genauigkeit und da eine Nachbearbeitung nur bedingt möglich ist, wurde im Rahmen der Entwicklung einer neuen Anlage für das Schwungrad-Reibschweißen (siehe Abbildung 7.1) ein System zur präzisen, koaxialen Ausrichtung der zu schweißenden Teile integriert. Dieses besteht im Wesentlichen aus einem automatisierten Messsystem zum Vermessen der zu verschweißenden Bauteile sowie aus einer Korrektureinrichtung, die eine exakte Ausrichtung der Teile zueinander vor Beginn des Schweißprozesses erlaubt.

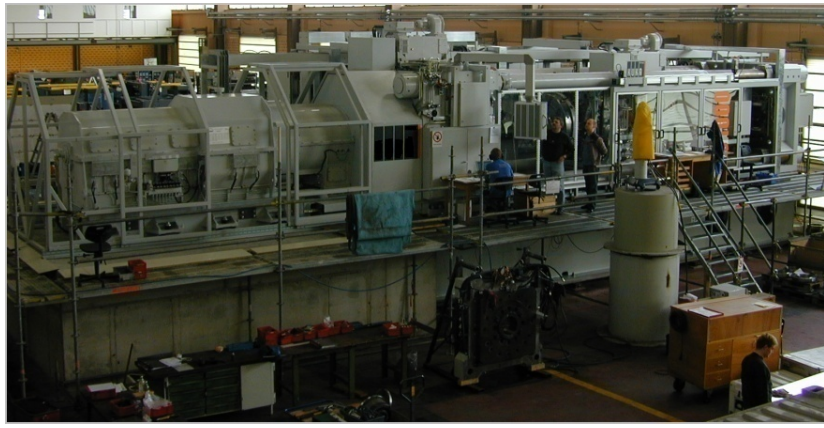


Abbildung 7.1: Reibschweißanlage zu Herstellung von Flugzeugtriebwerken

Die Aufgabe bestand in der Konzeption und Detaillierung, der Umsetzung und der Inbetriebnahme der Korrektur- und Messeinrichtung. Die Durchführung der Arbeiten erfolgte parallel zur Entwicklung der Reibschweißanlage.

Im Rahmen mehrerer studentischer Arbeiten wurden weiterhin ein alternativer Werkzeugwechsler sowie ein dazugehöriges Werkzeugmagazin für ein bestehendes Fräsbearbeitungszentrum entwickelt. Die Aufgabe bestand darin, sowohl die Konzeption und

[1] Blisk = “Blade Integrated Disk (Turbinenschaufeln und Rotor werden aus einem Teil gefertigt).

Konstruktion der mechanischen Komponenten durchzuführen als auch die notwendige Steuerungssoftware zu entwerfen und umzusetzen.

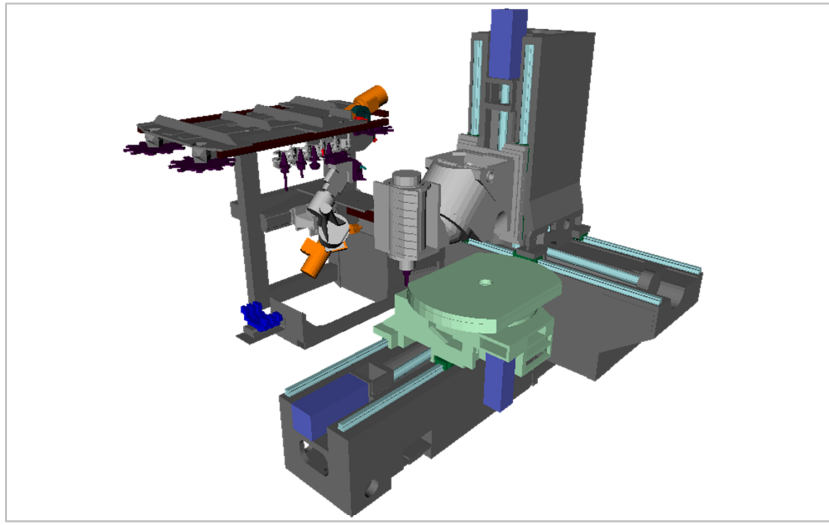


Abbildung 7.2: Modell eines Fräsbearbeitungszentrums

Die Inbetriebnahme und Optimierung der Software wurde anhand der erstellten virtuellen Modelle realisiert. Auf die reale Umsetzung des Werkzeugwechslers wurde aus Kostengründen verzichtet.

7.3 Aufbau des Systems zur modellgetriebenen Softwareentwicklung

Abbildung 7.3 zeigt die prinzipielle Systemarchitektur der implementierten Umgebung zur modellgetriebenen Softwareentwicklung. Diese besteht im Wesentlichen aus vier Modulen, die im koordinierten Zusammenspiel die notwendige Funktionalität zur Verfügung stellen. Das Teilsystem für die Modellbildung bietet Werkzeuge zum Aufbau der Systemstruktur und des Systemverhaltens. Es umfasst Editoren für alle in Kapitel 6 vorgestellten Diagramme und Modellkonstrukte. Der Bereich „Simulation“ hingegen stellt Funktionen zur Simulation präskriptiver und deskriptiver Modelle bereit. Das Modul „3D-Visualisierung“ besteht im Wesentlichen aus einer Bibliothek und einer Oberfläche zur dreidimensionalen Darstellung der Fertigungssysteme oder ihrer Komponenten. Es dient der anschaulichen Diskussion der zu entwickelnden Funktionen und Abläufe und erleichtert das Systemverständnis. Weiter bietet es Funktionen zur Interaktion mit dem Modell und visualisiert das Verhalten im Rahmen der Modellsimulation. Ferner verfügt das System auch über exemplarisch implementierte Funktionen zur Modelltransformation und über Hilfsmittel zur Verwaltung der Modelle. Hier-

zu gehört unter anderem eine Modellbibliothek, die es erlaubt, vordefinierte Teilmodelle im Rahmen der Modellbildung zu verwenden und somit den Aufwand zur Systemmodellierung zu minimieren.

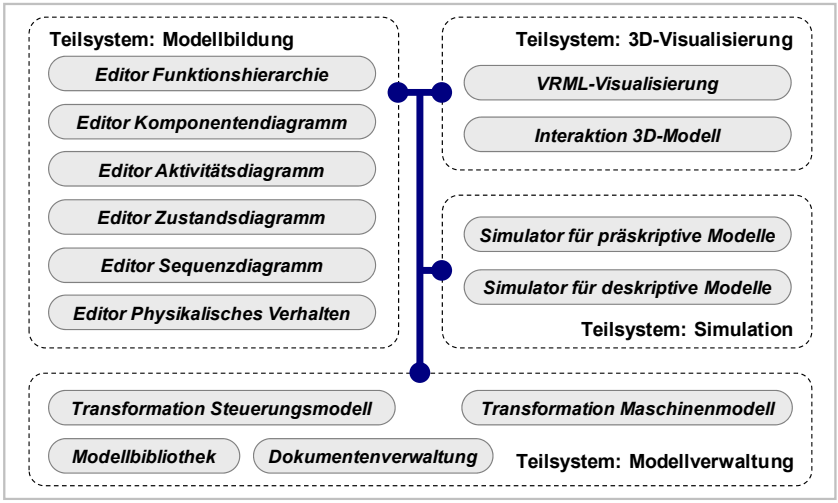


Abbildung 7.3: Aufbau des Systems zur modellgetriebenen Softwareentwicklung

7.4 Klären der Aufgabenstellung

Die Phase der Aufgabenklärung umfasst, wie in Abschnitt 5.3 beschrieben, im Wesentlichen die Definition der Entwicklungsziele in Bezug auf die Leistungsfähigkeit, die Kosten, den Funktionsumfang und sonstige Rahmenbedingungen. Die mit dem Kunden[2] üblicherweise in Form eines Lastenheftes festgelegten Aufgaben werden auf ihre prinzipielle Machbarkeit hin geprüft und in ein Pflichtenheft umgesetzt, das die Grundlage für die weiteren Entwicklungsschritte darstellt. Die Erfassung der Anforderungen erfolgt in der Praxis immer noch in Form von Textdokumenten mit vorgegebener Struktur. Es existieren zwar auch semiformale Mittel zur Aufnahme und Strukturierung der Anforderungen[3], diese sind jedoch nicht stark verbreitet. Da die Klärung der Aufgabenstellung nicht Schwerpunkt der Arbeit ist (siehe Abschnitt 3.3) , wird die Phase der Anforderungserfassung im Folgenden auch nicht weiter betrachtet. Eine Dokumentation der Aufgabenstellung in informaler oder semiformaler Form stellt die

[2] Unter Kunden sind auch interne Organisationseinheiten (z. B. das Produktmanagement im Serienmaschinenbau) und externe Partner (z. B. im Anlagenbau) zu verstehen.

[3] Ein Beispiel hierfür ist das in Abschnitt 2.3.7 beschriebene Anforderungsdiagramm der Systems Modeling Language.

Ausgangsbasis für die modellgetriebene Softwareentwicklung dar. Weitere diesbezügliche, speziell für die Entwicklung automatisierter Systeme relevante Informationen sind *(VDI/VDE 3683)* und *(VDI/VDE 3694)* zu entnehmen.

7.5 Funktionale Phase der Systemkonzeption

Die funktionale Phase der Systemkonzeption ist vor allem bei Neuentwicklungen und im Sondermaschinenbau von herausragender Bedeutung. Sie stellt den ersten Schritt im ingenieursmäßigen Vorgehen zur Umsetzung der Anforderungen in ein komplexes technisches System dar. Aufgabe ist es, die in Form des Pflichtenheftes festgelegten Ziele auf eine Menge notwendiger Funktionen abzubilden, zu beschreiben und zu strukturieren. Während dieser Schritt für einfache Entwicklungsaufgaben mit geringer Komplexität nicht zwingend notwendig ist, erleichtert die resultierende Funktionshierarchie bei komplexeren Aufgabenstellungen den Übergang von der relativ abstrakten Ebene der Anforderungen hin zur technischen Umsetzung. Die Beherrschung der zugrunde liegenden Komplexität moderner Werkzeugmaschinen und Fertigungssysteme wird damit wesentlich unterstützt. Abbildung 7.4 zeigt den implementierten Editor zum Aufbau einer Funktionshierarchie. Die Darstellung visualisiert eine Funktionshierarchie des Mess- und des Justagesystems der Reibschweißanlage.

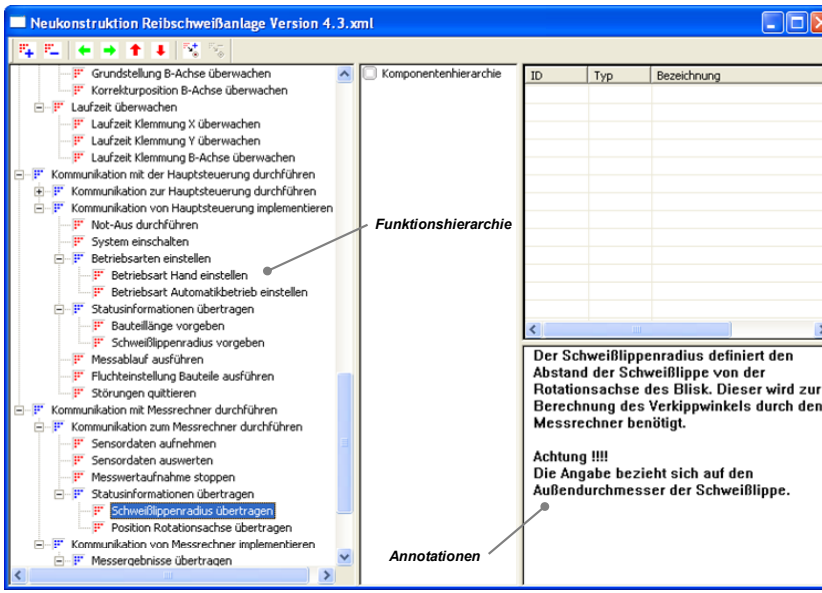


Abbildung 7.4: Editor zum Aufbau von Funktionshierarchien

Die Funktionen werden entsprechend der in *(Pörnbacher & Wünsch 2004)* beschriebenen Struktur nach Funktionen, Funktionseinheiten, Funktionsgruppen und Funktionsuntergruppen gegliedert. Funktionen sind als atomare Einheiten einer Funktionshierarchie zu verstehen, die in der Regel nicht in sinnvoller Weise weiter untergliedert werden können. Ein Beispiel ist etwa das Überwachen eines bestimmten Zustands einer Baugruppe oder das Erfassen einer Position. Funktionseinheiten fassen mehrere Funktionen zu einer übergeordneten Aufgabe zusammen. Dies kann beispielsweise das Ansteuern und Überwachen einer Baugruppe eines Fertigungssystems sein. In der gezeigten Anwendung ist etwa die Kommunikation mit dem übergeordneten Steuerungssystem der Reibschweißanlage als eine Funktionseinheit zu betrachten. Weitere Ebenen sind die Funktionsgruppen und die Funktionsuntergruppen. Während erstere vorwiegend die Aufgaben einzelner Teilsysteme kapseln, stellen Funktionsuntergruppen ergänzende Zwischenebenen dar, die häufig koordinative und überwachende Funktionen für untergeordnete Einheiten umfassen.

7.6 Gestaltende Phase der Systemkonzeption

Im Rahmen der gestaltenden Phase der Systemkonzeption wird der prinzipielle Aufbau der Entwicklung festgelegt. Aufgabe der mechanischen Konstruktion ist es, in einem ersten Schritt auf Basis der Funktionshierarchie bereits vorhandene Teillösungen zu finden, die den gestellten Anforderungen bereits entsprechen oder mit geringem Aufwand adaptierbar sind. Ist dies nicht möglich, so werden grundsätzlich neue Lösungen entwickelt. Im Rahmen dieser kreativen Tätigkeit wird beispielsweise die Strukturierung des Systems in einzelne Subsysteme vorgenommen und die Anordnung der Bewegungsachsen festgelegt. Ergebnis ist ein Konzeptmodell des Fertigungssystems, das üblicherweise in Form eines dreidimensionalen Modells dokumentiert wird und den wesentlichen Aufbau der prinzipiellen Lösung widerspiegelt. Auf eine weiterführende Betrachtung bezüglich der gestaltenden Phase der Systemkonzeption im Hinblick auf die Findung konstruktiver Lösungen wird im Folgenden verzichtet. Diesbezüglich und betreffend ergänzender Themen der Konstruktionsmethodik sei auf entsprechende Fachliteratur *(z. B. Ehrlenspiel 2003; Lindemann 2005; Pahl u. a 2005)* verwiesen. Die durch die mechanische Konstruktion festgelegte Struktur bietet jedoch die Grundlage für die Festlegung der Softwarearchitektur. Diese sollte sich im Wesentlichen an der prinzipiellen Struktur des mechanischen Aufbaus orientieren. Eine derartige Vorgehensweise ist insbesondere für Produkte von Interesse, die entsprechend den Kundenwünschen konfiguriert werden sollen. Ziel der Softwareentwicklerinnen bzw. Softwareentwickler ist es, die Schnittstellen zwischen einzelnen Subsystemen zu generalisieren und möglichst einfach zu gestalten. Dadurch wird eine einfache Konfiguration der Software, aber auch eine schnelle Anpassung an spezielle Anforderungen erleichtert.

Für die Festlegung der Systemarchitektur wird der in Abbildung 7.5 dargestellte Editor verwendet. Dieser erlaubt es, die Struktur des Fertigungssystems in Form eines Komponentendiagramms zu modellieren.

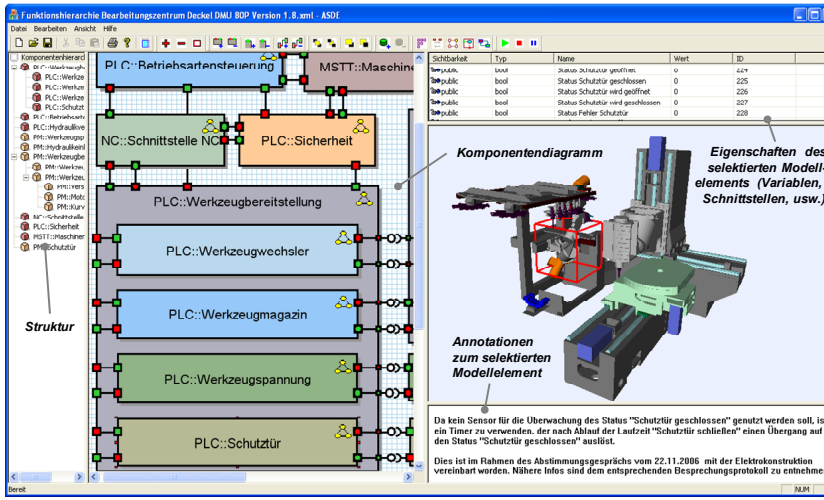


Abbildung 7.5: Aufbau der Systemstruktur in Form eines Komponentendiagramms

Auf der Grundlage der Struktur des mechanischen Teilsystems ergänzt die Softwareentwicklerin bzw. der Softwareentwickler in Kooperation mit den Konstrukteurinnen und Konstrukteuren den prinzipiellen Aufbau der Software. Zusätzlich werden typische, zustandsbeschreibende Eigenschaften der einzelnen Systembausteine in Form von Variablen abgebildet. Ergänzende informale Modellelemente unterstützen das Systemverständnis, indem sie nähere Informationen zu den einzelnen Modellelementen bereitstellen oder auf weiterführende Informationen verweisen. Ebenso hat die dreidimensionale Darstellung des Konzeptmodells die Aufgabe, die Diskussion und Weiterentwicklung der Lösung in einem interdisziplinären Team zu erleichtern.

Anschließend erfolgt die Verknüpfung der Systemarchitektur mit der Funktionshierarchie. Dies geschieht, indem unter Zuhilfenahme eines entsprechenden Editors (siehe Abbildung 7.6) die einzelnen Einträge der Funktionshierarchie den jeweiligen Komponenten im Strukturbaum zugeordnet werden.

Dabei kann es durchaus vorkommen, dass die Abbildung der Funktionen auf die strukturellen Elemente nicht eindeutig ist. Zur Umsetzung einer definierten Funktion kann es beispielsweise notwendig sein, dass mehrere Komponenten koordiniert zusammenwirken. Tritt ein derartiger Fall auf, so ist es in der Regel sinnvoll, die besagte Funktion weiter zu untergliedern, bis eine eindeutige Zuordnung gegeben ist. Umgekehrt im-

plementiert eine Komponente oft mehr als eine Funktion. Beispielsweise hat die Schutzeinrichtung eines Werkzeugwechslers unter anderem die Aufgabe, den Zugang zum Arbeitsraum zu öffnen und nach erfolgtem Werkzeugwechsel wieder zu schließen. Welche Funktionen einer bestimmten Komponente zugeordnet sind, kann überprüft werden, indem die jeweilige Komponente im Strukturbaum des Editors selektiert wird.

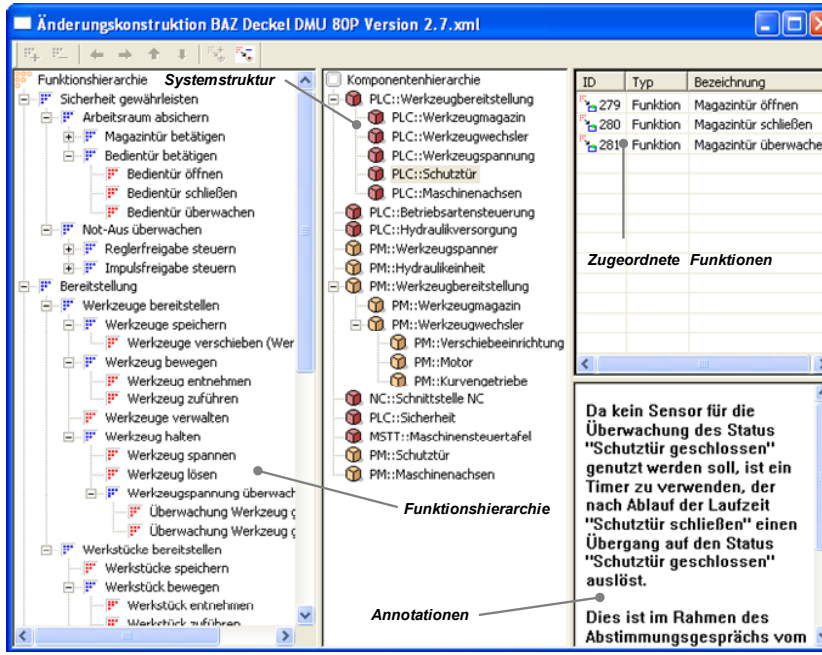


Abbildung 7.6: Zuordnung der Funktionen zu den Komponenten im Systemmodell

Besonders bei Änderungskonstruktionen ist es zweckmäßig, bereits bestehende Modellbestandteile aus Bibliotheken zu verwenden. Soll diese Vorgehensweise angewandt werden, ist es prinzipiell sinnvoll, die diesen Bibliothekselementen zugeordneten Funktionen mit der Funktionshierarchie abzugleichen. Dadurch kann zum einen sichergestellt werden, dass die verwendeten Modellbausteine die notwendige Funktionalität bereitstellen. Zum anderen wird aber auch ersichtlich, ob die generierte Funktionshierarchie noch Unzulänglichkeiten aufweist und ergänzt werden muss.

In einem weiteren Schritt im Rahmen der gestaltenden Phase der Systemkonzeption gilt es, die zu implementierenden Abläufe anhand von Interaktionen (siehe Abbildung 7.7) zu beschreiben. Die detaillierte technische Realisierung der hierzu benötigten Elementarfunktionen ist in dieser Entwicklungsphase noch von untergeordneter

Bedeutung. Das primäre Ziel besteht vielmehr darin, ein abteilungsübergreifendes, gemeinsames Systemverständnis zu generieren und kritische Probleme der Entwicklung anhand der Modelle zu diskutieren. Gerade für diese Aufgabe sind erlebbare Modelle von herausragender Bedeutung. Deshalb wurde in die Entwicklungsumgebung die Möglichkeit implementiert, Sequenzdiagramme zu simulieren und das Systemverhalten anhand eines Kinematikmodells zu visualisieren.

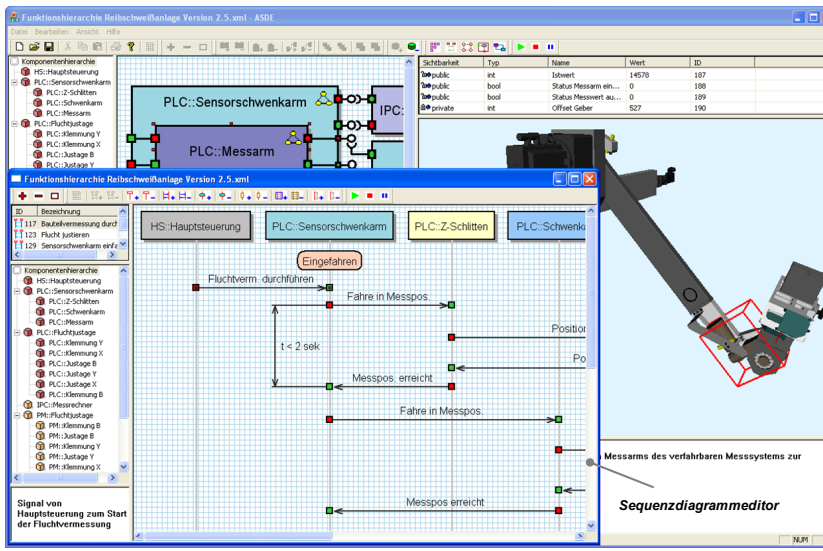


Abbildung 7.7: Sequenzdiagrammeditor zur Modellierung von Interaktionen

Der Vorteil gegenüber gängigen Kinematiksimulationen besteht dabei darin, dass durch die Integration der Sequenzdiagramme bereits eine Formalisierung der Abläufe vorgenommen wird, die für weitere Entwicklungsschritte direkt nutzbar wird. Dies trägt zudem wesentlich dazu bei, die visuell geprägte Denkweise der Konstrukteurinnen und Konstrukteure mit der logisch orientierten Sicht der Softwareentwicklerinnen und Softwareentwickler zu verknüpfen und erleichtert den Übergang von der Funktions- zur Systemstruktur.

Den Aufbau der Interaktionen realisiert ein interdisziplinäres Entwicklungsteam, indem die an der Umsetzung eines Ablaufs beteiligten Komponenten aus dem Komponentendiagramm ausgewählt und in Sequenzdiagrammen in Form von Lebenslinien abgebildet werden. Die Kommunikation zwischen den Interaktionspartnern wird mithilfe von Nachrichten beschrieben. Diese referenzieren indirekt Ausführungsspezifikationen, die als Verbreiterung der Lebenslinie dargestellt werden. Zeitliche Restriktionen, wie beispielsweise die maximal zulässige Ausführzeit für einen Werkzeugwech-

sel, können berücksichtigt und auf einzelne Teilabläufe verteilt werden. Zusätzlich sind Vorbedingungen zur weiteren Ausführung einer Funktion durch Zustandsinvarianten definierbar. Abbildung 7.7 zeigt beispielhaft für die Reibschweißanlage den Ausschnitt einer Interaktion für das Vermessen der zu verschweißenden Bauteile vor Beginn des Schweißprozesses.

Um die Abläufe simulierbar zu gestalten, gilt es, insbesondere das für die Visualisierung notwendige Verhalten des physikalischen Teilsystems in geeigneter Form im Modell zu hinterlegen. Dies wird realisiert, indem die Ausführungsspezifikationen im Sequenzdiagramm auf Funktionen der Komponente verweisen. Diese haben wiederum eine Verknüpfung zu Verhaltenskonstrukten, die einerseits das Systemverhalten entsprechend dem aktuellen Entwicklungsstand abbilden. Andererseits werden zustandsbeschreibende Variablen über Ports und Schnittstellen[4] mit Elementen im Kinematikmodell verknüpft, um das Systemverhalten anschaulich darstellen zu können. Beispiele für derartige Größen sind Schaltzustände, die im Kinematikmodell mittels Farbwechsel sichtbar gemacht werden, oder aktuelle Achspositionen, die durch das Bewegen der entsprechenden Komponenten visualisiert werden.

In dieser frühen Entwicklungsphase ist vor allem die Abbildung des Bewegungsverhaltens, des logischen Verhaltens und des Zeitverhaltens von Interesse. Für die Modellierung wird daher eine Kombination aus elementaren mathematischen, kinematischen und logischen Basisbausteinen sowie Zeitgliedern empfohlen. Logische und mathematische Elemente beschreiben eine Reaktion, die in Abhängigkeit von bestimmten Eingangsgrößen nach einer vorgegebenen Berechnungsvorschrift eintritt. Beispiele für deren Einsatz sind die Abbildung eines mechanischen Getriebe, das die Abtriebsdrehzahl in Abhängigkeit von der Antriebsdrehzahl berechnet, oder ein Sensor, der schaltet, sobald eine systembeschreibende Zustandsgröße einen bestimmten Wert überschritten hat. Verhaltenskonstrukte zur Modellierung des Bewegungsverhaltens im physikalischen Teilsystem berechnen die bewegungsbeschreibenden Zustandsgrößen anhand von Eingangsgrößen und anderen Zustandsgrößen sowie in Abhängigkeit von der Zeit. Zeitglieder hingegen definieren ein noch nicht näher spezifizierbares Systemverhalten in abstrakter Form, indem nach dem Ablauf einer definierten Zeit Zustandsgrößen entsprechend dem gewünschten Verhalten modifiziert werden.

Zur konkreten Modellierung der beschriebenen Grundelemente sind physikalische Verhaltensdiagramme bzw. die hierfür definierten Modellkonstrukte zu verwenden. Als besonders sinnvoll hat sich in dieser frühen Entwicklungsphase eine abstrahierte, funktionale Modellierung in Form einer Kombination aus Kontrollkonstrukten und Gleichungen bzw. Zuweisungen erwiesen. Abbildung 7.8 zeigt exemplarisch die Mo-

[4] Hierzu kommen Ports des Typs *"DSLExternalPort"* und Schnittstellen des Typs *„DSLSignalInterface*" zum Einsatz.

dellierung einer Bewegungsachse, die, entsprechend den Signalzuständen „Vorwärts“ und „Rückwärts“ sowie einer vorgegebenen Bewegungsgeschwindigkeit, die Achsposition berechnet. Informationen zum aktuellen Zustand werden ebenfalls in Form von Signalen zur Verfügung gestellt.

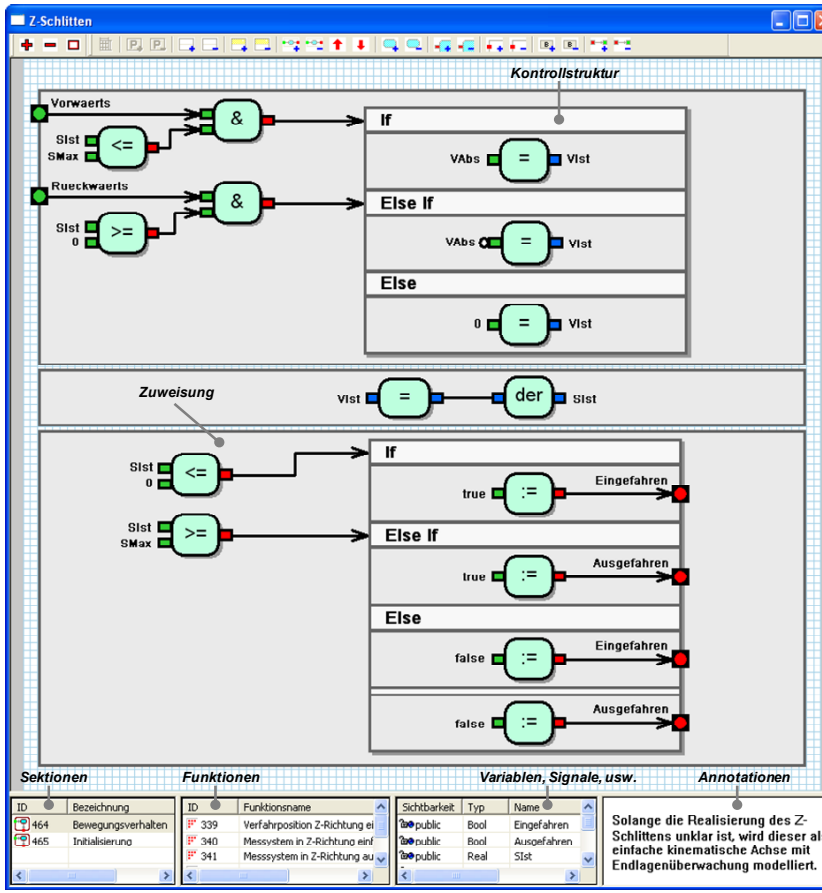


Abbildung 7.8: Physikalisches Verhaltensdiagramm einer Bewegungsachse

Anhand der Anwendungsbeispiele konnte gezeigt werden, dass zur Modellierung von Abläufen nur eine begrenzte Anzahl derartiger Verhaltenskonstrukte notwendig ist. Der Aufwand zur Modellbildung kann daher durch Hinterlegung entsprechender Modellbausteine in Bibliotheken wesentlich reduziert werden.

Neben dem Verhalten des physikalischen Teilsystems kann auch die Abbildung von Funktionen des Steuerungssystems in dieser Entwicklungsphase von Interesse sein. Beispiele hierfür sind die Modellierung von Berechnungsfunktionen oder die Abbildung von Schnittstellenfunktionen zu nicht modellierten Teilsystemen. Hierfür werden Aktivitätsdiagramme als geeignet betrachtet. Abbildung 7.9 zeigt einen Editor zur Modellierung des Verhaltens in Form von Aktivitäten. Dargestellt ist die Berechnung der Sollposition eines Antriebs der Verstelleinrichtung der Reibschweißanlage. Anhand der gemessenen Verkippung der Bauteile zueinander und unter Nutzung geometrischer Beziehungen wird der Korrekturwert für den Antrieb ermittelt.

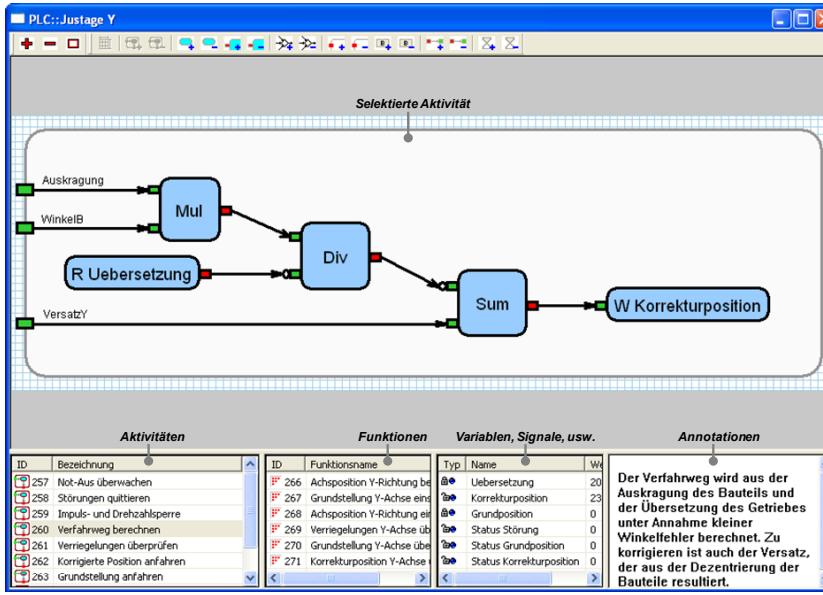


Abbildung 7.9: Editor zur Modellierung von Aktivitäten

Mit der Festlegung oder Erarbeitung neuer prinzipieller Lösungen für das physikalische Teilsystem erfolgt der Übergang von den präskriptiven zu den deskriptiven Modellen. Die dabei anwendbaren Transformationen sind, wie in Abschnitt 6.5.3 erläutert, vor allem für Interaktionen von Interesse. Insbesondere gilt dies für Komponenten, die Funktionseinheiten umsetzen und Elementarfunktionen koordinieren. Das entsprechende Verhalten ist in den Interaktionen in der Regel bereits als geordnete Folge von ein- und ausgehenden Nachrichten sowie Zustandsinvarianten modelliert. Dieser als Trace bezeichnete Verhaltensabschnitt lässt sich besonders einfach in einen Zustandsgraphen oder einen Ausschnitt desselben überführen. Abbildung 7.10 zeigt dies anhand der Ansteuerung der Schutztür des Werkzeugwechslers des in Abschnitt 7.2

beschriebenen Fräsbearbeitungszentrums. Indem Zustandsinvarianten als Zustände, eingehende Nachrichten als Trigger für Transitionen und ausgehende Nachrichten als Aktivitätsaufrufe interpretiert werden, lässt sich der Modellierungsaufwand reduzieren.

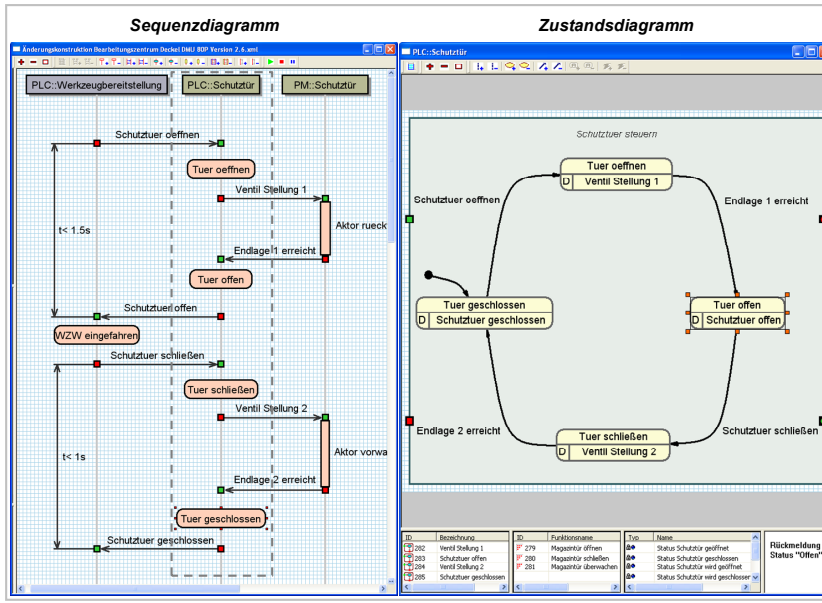


Abbildung 7.10: Transformation am Beispiel einer trennenden Schutzeinrichtung

Für Komponenten, die das Systemverhalten auf der Ebene der Funktionsgruppen oder -untergruppen implementieren, empfiehlt sich häufig eine Überführung der Trace in Aktivitäten. Funktionen auf dieser Ebene werden in der Praxis oft in Form von Schrittketten implementiert. Durch die hohe Affinität der Aktivitäten zur Ablaufsprache der IEC 61131-3 wird daher eine praxisnahe Modellierung sichergestellt.

Die abschließende Aufgabe der gestaltenden Phase der Systemkonzeption ist es, das Systemverhalten der einzelnen Komponenten zu einem konsistenten Gesamtverhalten zusammenzuführen, zu prüfen und zu ergänzen. Diese Tätigkeit entspricht aufgrund der zunehmenden Detaillierung der prinzipiellen Lösungen und der resultierenden Orientierung an der technologischen Umsetzung einem schrittweisen Übergang in die Phase des Systementwurfs.

7.7 Systementwurf

Im Rahmen des Systementwurfs erfolgt die Dimensionierung und geometrische Ausarbeitung der prinzipiellen Lösungen. Das Konzeptmodell kann aufgrund der detaillierteren Entwicklungsinformationen weiter verfeinert und ergänzt werden. Für das Modell des physikalischen Teilsystems muss nun abgeklärt werden, ob die vereinfachte Modellierung der Funktionalität aus der Konzeptphase für den beabsichtigten Modellzweck hinreichend ist oder ob eine Verfeinerung der Struktur und des Verhaltens erforderlich ist. Diese Problemstellung ist nicht allgemeingültig lösbar und von Fall zu Fall zu klären. Detaillierte Modelle erlauben eine genauere Abbildung des Systemverhaltens und sind dann zu bevorzugen, wenn für die Entwicklung immer wieder ähnliche Komponenten und Baugruppen zum Einsatz kommen, die jedoch unterschiedlich kombiniert und verschaltet werden. Eine komponentenorientierte Modellierung mit einer Verhaltensabbildung basierend auf physikalischen Gesetzmäßigkeiten bietet gegenüber einer abstrakt-logischen Betrachtung eine verbesserte Wiederverwendbarkeit einzelner Bausteine und eine implizite Abbildung kausaler Wirkketten. Zudem wird die Transparenz der Modelle erhöht, wodurch ein tiefgehendes Verständnis der Funktionalität gefördert wird. Eine Abbildung von Störungen im Modell zum Test der Steuerungsfunktionen zur Störungsbehandlung wird ebenfalls erleichtert. Nachteilig ist der erhöhte Aufwand, der eine derartige Form der Modellierung trotz des Mehrwertes nicht immer als gerechtfertigt erscheinen lässt. Bei einer komponentenorientierten Modellierung empfiehlt es sich, die Modellbausteine zu parametrisieren, um die Anzahl der notwenigen Elemente zu reduzieren. So sind beispielsweise unterschiedliche geometrische Ausprägungen einzelner Komponenten durch eine Adaption der Parameter nachbildbar.

Wesentliche Arbeiten im Steuerungsmodell betreffen die Verfeinerung der modellierten Funktionen im Hinblick auf die explizite Umsetzung der Prinziplösungen. So sind beispielsweise Aufgaben, die im Konzeptmodell aufgrund einer noch nicht im Detail klaren technischen Lösung vereinfacht abgebildet wurden, zu überarbeiten und an die konkrete Ausprägung anzupassen. Darüber hinaus müssen aber auch Ergänzungen vorgenommen werden. Beispielsweise sind softwarerelevante Aufgaben, die ein deterministisches Verhalten sicherstellen und ein potentielles Fehlverhalten überwachen sollen, und die Koordination kontinuierlich auszuführender Aktivitäten in den Modellen zu komplettieren. Abbildung 7.11 zeigt dies beispielhaft an einem Modellbaustein zur Klemmung der Verstellvorrichtung der Reibschweißanlage. Der koordinierende Zustandsgraph, der die Ansteuerung der entsprechenden Hydraulikventile übernimmt, wird um einen zusätzlichen, orthogonal unabhängigen Graphen erweitert, der die Freigabe der Klemmvorrichtung überwacht. Ist eine in der Maschine vorhandene Schutzeinrichtung nicht geschlossen, so wird eine den Zustand abbildende Variable im Baustein gesetzt. Dies hat wiederum zur Folge, dass die Klemmvorrichtung deaktiviert

wird. Weiterhin werden im aufgeführten Beispiel Aktivitäten zur Umsetzung von Laufzeitüberwachungen sowie zum Einstellen der Betriebsart ergänzt.

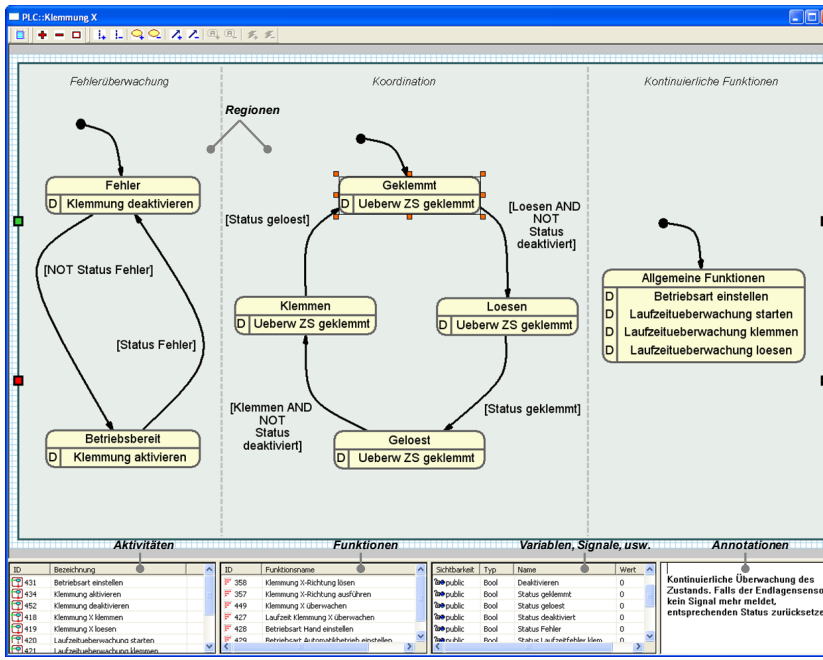


Abbildung 7.11: Ergänzende Modellierung am Beispiel einer Klemmvorrichtung

Auch in der Phase des Systementwurfs ist der Einsatz von Simulationswerkzeugen zur Absicherung der Modelle von wesentlicher Bedeutung. Durch die simulationsgestützte Untersuchung des Systemverhaltens können Fehler in der technologischen Umsetzung der benötigten Funktionalitäten erkannt und beseitigt werden. Für diese Entwicklungsphase sind Methoden der Modellinterpretation (siehe Abschnitt 5.6.2) vorteilhaft einsetzbar. Diese haben die Aufgabe, die eingesetzten Sprachelemente in Zustandsgraphen und Aktivitäten im Steuerungsmodell bzw. Verhaltensdiagramme im physikalischen Teilmodell zur Laufzeit auszuwerten und entsprechende Änderungen im Modellverhalten zu induzieren. Da die Umsetzung einer umfassenden Entwicklungsumgebung nicht Schwerpunkt der Arbeit war, wurde nur ein Teil der notwendigen Funktionalitäten in den Prototyp der Entwicklungsumgebung implementiert.

7.8 Softwareerstellung (Implementierung)

Mit Abschluss der Entwurfsphase ist das Verhalten und die Struktur des Fertigungssystems möglichst vollständig und hinreichend genau abgebildet und formalisiert. In

der nachfolgenden Implementierungsphase gilt es, das Steuerungsmodell in Softwarebausteine zu überführen und zu ergänzen. Um dies zu ermöglichen, ist in einem ersten Schritt die technologische Umsetzung der Automatisierungslösung in Form einer Hardwarebeschreibung festzulegen. Hierzu kommen in der Praxis Softwarewerkzeuge zum Einsatz, die üblicherweise durch den Steuerungshersteller zur Verfügung gestellt werden und an die jeweilige Hardware angepasst sind. Aufgrund der Vielfalt der am Markt verfügbaren Lösungen wird, wie in Abschnitt 6.5.4 erläutert, auf eine Modellierung der Hardwarearchitektur in Form eines eigenen Diagramms[5] verzichtet. Die in kommerziell verfügbaren Systemen üblicherweise klare Trennung der Adressierung von Steuersignalen und deren physikalischen Entsprechung unterstützt diese Vorgehensweise. Datentypen und Adressbereiche der Variablen im Steuerungsmodell können daher auf Basis der Hardwareprojektierung im Modell ergänzt werden. Alternativ ist dieser Arbeitsschritt bereits im Rahmen der Entwurfsphase möglich, sofern Richtlinien[6] existieren, die eine Festlegung der Adressierung eindeutig regeln. Ähnliches gilt für die Definition der Namenskonventionen und der Programmiersprachen für einzelne Softwarebausteine, die diesen als Teil der Markierung vor der Überführung in Steuerungscode zugewiesen werden.

Sind die notwendigen Informationen im Steuerungsmodell ergänzt, so kann dieses unter Nutzung entsprechender Transformationsregeln in Softwarebausteine überführt werden, wobei die in Abschnitt 6.5.4 beschriebenen Rahmenbedingungen zu berücksichtigenden sind. Besonders empfehlenswert ist auf dieser Ebene die Integration von Standardfunktionen zur Umsetzung bestimmter Funktionen, die bereits vom Hersteller der Steuerungshardware oder alternativ von Zulieferern bereitgestellt werden. Indem durch eine spezielle Markierung auf einen entsprechenden Baustein verwiesen wird, kann dieser automatisiert aus einer Bibliothek entnommen und in das Steuerungsprojekt eingebunden werden. Entsprechende Modellbausteine werden dann nicht transformiert, jedoch ist durch die Prüfung der Schnittstellen und der übergebenen Parameter des Funktionsaufrufs die Konformität mit den generierten Teilen des Steuerungscodes sicherzustellen. Weiterhin empfiehlt es sich, informale Modellbestandteile als Teil der Dokumentation in den Baustein zu integrieren. Dies trägt wesentlich zur besseren Lesbarkeit des Generats bei.

Der Prototyp der Entwicklungsumgebung beschränkt sich auf die Transformation von Zustandsgraphen und Aktivitäten in Anweisungsliste (AWL)[7]. Abbildung 7.12 zeigt das Ergebnis der Transformation beispielhaft anhand eines Bausteins zur Ansteuerung

[5] Die UML bietet mit dem Verteilungs-Diagramm prinzipiell auch hierzu eine Lösung.

[6] Hersteller von Fertigungssystemen weisen einzelnen Teilsystemen oft bestimmte Adressbereiche zu, um die Wartbarkeit und Anpassbarkeit der Software zu verbessern. Dies gilt insbesondere für Hersteller, die Gesamtsysteme entsprechend den Kundenwünschen aus einzelnen Modulen konfigurieren.

[7] Als Grundlage zur Umsetzung der Transformation diente die in *(Lutz 1999, S. 152-158)* aufgezeigte Methode.

einer hydraulischen Klemmvorrichtung. Als Beispiele für weiterführende Arbeiten, die sich mit der Transformation von UML-Modellen in Steuerungscode beschäftigen, seien *(Braatz 2003, Bayrak u. a. 2008)* und *(Witsch u. a. 2008)* genannt.

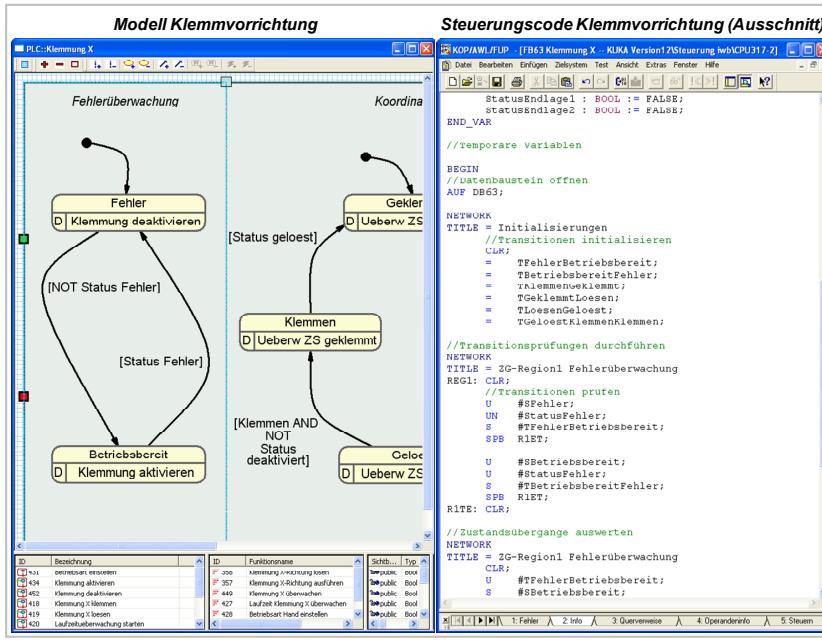


Abbildung 7.12: Generierung von Steuerungscode aus einem Zustandsgraphen

Trotz der Vorteile einer Generierung von Softwarebausteinen sind häufig ergänzende Programmieraufgaben nicht vermeidbar. Dies gilt insbesondere dann, wenn der Steuerungscode in Numerische Steuerungen integriert werden muss. Diese bieten meist ein Grundgerüst an, das standardisierbare Aufgaben bereits implementiert hat und somit das Entwicklungspersonal entlastet. Die verfügbaren Schnittstellen sind jedoch komplex und in ihrer Struktur und ihren Eigenschaften (Adressbereiche, Datentypen, usw.) nicht adaptierbar. Da dies bei der Codegenerierung berücksichtigt werden muss, empfiehlt es sich, Schnittstellen soweit notwendig als Komponenten im Steuerungsmodell abzubilden. Der entsprechende, einmalige Aufwand wird durch die Vermeidung nachfolgender Anpassungen mehr als egalisiert.

Die nachfolgende Phase der simulationsgestützten Integration und Inbetriebnahme der Steuerungssoftware erfordert auch eine Adaption des Modells des physikalischen Teilsystems. Wesentliche Aufgabe ist die Auswertung der graphischen Modellbeschreibungen und deren Überführung in entsprechende Anweisungen in Modelica. Darüber

hinaus ist es notwendig, die Schnittstelle zum Steuerungsmodell konform abzubilden. Als problematisch erweist sich trotz der prinzipiellen Vorteile von Modelica die fehlende Funktionalität entsprechender Simulationssysteme. Kommerziell verfügbare Produkte sind nicht an die Erfordernisse einer Co-Simulation adaptiert und bieten nur eingeschränkte Schnittstellen. Zum Nachweis der Funktionsfähigkeit wurde im Rahmen der Arbeit eine Koppelung des kommerziell verfügbaren Modelica-Simulators „Dymola“ an eine Numerische Steuerung (Siemens 840D) implementiert *(Guserle & Zäh 2005)*. Diese weist jedoch insbesondere in Bezug auf die ergonomische Gestaltung noch Schwachstellen auf, die einen effizienten Praxiseinsatz verhindern. Für eine simulationsgestützte Integration und Inbetriebnahme der Steuerungssoftware wurde daher eine effizientere Lösung entwickelt. Deren Aufbau sowie deren praxisnaher Einsatz werden im folgenden Abschnitt beschrieben.

7.9 Integration und Inbetriebnahme

Der abschließende Schritt der Integration und Inbetriebnahme der entwickelten Steuerungssoftware umfasst die Zusammenführung einzelner Teilsysteme, die Einbindung von Zulieferkomponenten, die Einstellung von Parametern (Laufzeitüberwachungen, usw.) sowie die Verifikation des Zusammenspiels der Steuerungssoftware mit den hydraulischen, mechanischen, elektromechanischen und pneumatischen Komponenten des Fertigungssystems. Um diese Aufgaben soweit möglich bereits vor oder während des Aufbaus der Maschinen bzw. Anlagen zu erledigen und die Vorteile einer simulationsgestützten Vorgehensweise vollständig nutzen zu können, muss eine entsprechende Umgebung modular aufgebaut und flexibel an den jeweiligen Anwendungsfall anpassbar sein. Zudem bedarf es der Unterstützung der in der industriellen Praxis üblichen und bewährten Bottom-up-Strategie zum Softwaretest. Abbildung 7.13 zeigt die Architektur eines prototypisch entwickelten Simulationssystems. Dieses besteht aus einem Steuerungssystem, das die Ausführung der entwickelten Programme zur Laufzeit übernimmt. An dieses wird ein kommerziell verfügbarer Simulator angekoppelt, der das Modell des physikalischen Teilsystems umsetzt. Dieser ist wiederum an einen Kinematiksimulator angebunden und erlaubt damit die Visualisierung des Maschinenverhaltens. Als besonders vorteilhaft erweist sich die Tatsache, dass das System sowohl als Hardware- als auch als Software-in-the-Loop-Simulation[8] betrieben werden kann[9]. Ferner können aufgrund der Möglichkeit zur Anbindung des Steuerungssystems an den Simulator des physikalischen Teilmodells über einen Feldbus auch reale Bau-

[8] Ein Vergleich der beiden Simulationsmöglichkeiten bzgl. ihrer Vor- und Nachteile bei der Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme ist *(Zäh & Pörnbacher 2006)* zu entnehmen.

[9] Die Kommunikation zwischen dem Steuerungsemulator und dem Simulator für das physikalische Teilsystem erfolgt dann über einen gemeinsamen Speicherbereich. Bei der Hardware-in-the-Loop-Simulation wird ein standardisiertes Feldbussystem eingesetzt.

gruppen, deren Nachbildung im Simulationsmodell zu aufwendig oder technisch nicht möglich ist, integriert werden.

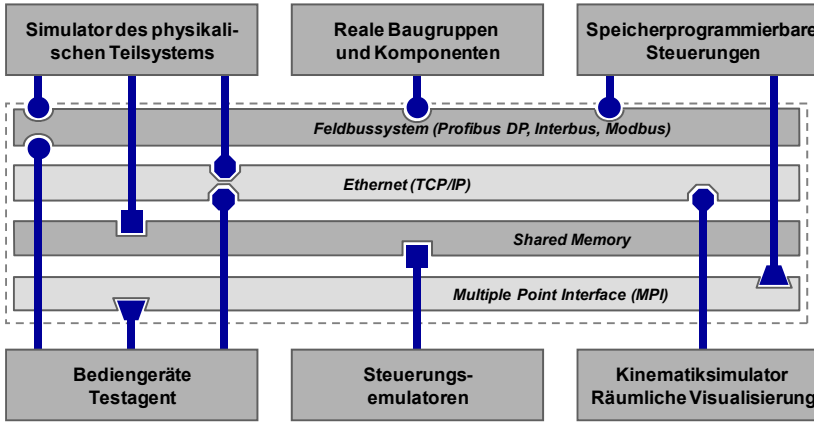


Abbildung 7.13: Architektur des entwickelten Simulationssystems

Während sich die Koppelung der einzelnen Teilsysteme aufgrund der in den Modellen bereits inhärenten Informationen sehr effizient gestalten lässt, weist die implementierte Lösung in Bezug auf die Integration des Modells des physikalischen Teilsystems noch Schwachstellen auf. Da der verwendete Simulator bezüglich der Modelle lediglich proprietäre Schnittstellen bereitstellt, konnte die Forderung der Datendurchgängigkeit nicht vollständig erfüllt werden.

Im Rahmen der Umsetzung der in Abschnitt 7.2 beschriebenen Anwendungsbeispiele wurden entsprechend den Anforderungen unterschiedliche Konfigurationen des Simulationssystems verwendet. Abbildung 7.14 zeigt den für die Umkonstruktion des Werkzeugwechselsystems des Fräsbearbeitungszentrums eingesetzten Aufbau.

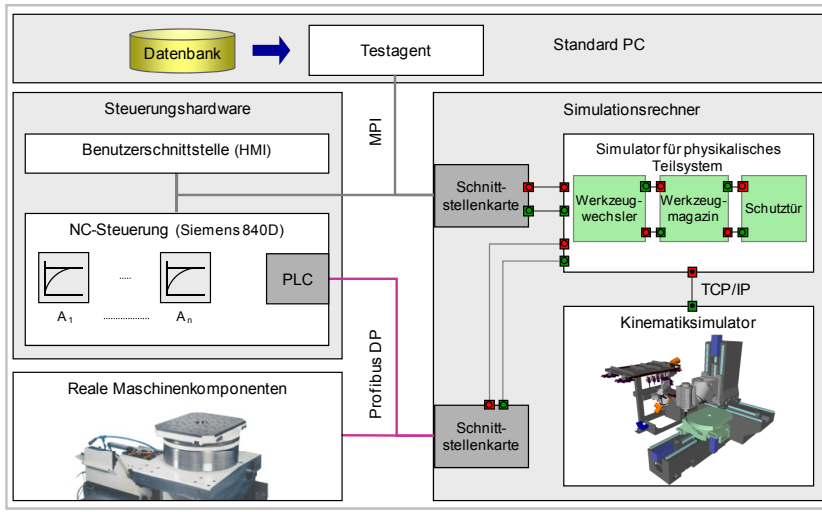


Abbildung 7.14: Aufbau einer HIL-Simulation mit einer Numerischen Steuerung

Eine Numerische Steuerung (Siemens 840D) wird über ein standardisiertes Feldbussystem (Profibus-DP) und eine Interfacekarte, die das entsprechende Feldbusprotokoll implementiert, an einen Simulationsrechner angeschlossen. Der Simulator des physikalischen Teilsystems kommuniziert zum einen über die Schnittstellenkarte mit der Steuerung und leitet zum anderen über eine eigens entwickelte Datenschnittstelle die Visualisierungsinformationen an einen Kinematiksimulator weiter. Um auch die Positionswerte der numerischen Achsen des Bearbeitungszentrums für die Simulation bereitzustellen, wurde die Steuerung mit kurzgeschlossenen Regelkreisen betrieben. Dabei wird das Verhalten der elektromechanischen Antriebe in der Steuerung durch ein Zeitglied erster Ordnung abgebildet. Durch eine speziell entwickelte Funktion werden die berechneten Istpositionen über die Speicherprogrammierbare Steuerung weitergeleitet und damit für die Simulation und Visualisierung nutzbar gemacht. Die implementierten Programmbausteine können somit bezüglich der korrekten Funktionsweise geprüft werden. Zur Minimierung des hierzu erforderlichen Aufwandes wurde ergänzend der Prototyp eines automatisierten Testsystems konzipiert und umgesetzt. Ein Testagent kann auf der Grundlage von Skriptdateien sowohl potentielle Maschinenbedienerinnen bzw. -bediener simulieren als auch aktiv Änderungen in der Simulation vornehmen. Damit besteht die Möglichkeit zum gezielten Auslösen von Störsituationen im Modell des physikalischen Teilsystems und somit auch zum Test der Steuerungsfunktionen zur Störungsbehandlung. Aufgrund der Gefahr von Maschinenschäden sind derartige Versuche an realen Fertigungssystemen meist nicht zulässig bzw. zu riskant. Das Verhalten des Systems wird durch die Beobachtung von Signalen im Si-

mulationsmodell und der Ein- und Ausgänge oder internen Zustände der Steuerung mit einem gewünschten Sollverhalten verglichen und protokolliert. Damit ist es möglich, verschiedene Testfälle automatisiert abzuarbeiten und deren Ergebnis zu speichern. Bezüglich weiterführender Informationen zur prototypischen Implementierung und Funktionsweise des beschriebenen Testsystems sowie zu dessen Handhabung wird auf *(Pörnbacher & Wünsch 2004, Zäh & Pörnbacher 2005)* bzw. *(Wünsch 2008)* verwiesen.

Die Abbildungen 7.15 bis 7.17 zeigen weitere Konfigurationen des Simulationssystems, die für die Entwicklung des Mess- und Justagesystems der Reibschweißanlage eingesetzt wurden. Der erste Aufbau bestand aus der Koppelung eines Steuerungsemulators mit einer Simulation der Messeinrichtung (Abbildung 7.15). Da die Software zur Auswertung der von den optischen Sensoren des Systems erfassten Signale von einem weiteren Projektpartner entwickelt wurde, beschränkte sich die Modellierung der Einrichtung zur Signalerfassung und -verarbeitung auf die Abbildung der Schnittstelle und die vereinfachte Modellierung der Messfunktionen[10].

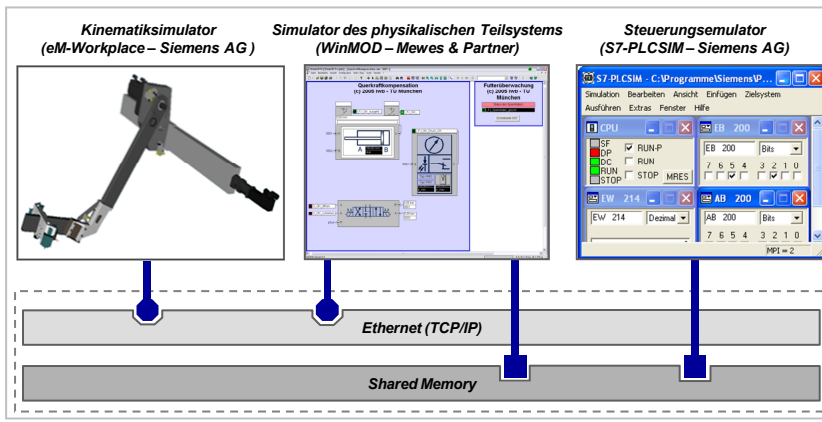


Abbildung 7.15: Steuerungsemulation, gekoppelt mit einem Simulationsmodell

Die Software konnte auf diese Weise bereits in Betrieb genommen werden, bevor die mechanischen und elektromechanischen Komponenten des System gefertigt und montiert wurden. Nach der erfolgreichen Prüfung der Softwarebausteine zur Ansteuerung des mechanischen Aufbaus wurden in einem zweiten Schritt die Steuerungsemulation und das simulierte Messsystem sukzessive durch die realen Komponenten ersetzt (siehe Abbildung 7.16). Die nachfolgenden Arbeiten am System beschränkten sich auf die

[10] Dies erfolgte, indem bei entsprechenden Funktionsaufrufen vorgegebene Werte als Ergebnis zurückgegeben wurden.

genaue Einstellung der Sollpositionen der Verfahrachsen, die Einstellung der Verfahrgeschwindigkeiten und der Parameter der Laufzeitüberwachungen. Die Stelleinrichtung sowie die Schnittstelle zur übergeordneten Hauptsteuerung wurden weiterhin unter Nutzung des Simulators abgebildet.

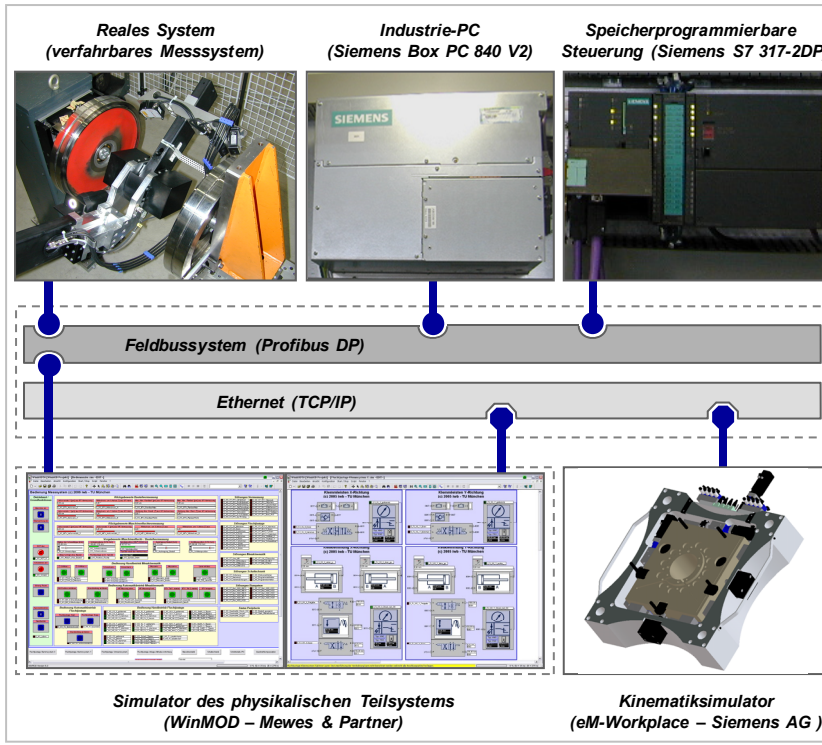


Abbildung 7.16: Hardware-in-the-Loop-Simulation des Mess- und Justagesystems

Der vorletzte Integrationsschritt bestand darin, das Simulationsmodell der Stelleinrichtung durch die realen Komponenten zu ersetzen (siehe Abbildung 7.17). Auch hierbei zeigte sich, dass mithilfe der Simulation die Software nahezu vollständig getestet werden kann. Die implementierten Programmbausteine funktionierten unmittelbar korrekt, lediglich eine Anpassung einzelner Softwareparameter war für einen einwandfreien Betrieb noch erforderlich.

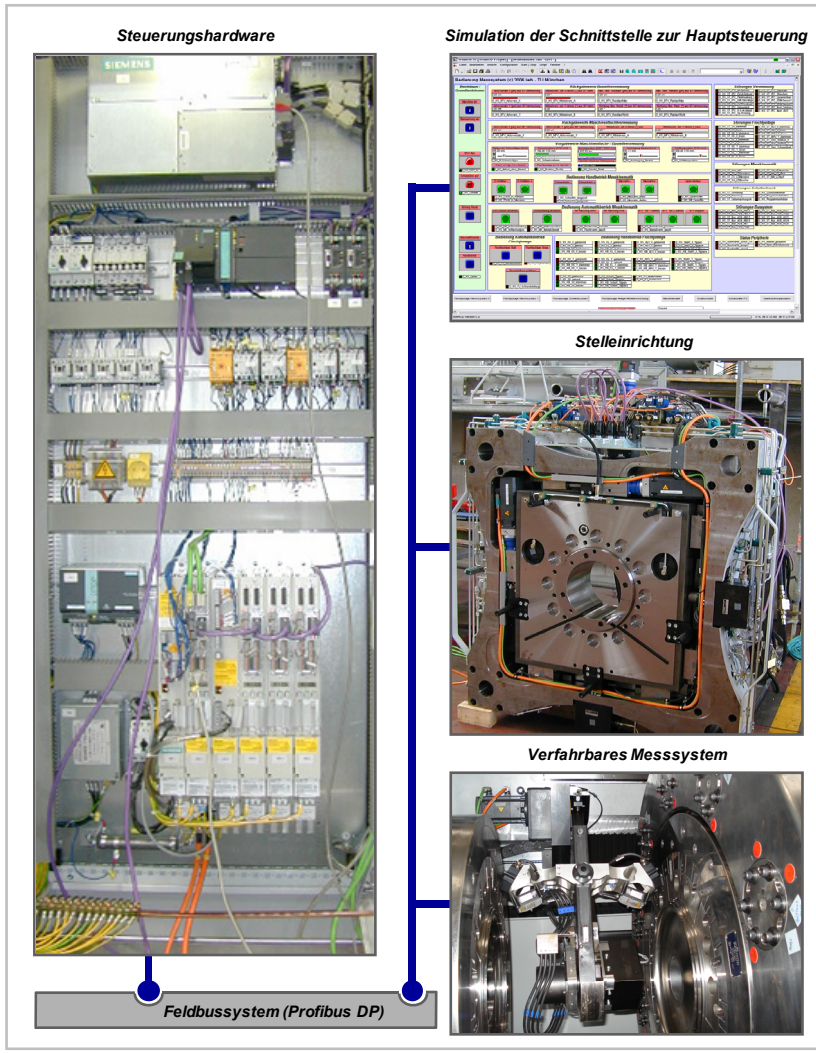


Abbildung 7.17: Testaufbau mit Simulation der Schnittstelle zur Hauptsteuerung

Da die Arbeiten parallel zur Entwicklung der Reibschweißanlage durchgeführt wurden, bestand der abschließende Integrationsschritt im Wesentlichen darin, die simulierte Schnittstelle zur Hauptsteuerung durch die entsprechende Datenverbindung des realen Systems (Verknüpfung über Buskoppler) zu ersetzen und geometrierelevante Parameter nachzustellen.

7.10 Zusammenfassung

Durch die Anwendung der entwickelten Vorgehensweise und der Modellbildungsmethoden an Beispielen aus der industriellen Praxis konnten die Vorteile der modellgetriebenen Softwareentwicklung aufgezeigt werden. Die dabei gesammelten Erfahrungen resultieren jedoch auch in einigen prinzipiell zu beachtenden Rahmenbedingungen, deren Beachtung im Sinne eines erfolgreichen Einsatzes notwendig erscheint:

- Die Phase der Systemkonzeption ist als die wichtigste Phase des gesamten Entwicklungsprozesses zu betrachten. Eine eindeutige Klärung der umzusetzenden Funktionalität im interdisziplinären Team ist für eine effiziente und zielgerichtete Entwicklung von herausragender Bedeutung. Die vorgestellten Techniken zum Aufbau präskriptiver und deskriptiver Modelle lassen sich hierzu gut verwenden. Durch die Kombination formaler, ausführbarer Beschreibungsmittel mit einer dreidimensionalen Visualisierung wird die Entwicklung eines gemeinsamen System- und Funktionsverständnisses wesentlich erleichtert. Fehlentwicklungen, die auf Missverständnissen beruhen, lassen sich dadurch besser vermeiden. Darüber hinaus resultiert eine vollständige Klärung der Funktionalität am Beginn der Entwicklung auch in einer besser strukturierten Software. Folgekosten für Wartung und Pflege der Software können dadurch reduziert und die Anpassung bzw. Erweiterung der Funktionalität erheblich erleichtert werden.
- Die Anwendung der Methoden zur modellgetriebenen Softwareentwicklung erfordert neben einer strukturierten Vorgehensweise auch ein Bewusstsein für die Entwicklungsziele und die vorhandenen Möglichkeiten im Unternehmen. Bereits vor der Modellierung sollte der Modellzweck eindeutig geklärt und kommuniziert werden. Die Potentiale der zur Verfügung stehenden Entwicklungswerkzeuge, aber auch deren Schwächen sind dabei genauso zu beachten, wie die Fähigkeiten der Anwenderinnen und Anwender.
- Inwieweit eine durchgängige Umsetzung der beschriebenen Methoden möglich ist, ist wesentlich vom Anwendungsbereich, den zur Verfügung stehenden Werkzeugen und dem Schwerpunkt der Softwareentwicklung abhängig. Auch der Einsatz ausgewählter Techniken kann zur Verbesserung der Prozesse im Rahmen der Softwareentwicklung beitragen. Darüber hinaus können durch die praxisnahe Umsetzung modellgetriebener Methoden auch neue Anwendungsbereiche erschlossen werden. Als Beispiel sei die Verwendung der generierten Simulationsmodelle für Schulungs- und Präsentationszwecke genannt.

8 Technische und wirtschaftliche Betrachtung

Der von den Herstellern von Fertigungssystemen für die Einführung der Geschäftsprozesse zur modellgetriebenen Entwicklung der Steuerungssoftware zusätzlich zu erbringende Aufwand lässt sich nur indirekt einem entsprechenden Mehrwert gegenüberstellen. Zum einen hängt der Nutzen wesentlich von der geforderten Integrationstiefe, der Art der Entwicklungsprojekte, dem Produktspektrum und der Ausgangssituation im Unternehmen ab. Zum anderen entziehen sich die Vorteile der modellgetriebenen Softwareentwicklung, wie sie im Rahmen dieser Arbeit vorgestellt werden, großteils einer Quantifizierung mit Kennzahlen, die mit den anfallenden Kosten in Beziehung gebracht werden könnten. Die Bewertung der vorgestellten Lösungsansätze erfolgt daher auf der Basis einer qualitativen Betrachtung der zu erwartenden Effekte und der notwendigen Aufwendungen. Soweit möglich fließen auch quantifizierbare Größen in die Ausführungen ein.

- **Verbesserte Integration der Softwareentwicklung in den gesamten Entwicklungsprozess von Fertigungssystemen**

  Nach *(Vogel-Heuser & Fischer 2002)* fordern die Anwenderinnen und Anwender bei der Entwicklung automatisierter Systeme eine bessere Strukturierung der Entwicklungsprozesse. Insbesondere wird die Notwendigkeit einer verstärkten Einbindung der Softwareentwicklung in die frühen Entwicklungsphasen der Anforderungserfassung und der Systemkonzeption als notwendig erachtet, da sich Fehler aus diesen Entwicklungsphasen nur mit hohem Zusatzaufwand beheben lassen. Anhand eines Beispiels aus dem Anlagenbau wird das Einsparpotential durch eine kostengünstigere technische Lösung auf ca. 10 bis 15 Prozent geschätzt. Das Potential zur Verkürzung der Inbetriebnahmezeit aufgrund einer Vermeidung von Fehlentwicklungen wird mit 10 bis 20 Prozent angegeben. Vor dem Hintergrund, dass auch bei der Entwicklung automatisierter Fertigungssysteme weit mehr als 50 Prozent aller Softwarefehler in den frühen Entwicklungsphasen zustande kommen *(Weck & Brecher 2006, S. 132)*, sind ähnliche Einsparpotentiale zu erwarten.

- **Reduktion der Entwicklungszeit durch Parallelisierung und Synchronisation der Geschäftsprozesse**

  Die bessere Integration des Softwareengineerings in die weiteren Geschäftsprozesse der Produktentwicklung führt zu einer entscheidenden Verkürzung der Time to Market. Durch die frühzeitige Einbindung der Softwareentwicklung im Rahmen der Anforderungsanalyse und der Konzeption wird auch die Effektivität der Entwicklung erhöht, da die Softwarearchitektur und die Funktionalität wesentlich besser auf das Produkt abgestimmt werden können. Aufwändige Änderungen im Rahmen der Inbetriebnahme sind somit zu einem großen Teil vermeidbar. Vogel-Heuser und Fischer *(Vogel-Heuser & Fischer 2002)* bewerten das erreichbare Einsparpotential mit ca. 10 bis 20 Prozent der Durchlaufzeit. Ähnliche Potentiale werden von Ehrlenspiel (*Ehrlenspiel 2003, S. 206)* angegeben. Entsprechenden Unter-

suchungen zufolge sind die Möglichkeiten zur Reduktion der Durchlaufzeit auf ca. 25 Prozent zu schätzen. Ferner wird angenommen, dass sich aufgrund der Umsetzung des Concurrent Engineering und der daraus resultierenden Synergieeffekte auch die Herstellkosten um ca. 25 Prozent senken lassen.

- **Gesteigerte Softwarequalität durch eine durchgängige und konsistente Modellbildung und simulationsgestützte Absicherung und Optimierung der Steuerungssoftware**

  Die konsequente und durchgängige Abbildung der erforderlichen Steuerungsfunktionalität in Modellen steigert die Transparenz der Entwicklung und fördert das fachbereichsübergreifende Systemverständnis. In der Folge zeichnen sich die entwickelten Lösungen durch eine deutlich höhere Qualität aus. Dies führt nicht nur zu den bereits dargestellten positiven Effekten für die Hersteller von automatisierten Fertigungssystemen, sondern resultiert nach Ehrlenspiel (*Ehrlenspiel 2003, S. 206)* auch in einer Reduktion der Betriebskosten um ca. 20 Prozent.

  Aufgrund der komponentenorientierten Gestaltung der Softwaremodelle und deren Orientierung an der Struktur des physikalischen Teilsystems wird weiterhin die Wiederverwendbarkeit bestehender Ergebnisse signifikant verbessert. Derartige, mehrfach verwendbare Resultate sind beispielsweise allgemeine Projekterfahrungen, Funktionshierarchien, Softwarearchitekturen, Komponenten, Verhaltensmodelle oder deren Bestandteile, aber auch fertige Softwarebausteine.

  Ein wesentlicher Nutzen ist auch durch die entwicklungsbegleitende, simulationsgestützte Verifikation und Validierung der Modelle zu erwarten. Entsprechende Methoden tragen erheblich dazu bei, Fehlentwicklungen frühzeitig zu erkennen und die daraus resultierenden Folgen zu vermeiden. Dies gilt insbesondere für Softwarefunktionen zur Fehlererkennung und Diagnoseroutinen (beispielsweise Paarfehler- oder Laufzeitüberwachungen), die bei automatisierten Fertigungssystemen bereits heute mehr als 50 Prozent des Steuerungscodes umfassen *(Weck & Brecher 2006, S. 137)*. Diese können oft im Rahmen der Inbetriebnahme nicht getestet werden, da die resultierenden Folgekosten aufgrund eines unvorhersehbaren Verhaltens als nicht tragbar einzuschätzen sind. Eine simulationsgestützte Entwicklung der Steuerungssoftware hat aufgrund der erreichbaren Qualität nicht nur geringere Wartungskosten und eine bessere Außenwirkung beim Kunden zur Folge, sondern kann auch erheblich zur Reduktion der Inbetriebnahmezeit beitragen. Mewes *(Mewes 2004)* schätzt die Potentiale zur Verkürzung der Inbetriebnahmezeit der Software auf ca. 50 Prozent. Vor dem Hintergrund, dass die Inbetriebnahme der Software automatisierter Anlagen etwa 90 Prozent der Inbetriebnahmezeit beansprucht *(Schelberg 1994)*, ergeben sich durch den Einsatz von Simulationswerkzeugen beachtliche Vorteile für die Entwicklung.

- **Langfristige Sicherung von Wissen durch Modellierung und Formalisierung**

  Die Formalisierung von Expertenwissen anhand von Modellen trägt zum einen zu einem effizienteren Engineering bei. Redundante Entwicklungen werden durch die

Dokumentation der Ergebnisse verhindert und sind aufgrund der formalen Darstellung auch anderen Mitgliedern des Entwicklungsteams zugänglich. Zum anderen wird die Einarbeitung für neue Mitarbeiterinnen bzw. Mitarbeiter erheblich erleichtert. Dies ist insbesondere deshalb der Fall, da in der Praxis derzeit eine unmittelbare Implementierung der Funktionalität erfolgt und auf Dokumentationsaufgaben meist verzichtet wird. Auch über die Softwareentwicklung hinaus hat der Aufbau formaler Modelle positive Auswirkungen auf die Kompetenz des Entwicklungspersonals und die erarbeiteten Ergebnisse. Die fachbereichsübergreifende Diskussion der Modelle im interdisziplinären Team trägt wesentlich dazu bei, die unterschiedliche Denkweise der beteiligten Entwicklungsdomänen zu integrieren und hilft gerade aufgrund der entstehenden Synergieeffekte bei der Erarbeitung innovativer, qualitativ hochwertiger und kostengünstiger Lösungen.

- **Weiterverwendung der Modelle für ergänzende Geschäftsprozesse**

  Durch die Weiterverwendung der erstellten Modelle für alternative Zwecke lässt sich ein zusätzlicher Mehrwert generieren. Dieser ist sowohl der Phase der Produktentwicklung, aber auch der nachfolgenden Nutzung der Systeme zuzuordnen. Im Rahmen der Kommunikation mit Auftraggebern, Kunden und Zulieferern können die Modelle als Diskussionsbasis herangezogen werden. Die umzusetzende oder bereitgestellte Funktionalität kann anhand präskriptiver Modelle visualisiert und diskutiert werden. Somit können die Arbeiten stringenter und zielführender gestaltet werden. Als weiteres Beispiel sei die Nutzung der Modelle durch andere Entwicklungsabteilungen genannt. So ist etwa die den Modellen inhärente Beschreibung der Ein- und Ausgangsbelegung der Steuerungen direkt für die Erstellung der Elektrodokumentation nutzbar, wobei eine gemeinsame Datenbasis dabei hilft, Redundanzen und daraus resultierende Fehler zu vermeiden. Eine zusätzliche Möglichkeit zur Weiterverwertung, insbesondere der Modelle des physikalischen Teilsystems, in nachstehenden Phasen des Produktlebenszyklus, ist der Einsatz für Trainings- und Schulungszwecke. Kommuniziert anstelle der realen Maschine ein Modell mit einer Steuerung, so kann zum einen eine unwirtschaftliche Belegung der Fertigungssysteme vermieden werden. Zum anderen ist die Gefahr von Maschinenschäden durch Fehlbedienungen unerfahrener Anwenderinnen und Anwender bei simulierten Systemen nicht gegeben.

- **Einführung im Unternehmen und Erstellung von Modellbibliotheken**

  Die Integration domänenübergreifender, modellgetriebener Entwicklungsmethoden in Unternehmen des Maschinen- und Anlagenbaus ist mit einem gewissen Aufwand verbunden. Die Umstellung erfordert unter Umständen eine Weiterqualifikation des Entwicklungspersonals, aber auch die Anpassung organisatorischer Strukturen. Litto und Lewek *(Litto & Lewek 2005)* stellen zur Einführung zwei prinzipielle Strategien vor. Während die Globalstrategie eine durchgängige Umsetzung für das gesamte Produktspektrum propagiert, zielt die Keimzellenstrategie auf eine sukzessive Einführung entsprechender Methoden und Vorgehensweisen. Indem zunächst nur die Produkte oder Baugruppen betrachtet werden, bei denen der

Mehrwert am höchsten ist, wird zum einen der notwendige Aufwand reduziert und zum anderen arbeitsbegleitend Wissen aufgebaut. Dem zufolge wird auch die zweitgenannte Vorgehensweise favorisiert. Weiterer Aufwand fällt für die Erstellung von Modellbibliotheken an. Deren Aufbau sollte jedoch erst im Rahmen der Anwendung schrittweise erfolgen. Der anfängliche Modellierungsaufwand ist daher als wesentlich höher einzuschätzen. Die spätere Nutzung entsprechender Bausteine sowie die stringentere Gestaltung der Entwicklung dürften entsprechende Anstrengungen jedoch bereits mittelfristig egalisieren.

- **Anschaffung und Wartung der Engineeringwerkzeuge (Editoren, Simulationssysteme, usw.)**

  Für die Anschaffung von Softwarewerkzeugen zur modellgetriebenen Entwicklung und deren Wartung ist ebenfalls ein gewisser Aufwand einzuplanen. Aufgrund der unzureichenden Verfügbarkeit entsprechender Entwicklungsumgebungen sind hierzu keine quantitativen Angaben verfügbar. Langfristig kann jedoch auf alternative Entwicklungswerkzeuge verzichtet werden, so dass entsprechende Aufwände relativiert zu betrachten sind.

Ergänzend kann die vorgestellte Methodik und Vorgehensweise für die modellgetriebene Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme, wie in Abbildung 8.1 veranschaulicht, in Bezug auf die in Abschnitt 3.6 beschriebenen domänenspezifischen Anforderungen eingeordnet werden.

Methodische Anforderungen				Technologische Anforderungen							
Integration in ein mechatronisches Entwicklungsvorgehen	Förderung der fachbereichsübergreifenden Zusammenarbeit	Skalierbarkeit und Flexibilität	Effizienz und Effektivität	Integrierte Modellbildung	Interdisziplinarität und Praxisnähe der Beschreibungsmittel	Simulationsgestützte Validierung und Verifikation	Durchgängigkeit der Modellbildung	Wiederverwendbarkeit, Hierarchisierung und Modularisierung	Nutzung für weitere Aufgaben	Erweiterbarkeit	Berücksichtigung von Normen und Standards
●	●	●	◑	●	●	◑	◑	●	●	●	●

● weitgehend erfüllt ◑ teilweise erfüllt ○ kein Beitrag

Abbildung 8.1: Einordnung in Bezug auf die domänenspezifischen Anforderungen

Wie anhand der vorangegangenen Ausführungen gezeigt wurde, genügt der vorgestellte Ansatz den Anforderungen weitgehend. Da eine uneingeschränkt praxistaugliche, ergonomische Engineeringplattform im Rahmen der Arbeit aufgrund des hohen Implementierungsaufwands nur unvollständig umgesetzt werden konnte, wurde für diesbezüglich relevante Kriterien eine bedingte Erfüllung der Anforderungen angenommen. Die nur als teilweise erfüllt bewerteten Forderungen sind folglich im Fehlen geeigneter Entwicklungswerkzeuge, jedoch nicht in der erarbeiteten Methodik begründet.

9 Zusammenfassung und Ausblick

Die Konzeption, die Implementierung und der Test der Steuerungssoftware tragen verstärkt zur Wertschöpfung bei der Entwicklung und Herstellung automatisierter Fertigungssysteme bei. Die Beherrschung der zunehmenden Komplexität der Systeme, die Forderung nach einer Verkürzung der Entwicklungszeit und einer Reduktion der resultierenden Kosten sowie ein verstärkter Trend zu kundenspezifischen Lösungen bedingen jedoch eine Adaption der eingesetzten Methoden und Werkzeuge. Insbesondere eine Optimierung der abteilungsübergreifenden Zusammenarbeit, eine frühzeitige Absicherung der Produkteigenschaften sowie eine Parallelisierung und Synchronisierung der Engineeringtätigkeiten sind für die Bewältigung zukünftiger Marktanforderungen zwingend erforderlich. Grundlage hierfür ist der durchgängige Einsatz digitaler Modelle im Rahmen einer modellgetriebenen Entwicklung. Die Struktur der Produkte sowie deren Funktionen und Eigenschaften werden unter Nutzung geeigneter Modellierungstechniken am Rechner abgebildet und schrittweise verfeinert. Die Modelle sind zentraler Bestandteil der Entwicklung. Fertigungszeichnungen, Dokumentationen und weitere Projektunterlagen werden direkt aus den Modellen abgeleitet. Während ein derartiger Ansatz in einigen Domänen des Produktionsanlagenbaus bereits umgesetzt ist, wird Modellen im Umfeld der Softwareentwicklung für automatisierte Fertigungssysteme lediglich eine untergeordnete Bedeutung beigemessen. Ihr Gebrauch beschränkt sich weitestgehend auf die Dokumentation einzelner Funktionen und Abläufe. Ein durchgängiger Einsatz bei der Entwicklung ist bislang nicht Stand der Technik.

Auf der Basis theoretischer Überlegungen mit konkretem Anwendungsbezug wird in der vorliegenden Arbeit ein Beitrag zur modellgetriebenen Entwicklung der Steuerungssoftware für automatisierte Fertigungssysteme geleistet. Als Grundlage für die Ausarbeitung eines geeigneten Ansatzes erfolgte zunächst eine detaillierte Untersuchung typischer Entwicklungsabläufe sowie des Aufbaus von Fertigungssystemen. Basierend auf den gewonnenen Erkenntnissen sowie einer Analyse zukünftiger Trends in der Produktions- und Automatisierungstechnik wurden Anforderungen an einen Entwicklungsprozess abgeleitet. Ebenso war eine Diskussion und Bewertung des aktuellen Standes der Forschung und Technik Grundlage für die Ausarbeitung geeigneter Modellierungstechniken und einer Vorgehensweise zur Softwareentwicklung für automatisierte Fertigungssysteme.

Auf der Grundlage der genannten Vorarbeiten wurde ein Rahmenkonzept für eine modellgetriebene Entwicklung der Steuerungssoftware vorgestellt. Fertigungssysteme wurden nach systemtechnischen Gesichtspunkten bewertet und klassifiziert. Daran schloss sich die Vorstellung eines geeigneten, mehrstufigen Vorgehensmodells für die Softwareentwicklung an. Ergänzend wurden verschiedene Abstraktionsebenen für die Modellbildung definiert und den jeweiligen Entwicklungsphasen zugeordnet. Um eine zielgerichtete Umsetzung der gewünschten Softwarefunktionalität zu ermöglichen, sind zweckmäßige Modellierungstechniken notwendig, die zum einen eine konsistente

Modellbildung und Detaillierung erlauben und zum anderen an den jeweiligen Stand der Entwicklung angepasst sind. Daher wurden auf Basis der erarbeiteten Anforderungen prinzipiell geeignete Techniken ausgewählt und zu einem Gesamtkonzept integriert. Da eine simulationsgestützte Absicherung der Produkteigenschaften in den einzelnen Engineeringphasen zentraler Bestandteil des modellgetriebenen Entwicklungsparadigmas ist, wurden entsprechende Technologien untersucht und den jeweiligen Abstraktionsebenen zugeordnet.

Trotz der immanenten Vorteile der ausgewählten Modellierungstechniken sind für deren praxisgerechten Einsatz Anpassungen erforderlich. Im Sinne einer einfachen Anwendbarkeit war der Umfang der verfügbaren Sprachmittel zu reduzieren und auf das Notwendigste zu beschränken. Unzulänglichkeiten einzelner Sprachbausteine der Modellierungssprache, wie beispielsweise Schwächen in Bezug auf die Formalität, wurden durch geeignete Präzisierungen und Erweiterungen beseitigt. Insbesondere wurde auf eine hohe Affinität der ausgewählten Methoden zu in den beteiligten Entwicklungsdomänen bekannten Konzepten sowie auf eine fachbereichsübergreifende, generische Anwendbarkeit geachtet. Somit wird die Grundvoraussetzung für eine einfache, frühzeitige und zielgerichtete Abstimmung der Entwicklung geschaffen. Die durchgängige Weiterverwendung der in den Modellen enthaltenen Informationen wurde ebenfalls betrachtet. Prinzipiell anwendbare Konzepte wurden beschrieben, Möglichkeiten zur Transformation von Modellen dargestellt und diskutiert.

Zur Prüfung der Einsatztauglichkeit der beschriebenen Methoden und Vorgehensweisen erfolgte zunächst eine prototypische Umsetzung in Form einer integrierten Entwicklungsumgebung. Anschließend wurde die Tragfähigkeit der erarbeiteten Lösungen anhand der Anwendung bei der Entwicklung von automatisierten Fertigungssystemen bzw. ausgewählten Teilsystemen nachgewiesen. Durch eine qualitative und soweit möglich auch quantitative Bewertung der vorgestellten Konzepte aus ökonomischer Sicht konnte deren genereller Nutzen herausgearbeitet und dargestellt werden.

Potentiale für weiterführende Arbeiten bieten sich im Hinblick auf eine optimale Weiterverwendung der Modelle in anderen Entwicklungsdomänen. Insbesondere eine verbesserte Integration der dargestellten Methoden mit den Abläufen zur Planung der elektrischen, elektromechanischen und fluidtechnischen Komponenten weist vielversprechende Ansatzpunkte auf. Aber auch eine Beseitigung noch vorhandener Brüche in der Durchgängigkeit der Daten ist neben einer Beantwortung ergonomischer Fragestellungen für eine effiziente Anwendung noch erforderlich. Eine weitere Herausforderung stellt der zunehmende Trend zum Einsatz intelligenter Feldgeräte dar, der längerfristig eine Erweiterung der Modellierungssprache als notwendig erscheinen lässt. Ebenso ist eine Nutzung des Ansatzes bei der Entwicklung der Bedien- und Beobachtungssoftware als vorteilhaft zu betrachten. Dies ist im vorgestellten Konzept zwar bereits ansatzweise umgesetzt. Dennoch ist insbesondere durch die Berücksichtigung von bestehenden Standards und Datenformaten eine weitere Optimierung möglich.

10 Literaturverzeichnis

Abel 1990
Abel, D.: Petri-Netze für Ingenieure. Berlin: Springer 1990.

Alanen u. a. 2004
Alanen, M.; Lilius, J.; Porres, I.; Truscan, D.: Model Driven Engineering: A Position Paper. In: Kishinevsky, M. (Hrsg.): Proceedings of the 4th International Conference on Application of Concurrency to System Design, Hamilton (Canada). Los Alamitos (CA): IEEE Computer Society 2004.

Albert u. a. 1999
Albert, J.; Holzmüller, T.; Jünger, B.; Kaiser, O.; Kriesel, W.; Bender, K. (Hrsg.): Echtzeitsimulation zum Test von Maschinensteuerungen, München. München: Utz 1999.

Albert 1998
Albert, J.: Software-Architektur für virtuelle Maschinen. München: Utz 1998. (Informationstechnik im Maschinenwesen 8).

Albertz 1995
Albertz, F.: Dynamikgerechter Entwurf von Werkzeugmaschinen-Gestellstrukturen. Berlin: Springer 1995. (iwb Forschungsberichte 93).

Anderl u. a. 2005
Anderl, R.; Schöfer, F.; Spieß, D.: Virtueller Test von Software-Funktionen durch Koppelung mit der Produktgeometrie. In: Gausemeier, J. (Hrsg.); Rammig, F. (Hrsg.); Schäfer, W. (Hrsg.); Wallascheck, J. (Hrsg.): 3. Paderborner Workshop Intelligente Mechatronische Systeme. Paderborn: Bonifatius 2005 (HNI-Verlagsschriftenreihe Bd. 163).

Anderl & Trippner 2000
Anderl, R.; Trippner, D. (Hrsg.): STEP Standard for the Exchange of Product Model Data – Eine Einführung in die Entwicklung, Implementierung und industrielle Nutzung der Normenreihe ISO 10303 (STEP). Stuttgart: Teubner 2000.

Angermann u. a. 2002
Angermann, A.; Beuschel, M.; Rau, M.; Wohlfarth, U.: Matlab-Simulink-Stateflow – Grundlagen, Toolboxen, Beispiele. München: Oldenbourg 2002.

Baudisch 2003
Baudisch, T.: Simulationsumgebung des mechatronischen Systems Werkzeugmaschine. München: Utz 2003. (iwb Forschungsberichte 179).

Bayrak u. a. 2008

Bayrak, G.; Abrishamchian, F.; Vogel-Heuser, B.: Effiziente Steuerungsprogrammierung durch automatische Modelltransformationen von Matlab/Simulink/Stateflow nach IEC 61131-3. atp – Automatisierungstechnische Praxis 50 (2008) 12, S. 49-55.

BDI u. a. 2005

BDI – Bundesverband der Deutschen Industrie e. V. (Hrsg.); FhG – Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e. V. (Hrsg.); VDMA – Verband Deutscher Maschinen- und Anlagenbau e. V. (Hrsg.): Intelligenter Produzieren – 32 Thesen zur Forschung für die Zukunft der industriellen Produktion. Berlin: BDI – Bundesverband der Deutschen Industrie e. V. 2005.

Beckering 2003

Beckering, H.: Funktionsbeschreibung für Maschinenabläufe. In: Simulationstechniken zur Verkürzung der Durchlaufzeiten bei der Entwicklung von Werkzeugmaschinen. Frankfurt: Verein deutscher Werkzeugmaschinenfabriken e. V. 2003.

Bender 1999

Bender, K. (Hrsg.): Echtzeitsimulation zum Test von Maschinensteuerungen. München: Utz 1999.

Bender u. a. 2000

Bender, K.; Birkhofer, R; Bregulla, M; Jack, P.; Meyer, H.; Römer, M.: Zukünftige Systemtechnik der Automatisierung. Elektronik, (2000) 1, S. 76-82.

Bertenkötter u. a. 2004

Bertenkötter, K.; Bisanz, S.; Hannemann, U.; Peleska, J.: Spezifikation von Echtzeit-Automatisierungssystemen mit HybridUML. . atp – Automatisierungstechnische Praxis 46 (2004) 8, S. 54-60.

Bettin 2004

Bettin, J.: Model-Driven Software Development – An emerging paradigm for Industrialized Software Asset Development. <http://www.soft-metaware.com/mdsd-and-isad.pdf> (07.12.2006).

Bley 2003

Bley, H.: Wege zur Modellierung für die virtuelle Fabrik. In: Reinhart, G. (Hrsg.); Zäh, M. F. (Hrsg.): Marktchanche Individualisierung. 1. Auflage. Berlin: Springer 2003, S. 214-220.

BMBF 2002

BMBF – Bundesministerium für Bildung und Forschung (Hrsg.); Barrho, T.; Dreher, C.; Lenz, B.; Reinhart, G.; Spath, D.; Zielasko, W.: Werkzeugmaschine 2010 – Voruntersuchung zum Thema Werkzeugmaschine 2010. Karlsruhe 2002.

BMBF 2005

BMBF – Bundesministerium für Bildung und Forschung (Hrsg.): Positionspapier Forschungs- und Handlungsbedarf für zuverlässige mechatronische Systeme. Karlsruhe: Forschungszentrum Karlsruhe 2005.

Bock & Zühlke 2006

Bock, C.; Zühlke, D.: Werkzeugunterstützung für die modellbasierte Produktentwicklung – Maschinenlesbare Spezifikationen selbst erstellen, atp – Automatisierungstechnische Praxis 48 (2006) 7, S. 42-49.

Böger 1998

Böger, F. H.: Herstellerübergreifende Konfiguration modularer Werkzeugmaschinen. Düsseldorf: VDI-Verlag 1998. (VDI-Fortschrittberichte, Reihe 2: Fertigungstechnik 462).

Boehm u. a. 1998

Boehm, B.; Egyed, A.; Kwan, J.; Port, D.; Shah, A.; Madachy, R.: Using the WinWin Spiral Model: A Case Study, IEEE-Computer 31 (1998) 7, S. 33-44.

Booch u. a. 2006

Booch, G.; Rumbaugh, J.; Jacobson, I.: Das UML Benutzerhandbuch. München: Addison-Wesley 2006.

Bossel 1994

Bossel, H.: Modellbildung und Simulation – Konzepte, Verfahren und Modelle zum Verhalten dynamischer Systeme. Braunschweig: Viehweg 1994.

Bossel 2004

Bossel, H.: Systeme, Dynamik, Simulation – Modellbildung, Analyse und Simulation komplexer Systeme. Norderstedt: Books on Demand 2004.

Braatz 2000

Braatz, A.: UML-basierte Softwarespezifikation von dezentralen Produktionsautomatisierungssystemen. In: Bender, K.; Brandenburg, G.; Schraft, R. D. (Hrsg.): Tagungsband SPS/IPC/Drives 2000: Elektrische Automatisierung – Systeme und Komponenten, Nürnberg. Heidelberg: Hüthig 2000.

Braatz 2003

Braatz, A.: From Model to Code – Generation of PLC-Software from UML. In: Bender, K.; Brandenburg, G.; Schraft, R. D. (Hrsg.): Tagungsband SPS/IPC/Drives 2003: Elektrische Automatisierung – Systeme und Komponenten, Nürnberg. Berlin: VDE-Verlag 2003.

Braatz 2005

Braatz, A.: Entwicklung einer Methode zur objektorientierten Spezifikation von Steuerungen. Heimsheim: Jost-Jetter 2005. (IPA-IAO Forschung und Praxis 423).

Brecher u. a. 2002

Brecher, C.; Denkena, B.; Giesler, M.; Gölzer, P.; Hamann, J.; Kelichhaus, T.; Prier, M.; Prust, D.; Reinhart, G.; Weck, M.: Effiziente Entwicklung von Werkzeugmaschinen – Mit virtuellen Prototypen direkt zum marktfähigen Produkt. In: AWK – Aachener Werkzeugmaschinen-Kolloquium (Hrsg.): Aachener Perspektiven – Wettbewerbsfaktor Produktionstechnik, Aachen. Aachen: Shaker 2002.

Brockhaus 2006

Brockhaus F. A.: Brockhaus – Die Enzyklopädie in 30 Bänden. 21. Aufl. Mannheim: F. A. Brockhaus 2006.

Broy & Rumpe 2007

Broy, M.; Rumpe, B.: Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. Informatik-Spektrum 30 (2007) 1, S. 3-18.

Bruns 1991

Bruns, M.: Systemtechnik – Ingenieurwissenschaftliche Methodik zur interdisziplinären Systementwicklung. Berlin: Springer 1991.

Chouikha u. a. 2000

Chouikha, M.; Decknatel, G.; Drath, R.; Frey, G.; Müller, C.; Simon, C.; Thieme, J.; Wolter, K.: Petri Net-Based Descriptions for Discrete-Continuous Systems. at – Automatisierungstechnik 48 (2000) 9, S. 415-425.

CIRP 2004

Internationale Forschungsgemeinschaft für mechanische Produktionstechnik: Wörterbuch der Fertigungstechnik – Band 3 Produktionssysteme. 1. Aufl. Berlin: Springer 2004.

Danzer u. a. 2006

Danzer, B.; Russ, M. Koritkiy, D.: Realisierung eines Funktionsprüfstandes für virtuelle eingebettete Systeme. In: Gausemeier, J.; Rammig, F.; Schäfer, W.; Trächtler, A.; Wallaschek, J. (Hrsg.): 4. Paderborner Workshop Entwurf Mechatronischer Systeme. Paderborn: HNI 2006 (HNI-Verlagsschriftenreihe Bd. 189).

Davidson & McWhinnie1996

Davidson, C. M.; McWhinnie, J.: The Use of Object Oriented Methods in PLC Software Development. In: Proceedings of the 1th international PLCopen conference on Industrial Control Programming, Paris. Paris: Zaltbommel 1996.

Denier & Otto 2005

Denier, J. P.; Otto, S. R.: An Introduction to Programming and Numerical Methods in MATLAB. London: Springer 2005.

Denkena u. a. 2004

Denkena, B.; Tracht, K.; Rehling, S.: Multidisciplinary Simulation of Manufacturing Systems and Processes. In: Monostori, L. (Hrsg.): Proceedings of the 37th CIRP International Seminar on Manufacturing Systems, Budapest. Budapest: MTA SZTAKI 2004, S. 43-46.

Dierßen 2002

Dierßen, S.: Systemkopplung zur komponentenorientierten Simulation digitaler Produkte. Düsseldorf: VDI-Verlag 2002. (VDI-Fortschrittberichte, Reihe 20: Rechnergestützte Verfahren 358).

DIN 16566-3

DIN 16566-3, Teil 3: Elektronisches Geschäftswesen: Geschäftsprozessmanagement in der öffentlichen Verwaltung: Vorgehensmodell: Begriffe. Berlin: Beuth 2006.

DIN 19223

DIN 19223: Leittechnik: Regeln für die Benennung von Messgeräten. Berlin: Beuth 2002.

DIN 19226-1

DIN 19226, Teil 1: Leittechnik: Regelungs- und Steuerungstechnik: Allgemeine Grundbegriffe. Berlin: Beuth 1994.

DIN 19226-5

DIN 19226, Teil 5: Leittechnik: Regelungs- und Steuerungstechnik: Funktionelle Begriffe. Berlin: Beuth 1994.

DIN 40150

DIN 40150: Begriffe zur Ordnung von Funktions- und Baueinheiten. Berlin: Beuth 1979.

DIN 66025-1

DIN 66025, Teil 1: Programmaufbau für numerisch gesteuerte Werkzeugmaschinen: Allgemeines. Berlin: Beuth 1983.

DIN 66025-2

DIN 66025, Teil 2: Programmaufbau für numerisch gesteuerte Werkzeugmaschinen: Wegbedingungen und Zusatzfunktionen. Berlin: Beuth 1988.

DIN 69651

DIN 69651: Werkzeugmaschinen: Klassifizierung der Werkzeugmaschinen für Metallbearbeitung, Nummernschema, Kennzahlenschlüssel für Maschinengattungen. Berlin: Beuth 1981.

DIN EN 60848

DIN EN 60848: GRAFCET – Spezifikationssprache für Funktionspläne der Ablaufsteuerung. Berlin: Beuth 2002.

DIN EN 61131-1

DIN EN 61131, Teil 1: Speicherprogrammierbare Steuerungen: Allgemeine Informationen. Berlin: Beuth 2004.

DIN EN 61131-3

DIN EN 61131, Teil 3: Speicherprogrammierbare Steuerungen: Programmiersprachen. Berlin: Beuth 2004.

DIN EN 61499-1

DIN EN 61499, Teil 1: Funktionsbausteine für industrielle Leitsysteme: Architektur. Berlin: Beuth 2006.

DIN V 19233

DIN V 19233: Leittechnik: Prozessautomatisierung: Automatisierung mit Prozessrechnersystemen: Begriffe. Berlin: Beuth 1994.

Douglas 2003

Douglas, B. P.: Real-Time UML – Developing Efficient Objects for Embedded Systems. Boston: Addison-Wesley 2003.

Duden 2004

Hartmann, E.; Hein, C. (Hrsg.): Duden – Technik. 2. Aufl. Mannheim: Dudenverlag 2004.

Dumke 2001

Dumke, R.: Software Engineering. 3. Aufl. Wiesbaden: Vieweg 2001.

Duden 2005

Wissenschaftlicher Rat der Dudenredaktion (Hrsg.): Duden – Das Fremdwörterbuch. 8. Aufl. Mannheim: Dudenverlag 2005.

Eckes 2007

Eckes, R.: Augmented Reality – basiertes Verfahren zur Unterstützung des Anlaufprozesses von automatisierten Fertigungssystemen. Paderborn: Heinz-Nixdorf-Institut 2007. (HNI-Verlagsschriftenreihe 206).

Ehrlenspiel 2003

Ehrlenspiel, K.: Integrierte Produktentwicklung – Denkabläufe, Methodeneinsatz, Zusammenarbeit. München: Hanser 2003.

Eisner 2002

Eisner, H.: Essentials of Project and Systems Engineering Management. 2. Edition. New York: Wiley and Sons 2002.

Engell u. a. 2002

Engell, S.; Frehse, G.; Schnieder, E. (Hrsg.): Modelling, Analysis and Synthesis of Hybrid Dynamical Systems. Heidelberg: Springer 2002.

Englberger u. a. 2003

Englberger, G.; Guserle, R.; Lercher, B.; Pörnbacher, C.: Die virtuelle Werkzeugmaschine als integrierter Entwicklungsarbeitsplatz. In: Zäh, M. F.; Reinhart, G. (Hrsg.): Mechatronische Produktionssysteme – Die virtuelle Werkzeugmaschine, Garching. München: Utz 2003. (iwb Seminarberichte 66).

Fondement & Silaghi 2004

Fondement, F.; Silaghi, R.: Defining Model Driven Engineering Processes. In: Baar, T. (Hrsg.): Proceedings of the 7th International Conference on Unified Modeling Language, Lissabon. Berlin: Springer 2004.

Frankel 2003

Frankel, D. S.: Model Driven Architecture – Applying MDA to Enterprise Computing. Indianapolis: Wiley and Sons 2003.

Frey 2002

Frey, G.: Design and formal Analysis of Petri-Net based Logic Control Algorithms. Aachen: Shaker 2002.

Frey & Bossert 2004

Frey, T.; Bossert, M.: Signal- und Systemtheorie. 1. Aufl. Stuttgart: Teubner 2004.

Freund u. a. 2000

Freund, E.; Hypki, A.; Pensky, D.: Innovative Systeme zur durchgängigen Simulation und Steuerung robotergestützter Arbeitszellen. In: VDI/VDE Gesellschaft für Mess- und Automatisierungstechnik (Hrsg.): Robotik 2000 – Leistungsstand, Anwendungen, Visionen, Trends, Berlin. Düsseldorf: VDI-Verlag 2000. (VDI-Berichte 1552).

Freund & Schluse 2004

Freund, E.; Schluse, M.: Zustandsorientierte Modellierung in der 3D-Simulations- und Steuerungstechnik. atp – Automatisierungstechnische Praxis 46 (2004) 10, S. 80-86.

Fritzson 2004

Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Piscataway (New Jersey): IEEE Press 2004.

Gausemeier u. a. 2000a

Gausemeier, J. (Hrsg.); Lindemann, U.; Reinhart, G.; Wiendahl, H-P.: Kooperatives Produktengineering: Ein neues Selbstverständnis des ingenieurmäßigen Wirkens. Paderborn: Bonifatius 2000 (HNI-Verlagsschriftenreihe Bd. 79).

Gausemeier u. a. 2000b

Gausemeier, J.; Lückel, J.; Flath, M.; Seuss, J.: Design Methodology and Modelling of Mechatronic Systems, Exemplified by a Novel Approach to Vehicle Convoy Driving. In: Isermann, R. (Hrsg.): Proceedings of the 1th IFAC Conference on Mechatronic Systems, Darmstadt. Oxford: Elsevier Science Ltd. 2000.

Gausemeier u. a. 2004

Gausemeier, J.; Binger, V.; Dreher, C.; Kinkel, S.: WZM 20XX – Initiative für die Werkzeugmaschine von morgen. ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb 99 (2004) 4, S. 146-151.

Gausemeier & Lückel 2000

Gausemeier, J.; Lückel, J.: Entwicklungsumgebungen Mechatronik – Methoden und Werkzeuge zur Entwicklung mechatronischer Systeme. Paderborn: Bonifatius 2000 (HNI-Verlagsschriftenreihe Bd. 80).

Geisberger 2005

Geisberger, E.: Requirements Engineering eingebetteter Systeme – ein interdisziplinärer Modellierungsansatz. Aachen: Shaker 2005. (Berichte aus der Softwaretechnik).

Gevatter & Grünhaupt 2006

Gevatter, H. J.; Grünhaupt, U. (Hrsg.): Handbuch der Mess- und Automatisierungstechnik in der Produktion. 2. Aufl. Berlin: Springer 2006.

Gewald & Mikk 2003

Gewald, N.; Mikk, E.: Ein industrielles Wiederverwendungskonzept für IEC 61131-3 auf der Basis von UML, atp – Automatisierungstechnische Praxis 45 (2003) 6, S. 59-68.

Girod u. a. 2005

Girod, B.; Rabenstein, R.; Stenger, A.: Einführung in die Systemtheorie. 3. Aufl. Stuttgart: Teubner 2005.

GMA 2003

GMA – VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik (Hrsg.): Automatisierungstechnik 2010. Düsseldorf GMA – VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik 2003.

Gonzalez & Parkin 2003

Gonzalez-Villela, V. J.; Parkin, R. M.: A model for mechatronics design with embedded micro-controllers. In: Parkin, R. M.; Al-Habaibeh, A.; Jackson, M. R. (Hrsg.): ICOM 2003 – International Conference on Mechatronics, Loughborough. London: Professional Engineering Publishing 2003.

Gräb 2001

Gräb, R.: Parametrische Integration von Produktmodellen für die Entwicklung mechatronischer Produkte. Aachen: Shaker 2001. (Forschungsberichte aus dem Fachgebiet Datenverarbeitung in der Konstruktion 9).

Großmann 2002

Großmann, K.: Was ist, soll und kann die virtuelle Werkzeugmaschine?. In: Großmann, K. (Hrsg.): Was kann die virtuelle Werkzeugmaschine? Tagungsband zum 4. Dresdner WZM-Fachseminar, Dresden. Dresden: Technische Universität Dresden 2002.

Großmann & Mühl 2004

Grossmann, K.; Mühl; A.: Gekoppelte Simulation der Systemdynamik von Werkzeugmaschine und Zerspanprozesss. In: Zäh, M. F.; Reinhart, G. (Hrsg.): Mechatronik – Strukturdynamik von Werkzeugmaschinen, Garching. München: Utz 2004. (iwb Seminarberichte 70).

Großmann & Wunderlich 2002

Großmann, K.; Wunderlich, B.: Die virtuelle Werkzeugmaschine als Grundlage zur prozessaktuellen Fehlerkorrektur. In: Großmann, K. (Hrsg.): Was kann die virtuelle Werkzeugmaschine? Tagungsband zum 4. Dresdner WZM-Fachseminar, Dresden. Dresden: Technische Universität Dresden 2002.

Grupp & Grupp 2006

Grupp, F. Grupp, F.: MATLAB 7 für Ingenieure – Grundlagen und Programmierbeispiele. 4. Aufl. München: Oldenbourg 2006.

Guserle & Zäh 2005

Guserle, R.; Zäh, M. F.: Application of multidisciplinary simulation and optimization of mechatronic systems in the design process. In: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Monterey (CA). Piscataway (NJ): IEEE Operations Center 2005.

Haberfellner u. a 2002

Haberfellner, R.; Nagel, P.; Becker, M.; Büchel, A.; von Massow, P.: Systems Engineering. 11. Aufl. Zürich: Industrielle Organisation 2002.

Hardebusch 2002

Hardebusch, C.: Entwicklung offener NC-Steuerungen – Methoden und Werkzeuge. Aachen: Shaker 2002. (WZL RWTH Aachen – Berichte aus der Produktionstechnik).

Heimann u. a. 2001

Heimann, B.; Gerth, W.; Popp, K.: Mechatronik: Komponenten, Methoden, Beispiele. 2. Aufl. München: Hanser 2001.

Henzinger 1996

Henzinger, T. A.: The Theory of Hybrid Automata. In: Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS), Los Alamitos (California). New Brunswick (NJ): IEEE Press 1996.

Herkommer 2003

Herkommer, G.: Der modulare Schaltschrank. Computer & Automation, 5 (2003) 11, S. 100-103.

Hess 2005

Hess, D.: Objektorientierte Erweiterung der IEC 61131-3. atp – Automatisierungstechnische Praxis 47 (2005) 11, S. 72-75.

Heverhagen 2003

Heverhagen, T.: Integration von Sprachen für Speicherprogrammierbare Steuerungen in die Unified Modeling Language durch Funktionsbausteinadapter. Aachen: Shaker 2003. (Berichte aus der Automatisierungstechnik).

Hitz u. a. 2005

Hitz, M.; Kappel, G.; Kapsammer, E.; Retschitzegger, W.: UML @ Work – Objektorientierte Modellierung mit der UML 2.0. 3. Aufl. Heidelberg: dpunkt 2005.

Hopcroft u. a. 2002

Hopcroft, J. E.; Motwani, R.; .Ullman, J. D.: Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie. 2. Aufl. München: Addison-Wesley 2002.

INCOSE 2006

International Council on Systems Engineering (INCOSE): Systems Engineering Handbook – A Guide for System Life Cycle Processes and Activities, 3. Version. <http://www.incose.org > (19.09.2006).

Isermann 2002

Isermann, R. (Hrsg.): Mechatronische Systeme – Grundlagen. 1. Auflage. Berlin: Springer 2002.

Isermann 2003

Isermann, R.: Mechatronic Systems – Fundamentals. London: Springer 2003.

Jeckle u.a. 2004

Jeckle, M.; Rupp, C.; Hahn, J.; Zengler, B.; Queins, S.: UML 2 Glasklar. München: Hanser 2004.

Kallenbach u. a. 2001

Kallenbach, E.; Zöppig, V.; Birli, O.; Feindt, K. Ströhla, T.; Saffert, E.; Schmidt, J.: Integration mechatronischer Systeme. In: Verein Deutscher Ingenieure (Hrsg.): Innovative Produktentwicklungen, Frankenthal. Düsseldorf: VDI-Verlag 2001. (VDI-Berichte 1631).

Kallmeyer 1998

Kallmeyer, F.: Eine Methode zur Modellierung prinzipieller Lösungen mechatronischer Systeme. Paderborn: Bonifatius 2000 (HNI-Verlagsschriftenreihe Bd. 42).

Kent 2002

Kent, S.: Model Driven Engineering. In: Buttler, M. (Hrsg.): Proceedings of the 3th International Conference on Integrated Formal Methods, Turku (Finland). Berlin: Springer 2002.

Koch 1996

Koch, M. R.: Autonome Fertigungszellen – Gestaltung, Steuerung und integrierte Störungsbehandlung. Berlin: Springer 1996. (iwb Forschungsberichte 98).

Koller 1998

Koller, R.: Konstruktionslehre für den Maschinenbau. Berlin: Springer 1998.

Kohring 1993

Kohring, A.: Systematisches Projektieren und Testen von Steuerungssoftware für Werkzeugmaschinen. Aachen: Shaker 1993. (WZL Aachen – Berichte aus der Produktionstechnik).

Korajda u. a. 2004

Korajda, I.; Seyfarth, M.; Pritschow, G.: Disziplinübergreifende Baukastensysteme – Konfigurieren von Fertigungseinrichtungen aus einem disziplinübergreifenden Baukasten. wt Werkstattstechnik online 94 (2004) 5, S. 215-219.

Krauss 2004

Krauss, P.: Softwareentwicklung für mechatronische Applikationen. In: VDI-Wissensforum (Hrsg.): Mechatronischer Systementwurf, Darmstadt. Düsseldorf: VDI-Verlag 2004 (VDI-Berichte 1842).

Krebs & Schnieder 2000

Krebs, V.; Schnieder, E.: Special Issue on Hybrid Systems I: Modeling and Control, at – Automatisierungstechnik 48 (2000) 9, S. 413-467.

Krebs & Schnieder 2001

Krebs, V.; Schnieder, E.: Special Issue on Hybrid Systems II: Analysis, Modeling and Verification, at – Automatisierungstechnik 49 (2001) 2, S. 51-96.

Kreusch 2002

Kreusch, K.: Verifikation Numerischer Steuerungen an virtuellen Werkzeugmaschinen. Aachen: Shaker 2002. (Berichte aus der Steuerungs- und Regelungstechnik).

Kyura & Oho 1996

Kyura, N.; Oho, H.: Mechatronics – An Industrial Perspective. In: IEEE/ASME Transactions on Mechatronics 1 (1996) 1, S. 10-15.

Langlotz 2000

Langlotz, G.: Ein Beitrag zur Funktionsstrukturierung innovativer Produkte. Aachen: Shaker 2000. (Forschungsberichte aus dem Institut für Rechneranwendung in Planung und Konstruktion der Universität Karlsruhe 2/2000).

Lee & Varaiya 2003

Lee, E. A.; Varaiya P.: Structure and Interpretation of Signals and Systems. Boston: Addison-Wesley 2003.

Lepers 2005

Lepers, H.: SPS Programmierung nach IEC 61131-3. Poing: Franzis 2005.

Lercher 2008

Lercher, B.: Konzeption und System einer Integrationsplattform zur Entwicklung von Werkzeugmaschinen. München: Dissertation TU München 2008. <http://mediatum2.ub.tum.de/doc/636085/636085.pdf > (14.12.08).

Lindemann 2005

Lindemann, U.: Methodische Entwicklung technischer Produkte. Berlin: Springer 2005.

Lippolt 2001

Lippold, C.: Eine domänenübergreifende Konzeptionsumgebung für die Entwicklung mechatronischer Systeme. Aachen: Shaker 2001. (Schriftenreihe Institut für Konstruktionstechnik 9).

Litto u. a. 2004

Litto, M.; Korajda, I.; Mangold, C.; Angerbauer, R.; Hils, W.; Lerche, M.: Baukastenorientiertes Engineering mit Föderal – Ein Leitfaden für Maschinen- und Anlagenbauer. Frankfurt am Main: VDMA-Verlag 2004.

Litto & Lewek 2005

Litto, M.; Lewek, J.: Interdisziplinäres, funktionales Engineering konventioneller Maschinen und Anlagen. In: Gausemeier, J.; Ramig, F.; Schäfer, W.; Wallaschek, J. (Hrsg.): 3. Paderborner Workshop Intelligente mechatronische Systeme, Paderborn. Paderborn: Bonifatius 2005 (HNI-Verlagsschriftenreihe Bd. 163).

Litz & Frey 1999

Litz, L.; Frei, G.: Methoden und Werkzeuge zum industriellen Steuerungsentwurf – Historie, Stand, Ausblick. at – Automatisierungstechnik 47 (1999) 4, S. 145-156.

Litz 2005

Litz, L.: Grundlagen der Automatisierungstechnik. Regelungssysteme, Steuerungssysteme, Hybride Systeme. München: Oldenbourg 2005.

Lu 2004

Lu, S. C-Y.: Beyond Concurrent Engineering – A New Foundation for Collaborative Engineering. In: International Society for Productivity Enhancement: Proceedings of ISPE/CE 2004 – 11th International Conference on Concurrent Engineering – Research and Applications, Beijing. Lisse: Balkema 2004.

Lunze 2003

Lunze, J.: Automatisierungstechnik – Methoden zur Überwachung und Steuerung kontinuierlicher und ereignisdiskreter Systeme. München: Oldenbourg 2003.

Lutz 1999

Lutz, R.: Softwaretechnik für maschinennahe Steuerungsfunktionen bei Fertigungseinrichtungen. Heimsheim: Jost-Jetter 1999. (ISW Forschung und Praxis 132).

Maier u. a. 2007

Maier, D.; Nebel, S.; Rüdele, H.; Walther, M.; Olgodin, V.; Heisel, U.; Bertsche, B.; Verl, A.; Klemm, P.: Sensorlose, vorausschauende Wartung von Vorschubantrieben an Werkzeugmaschinen – Gezielte vorbeugende Wartung durch automatisierte Zustandsbeobachtung. In: VDI Wissensforum (Hrsg.): Schwingungsüberwachung und Diagnose von Maschinen, Würzburg. Düsseldorf: VDI-Verlag 2007 (VDI-Berichte 1982).

Martin u. a 2005

Martin, C.; Urquia, A.; Dormido, S.: Modeling of Interactive Virtual Laboratories with Modelica. In: Schmitz, G. (Hrsg.): Proceedings of the 4th International Modelica Conference, Hamburg. Hamburg: University of Technology 2005.

Meier 2001

Meier, H.: Verteilte kooperative Steuerung maschinennaher Abläufe. München: Utz 2001. (iwb Forschungsberichte 155).

Meier u. a. 2004

Meier, M.; Dierssen, S.; Bahelt, J.: Die Virtuelle Maschine – oder wie die Inbetriebnahme vor der Montage möglich wird. In: Neugebauer, R. (Hrsg.): CPK 2004 – 4. Chemnitzer Produktionstechnisches Kolloquium Technologische Innovationen für die Antriebs- und Bewegungstechnik, Chemnitz. Zwickau: Verlag Wissenschaftliche Scripten 2004 (Berichte aus dem IWU 25).

Metz 1997

Metz, J.: Rechnergestützte Entwicklung von Automatisierungssystemen auf der Grundlage eines objektorientierten Vorgehensmodells. In: Gens, J. (Hrsg.): 42. Internationales Wissenschaftliches Kolloquium Informatik und Automatisierung im Zeitalter der Informationsgesellschaft, Illmenau. Illmenau: Technische Universität.

Mewes 2004

Mewes, J.: WinMOD für den Maschinen- und Anlagenbau. Hennigsdorf: Mewes 2004.

Milberg 1997

Milberg, J.: Produktion – eine treibende Kraft für unsere Volkswirtschaft. In: Reinhart, G. (Hrsg.); Milberg, J. (Hrsg.): Mit Schwung zum Aufschwung – Information, Inspiration, Innovation. Münchener Kolloquium 1997. Landsberg/Lech: Verlag Moderne Industrie 1997, S. 17-40.

Milberg 2003

Milberg, J.: Grenzen überwinden – Wachstum durch Innovation. In: Hoffmann, H. (Hrsg.); Milberg, J. (Hrsg.); Reinhart, G. (Hrsg.); Zäh, M. F. (Hrsg.): Grenzen überwinden – Wachstum der neuen Art. Münchener Kolloquium 2003. München: Utz 2003, S. 307-320.

Min u. a. 2002

Min, B. K.; Huang, Z.; Pasek, Z.; Yip-Hoi, D.; Husted, F.; Marker, S.: Integration of Real-Time Control Simulation to a Virtual Manufacturing Environment. Journal of Advanced Manufacturing Systems 1 (2002) 1, S. 67-87.

Mitchell & Gauthier 1995

Mitchell, E. E. L.; Gauthier, J. S.: ACSL: Advanced Continuous Simulation Language – Reference Manual. 11th Edition. Concord (Mass.): MGS 1995.

Modelica 2005

Modelica Association (Hrsg.): Modellica Language Specification – Version 2.2. <http://www.modelica.org/documents/ModelicaSpec22.pdf> (14.12.06).

Molt 2003

Molt, T.: Eine domänenübergreifende Softwarespezifikationstechnik für automatisierte Fertigungsanlagen. Paderborn: Bonifatius 2003 (HNI-Verlagsschriftenreihe Bd. 121).

Möhringer 2004

Möhringer, S.: Entwicklungsmethodik für mechatronische Systeme. Paderborn: Bonifatius 2004 (HNI-Verlagsschriftenreihe Bd. 156).

Neugebauer u. a 2003

Neugebauer, R.; Weidlich, D.; Kolbig, S.; Polzin, T.: Development Process Support for Machine Tools with Parallel Kinematics Using Virtual Reality Technologies. In: Bley, H. (Hrsg.); Weber, C. (Hrsg.); Hirt, G. (Hrsg.): Proceedings of the 36th CIRP International Seminar on Manufacturing Systems, Saarbrücken. Saarbrücken: Universität des Saarlandes 2003.

Niebert 2003a

Niebert, P.: Petrinetze – Ein anschaulicher Formalismus der Nebenläufigkeit (Teil 1). at – Automatisierungstechnik 51 (2003) 3, S. A5-A8.

Niebert 2003b

Niebert, P.: Petrinetze – Ein anschaulicher Formalismus der Nebenläufigkeit (Teil 2). at – Automatisierungstechnik 51 (2003) 4, S. A9-A12.

OMG 2003

Object Management Group (Hrsg.): Model Driven Architecture Guide – Version 1.0.1. <http://www.omg.org/docs/omg/03-06-01.pdf> (13.12.2006).

OMG 2005a

Object Management Group (Hrsg.): UML Superstructure Specification – Version 2.0. <http://www.omg.org/docs/formal/05-07-04.pdf> (14.12.2006).

OMG 2005b

Object Management Group (Hrsg.): UML Infrastructure Specification – Version 2.0. <http://www.omg.org/docs/formal/05-07-05.pdf> (14.12.2006).

OMG 2006

Object Management Group (Hrsg.): Object Constraint Language – Version 2.0. <http://www.omg.org/docs/formal/06-05-01.pdf> (14.12.2006).

OMG 2007

Object Management Group (Hrsg.): Systems Modeling Language – Version 1.0. <http://www.omg.org/docs/ptc/06-05-04.pdf> (14.12.2006).

Ören 2002

Ören, T. I.: Future of Modelling and Simulation: Some Development Areas. In: Wallace, J. (Hrsg.): Proceedings of the 2002 Summer Computer Simulation Conference, San Diego (CA). San Diego (CA): Society for Modeling and Simulation International (SCS) 2002.

Osmers 1998

Osmers, U.: Projektieren Speicherprogrammierbarer Steuerungen mit Virtual Reality. Karlsruhe: Grässer 1998. (Forschungsberichte aus dem Institut für Werkzeugmaschinen und Betriebstechnik der Universität Karlsruhe 87).

Otter 1995

Otter, M.: Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter. Düsseldorf: VDI-Verlag 1995. (VDI-Fortschrittberichte, Reihe 20: Rechnergestützte Verfahren 147).

Otterbach & Schütte 2004

Otterbach, R.; Schütte F.: Effiziente Funktions- und Software-Entwicklung für mechatronische Systeme im Automobil. In: Gausemeier, J. (Hrsg.); Lückel, J. (Hrsg.); Wallaschek, J. (Hrsg.): 2. Paderborner Workshop Intelligente mechatronische Systeme, Paderborn. Paderborn: Bonifatius 2004 (HNI-Verlagsschriftenreihe Bd. 145).

Pahl u. a. 2005

Pahl, G.; Beitz, W.; Feldhusen, J.; Grote, K. H.: Konstruktionslehre: Grundlagen erfolgreicher Produktentwicklung, Methoden und Anwendung. 6. Aufl. Berlin: Springer 2005.

PAS 1013

Technische Regel PAS 1013: MechaSTEP – STEP-Datenmodelle zur Simulation mechatronischer Systeme. Berlin: Beuth 2001.

Peters 2001

Peters, A.: Benutzerorientierte Leitsoftware für automatisierte Fertigungssysteme in Komponentenstruktur. Aachen: Shaker 2001. (WZL Berichte aus der Produktionstechnik 13/2001).

Pfeifer & Heiler 1987

Pfeifer, T.; Heiler, K. U.: Ziele und Anwendungen von Feldbussystemen. atp – Automatisierungstechnische Praxis 27 (1987) 12, S. 549-557.

Pickhardt 2000

Pickhardt, R.: Grundlagen und Anwendung der Steuerungstechnik – Petri-Netze, SPS, Planung. Braunschweig: Viehweg 2000.

Pörnbacher & Wünsch 2004

Pörnbacher, C.; Wünsch, G.: Abschlussbericht zum FWF-Projekt 0821 „Simulation von Maschinenabläufen an virtuellen Werkzeugmaschinen“, München. Frankfurt: Forschungsvereinigung Werkzeugmaschinen e. V. 2004.

Preuß 2002

Preuß, W.: Funktionaltransformationen: Fourier-, Laplace- und Z-Transformation. München: Hanser 2002.

Pritschow & Croon 2002

Pritschow, G.; Croon, N.: Wege zur virtuellen Werkzeugmaschine - Verschiedene Betrachtungsweisen und Modellierungsansätze. wt Werkstattstechnik online 92 (2002) 5, S. 194-199.

Pritschow & Röck 2004

Pritschow, G.; Röck, S.: „Hardware in the Loop“ Simulation of Machine Tools. In: CIRP Annals 2004 Manufacturing Technology 53/1, Krakow (Poland). Oxford: Elsevier Ldt. 2004.

Rausch & Broy 2007

Rausch, A.; Broy, M.: Das V-Modell XT – Grundlagen, Erfahrungen, Werkzeuge. 1. Aufl. Heidelberg: dpunkt 2007.

Reinhart u. a. 1998

Reinhart, G.; Ansorge, D.; Mauderer, M.; Sabbah, A.: Software in der Produktion. ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb 93 (1998) 6, S. 282-285.

Richert u. a. 2003

Richert, F.; Rückert, J.; Schlosser, A.: Vergleich von Modelica und Matlab anhand der Modellbildung eines Dieselmotors. at – Automatisierungstechnik 51 (2003) 6, S. 247-254.

Röck & Hamm 2006

Röck, S.; Hamm, C.: Echtzeitfähige Achsdynamik-Modelle für die Hardware-in-the-Loop-Simulation mit realen Steuerungssystemen. In: Tagungsband zum Abschlusskolloquium des Verbundforschungsprojekts SimCAT, Karlsruhe. Karlsruhe: wbk Institut für Produktionstechnik 2006.

Rodenacker 1991

Rodenacker, W. G.: Neue Gedanken zur Konstruktionsmethodik. Konstruktion 43 (1991) S. 330-334.

Rogério & Carvalho 2004

Rogério, P.; Carvalho, A.: An Approach towards System-Oriented Utility Automation Systems Integration. In: Drews, P. (Hrsg.): Mechatronics and Robotics 2004, Aachen. Aachen: Eysoldt 2004.

Roth 2000

Roth, K. H.: Konstruieren mit Konstruktionslkatalogen – Konstruktionslehre. 3. Aufl. Berlin: Springer 2000.

Ropohl 1999

Ropohl, G.: Allgemeine Technologie – Eine Systemtheorie der Technik. 2. Aufl. München: Hanser 1999.

Rooda & Vervoort 2006

Rooda J. E.; Vervoort, J.: Analysis of Manufacturing Systems. <http://se.wpa.wtb.tue.nl/documentation/lecnotes/mis/AMS.pdf > (19.09.2006).

Rugaber & Stirewalt 2004

Rugaber, S.; Stirewalt, K.: Model-Driven Reverse Engineering, IEEE Software 21 (2004) 4, S. 45-53.

Sabbah 2000

Sabbah, K-A.: Methodische Entwicklung störungstoleranter Steuerungen. München: Utz 2000. (iwb Forschungsberichte 139).

Schaich 2001

Schaich, C.: Informationsmodell zur fachübergreifenden Beschreibung intelligenter Produktionsmaschinen. München: Utz 2001. (Informationstechnik im Maschinenwesen 15).

Schelberg 1994

Schelberg, H. J.: Objektorientierte Projektierung von SPS-Software. Karlsruhe: Grässer 1994. (Forschungsberichte aus dem Institut für Werkzeugmaschinen und Betriebstechnik der Universität Karlsruhe 57).

Schenk 2003

Schenk, M.: Virtual Reality und Simulation – Perspektiven für Entwicklung, Test und Training in der Industrie. In: Homann, R. (Hrsg.): 17. Symposium Simulationstechnik, Magdeburg. Delft: Society for Modeling and Simulation (SCS) European Publ. House 2003.

Schlingloff u. a. 2004

Schlingloff, H.; Conrad, M.; Dörr, H.; Sühl, C.: Modellbasierte Steuergerätesoftwareentwicklung für den Automobilbereich. In: Plöderer, E.: Automotive – Safety & Security 2004 – Sicherheit und Zuverlässigkeit für automobile Informationstechnik, Stuttgart. Aachen: Shaker 2004.

Schnell & Wiedemann 2006

Schnell, G.; Wiedemann, B. (Hrsg.): Bussysteme in der Automatisierungs- und Prozesstechnik – Grundlagen, Systeme und Trends der industriellen Kommunikation. 6. Aufl. Wiesbaden: Vieweg 2006.

Schneider 2000

Schneider, C.: Strukturmechanische Berechnungen in der Werkzeugmaschinenkonstruktion. München: Utz 2000. (iwb Forschungsberichte 144).

Schnieder u. a. 1998

Schnieder, E.; Chouikha, M.; Janhsen, A.: Klassifikation und Bewertung von Beschreibungsmitteln für die Automatisierungstechnik, at – Automatisierungstechnik 46 (1998) 12, S. 582-591.

Schrüfer 1992

Schrüfer, E.: (Hrsg.): Lexikon Mess- und Automatisierungstechnik. Düsseldorf: VDI-Verlag 1992.

Schrüfer 2004

Schrüfer, E.: Elektrische Messtechnik. 8. Aufl. München: Hanser 2004.

Schweiker 2003

Schweiker, A.: Offene Numerische Steuerungen für prozessabhängige Bearbeitungen – vereinheitlichte Struktur, Funktionen und Schnittstellen. Heimsheim: Jost-Jetter 2003. (ISW Forschung und Praxis 145).

Schwindt & Landgraf 2003

Schwindt, H.; Landgraf, G.: Virtuelle Drive & Control-Komponenten. In: Zäh, M. F.; Reinhart, G. (Hrsg.): Mechatronische Produktionssysteme – Die virtuelle Werkzeugmaschine, Garching. München: Utz 2003. (iwb Seminarberichte 66).

Selic u. a. 1994

Selic, B.; Gullekson, G.; Ward, P.T.: Real-Time object-Oriented Modeling. New York: Wiley and Sons 1994.

Selic 1998

Selic, B.: Using UML for modeling complex real-time systems. In: Mueller, F.; Bestravos, A. (Hrsg.): Languages , Compilers and Tools for Embedded Systems, Montreal. Berlin: Springer 1998.

Siegler 1998

Siegler, R.: Mechatronischer Entwicklungsprozess am Beispiel Werkzeugmaschinen. In: Milberg, J.; Reinhart, G. (Hrsg.): Innovative Entwicklung von Produktionsmaschinen – Mechatronische Maschinen effizient entwickeln, München. München: Utz 1998. (iwb Seminarberichte 41).

Simon 2006

Simon, S.: Benchmarking im Werkzeugmaschinenbau – Ein Beitrag zur wettbewerbsfähigen Produktentwicklung. Aachen: Shaker 2006. (Darmstädter Forschungsberichte für Konstruktion und Fertigung).

Spath & Landwehr 2000

Spath, D.; Landwehr, R.: 3-D-Projektierung und Simulation von Ablaufsteuerungen. wt Werkstattstechnik online 90 (2000) 7/8, S. 292-296.

Stahl & Voelter 2005

Stahl, T.; Voelter, M.: Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management. 1. Aufl. Heidelberg: dPunkt 2005.

Stetter 2005

Stetter, R.: Experiences with virtual production in small and middle-sized companies. In: Zäh, M. F.; Reinhart, G. (Hrsg.): Proceedings of the 1th Conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV 2005), München. München: Utz 2005.

Storr u. a. 1999

Storr, A.; Lutz, R.; Weiner, M.: Softwarekomplexität im Werkzeugmaschinenbau beherrschen. VDI-Z Integrierte Produktion, 141 (1999) 11/12, S. 14-17.

Stützel & Russ 2005

Stützel, B.; Russ, M.: Orientierungshilfe im mechatronischen Entwicklungsprozess – das 3-Ebenen-Vorgehensmodell. atp – Automatisierungstechnische Praxis 47 (2005) 1, S. 47-50.

Tomaszunas 1998

Tomaszunas, J. J.: Komponentenbasierte Maschinenmodellierung zur Echtzeit-Simulation für den Steuerungstest. München: Utz 1998. (Informationstechnik im Maschinenwesen 3).

Unbehauen 2002

Unbehauen, R.: Systemtheorie 1 – Allgemeine Grundlagen, Signale und lineare Systeme im Zeit- und Frequenzbereich. 8. Aufl. München: Oldenbourg 2002.

Van Brussel 2005

Van Brussel, H.: Mechatronics, or how make better machines. In: VDI Wissensforum (Hrsg.): Mechatronik 2005 – Innovative Produktentwicklung, Wiesloch. Düsseldorf: VDI-Verlag 2005 (VDI-Berichte 1892).

VDI 2206

VDI-Richtlinie 2206: Entwicklungsmethodik für mechatronische Systeme. Berlin: Beuth 2004.

VDI 2221

VDI-Richtlinie 2221: Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte. Berlin: Beuth 1993.

VDI 2222

VDI-Richtlinie 2222 Blatt 1: Konstruktionsmethodik – Methodisches Entwickeln von Lösungsprinzipien. Berlin: Beuth 1997.

VDI 2223

VDI-Richtlinie 2223: Methodisches Entwerfen technischer Produkte. Berlin: Beuth 2004.

VDI 3260

VDI-Richtlinie 3260: Funktionsdiagramme von Arbeitsmaschinen und Fertigungsanlagen. Berlin: Beuth 1977.

VDI 3633-1

VDI-Richtlinie 3633, Blatt 1: Simulation von Logistik-, Materialfluss- und Produktionssystemen: Grundlagen. Berlin: Beuth 1993.

VD/VDE 2422

VDI/VDE-Richtlinie 2422: Entwicklungsmethodik für Geräte mit Steuerung durch Mikroelektronik. Berlin: Beuth 1994.

VDI/VDE 3681

VDI/VDE-Richtlinie 3681: Einordnung und Bewertung von Beschreibungsmitteln aus der Automatisierungstechnik. Berlin: Beuth 2005.

VDI/VDE 3683

VDI/VDE-Richtlinie 3683: Beschreibung von Steuerungsaufgaben – Anleitung zur Erstellung eines Pflichtenheftes. Berlin: Beuth 1986.

VDI/VDE 3694

VDI/VDE-Richtlinie 3694: Lastenheft / Pflichtenheft für den Einsatz von Automatisierungssystemen. Berlin: Beuth 2008.

Versteegen 2000

Versteegen, G.: Das V-Modell in der Praxis – Grundlagen, Erfahrungen, Werkzeuge. 1. Aufl. Heidelberg: dpunkt 2000.

Vogel-Heuser u. a. 2005

Vogel-Heuser, B.; Friedrich, D.; Katzke, U.; Witsch, D.: Usaility and benefits of UML for plant automation – some research results. atp – International 3 (2005) 1, S. 52-60.

Vogel-Heuser & Fischer 2002

Vogel-Heuser, B.; Fischer, K.: UML in der automatisierungstechnischen Anwendung – Stärken und Schwächen. atp 44 (2002) 10, S. 63-68.

Vogel-Heuser & Friedrich 2005

Vogel-Heuser, B.; Friedrich, D.: Einfluss der Modellierung auf die Qualität der Steuerungsprogrammierung in der Ingenieurausbildung. In: VDI/VDE Gesellschaft für Mess- und Automatisierungstechnik (Hrsg.): Automation als interdisziplinäre Herausforderung, Baden-Baden. Düsseldorf: VDI-Verlag 2005 (VDI-Berichte 1883).

Vogel-Heuser & Friedrich 2006

Vogel-Heuser, B.; Friedrich, D.: Nutzen von Modellierung für die Qualität und Effizienz der Steuerungsprogrammierung in der Automatisierungstechnik. atp – Automatisierungstechnische Praxis 48 (2006) 3, S. 54-60.

Vogel-Heuser & Katzke 2005

Vogel-Heuser, B.; Katzke, U.: UML-PA as an Engineering Model for Distributed Process Automation. In: Horacek, P. (Hrsg.): Proceedings of the 16th Triennial World Congress of the International Federation of Automatic Control (IFAC), Prague. Oxford: Elsevier Science Ltd. 2005.

Wagner 1997

Wagner, M.: Steuerungsintegrierte Fehlerbehandlung für maschinennahe Abläufe. Berlin: Springer 1997. (iwb Forschungsberichte 106).

Wahl 2000

Wahl, M.: Methode zur Entwicklung kostengünstiger Werkzeugmaschinen.. Aachen: Shaker 2000. (Darmstädter Forschungsberichte für Konstruktion und Fertigung).

Walker 2004

Walker, B.: Entwicklungslinien für die Werkzeugmaschine von Morgen. In: Forschungszentrum Karlsruhe: Karlsruher Arbeitsgespräche Produktionsforschung 2004 – Wege zur individualisieren Produktion, Karlsruhe: FZKA 2004. (FZKA-PFT Berichte 212).

Weck 2001

Weck, M.: Werkzeugmaschinen Fertigungssysteme 4 – Automatisierung von Maschinen und Anlagen. 5. Aufl. Berlin: Springer 2001.

Weck u. a. 2003

Weck, M.; Queins, M.; Witt, S.: Effektive Entwicklung von Werkzeugmaschinen. VDI-Z Integrierte Produktion 145 (2003) 10, S. 32-36.

Weck 2005

Weck, M.: Innovation sichert langfristig die Marktposition. wt Werkstattstechnik online 95 (2005) 7/8, S. 505.

Weck & Brecher 2005

Weck, M.; Brecher, C.: Werkzeugmaschinen – Maschinenarten und Anwendungsbereiche. 6. Aufl. Berlin: Springer 2005.

Weck & Brecher 2006

Weck, M.; Brecher, C.: Werkzeugmaschinen – Automatisierung von Maschinen und Anlagen. 6. Aufl. Berlin: Springer 2006.

Weilkiens 2005

Weilkiens, T.: Die Rolle des Systems-Engineerings. Objektspektrum 12 (2005) 3, S. 28-29.

Weilkiens 2006

Weilkiens, T.: Systems Engineering mit SysML/UML – Modellierung, Analyse, Design. 1. Aufl. Heidelberg: dpunkt 2006.

Wellenreuther & Zastrow 2001

Wellenreuther, G.; Zastrow, D.: Automatisieren mit SPS – Theorie und Praxis. 1. Aufl. Braunschweig: Viehweg 2001.

Wernicke 1996

Wernicke, J.: CAD-Mechatronik. In: Verein Deutscher Ingenieure (Hrsg.): CAD zur Verkürzung der Time to Market. Hagenburg: Network 1996.

Westkämper & Braatz 2001

Westkämper, E.; Braatz, A.: Eine Methode zur objektorientierten Softwarespezifikation von dezentralen Automatisierungssystemen mit der Unified Modeling Language (UML). at – Automatisierungstechnik 49 (2001) 5, S. 225-233.

Winner u. a. 1988

Winner, R.I.; Pennell, J. P.; Bertrand, H. E.; Slusarczuk, M. M. G.: The Role of Concurrent Engineering in Weapons System Acquisition. IDA Report R-338. Institute for Defense Analysis 1988.

Witsch u. a. 2008

Witsch, D.; Wannagat, A.; Vogel-Heuser, B.: Entwurf wiederverwendbarer Steuerungssoftware mit Objektorientierung und UML. atp – Automatisierungstechnische Praxis 50 (2008) 5, S. 54-60.

Wünsch 2006

Wünsch, G.: Wann lohnt sich eine Virtuelle Inbetriebnahme?. In: Zäh, M. F.; Reinhart, G. (Hrsg.): Virtuelle Inbetriebnahme – Von der Kür zur Pflicht?, München. München: Utz 2006. (iwb Seminarberichte 84).

Wünsch 2008

Wünsch, G.: Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme. München: Utz 2008. (iwb-Forschungsberichte 215).

Xu 2003

Xu, L.: Wiederverwendbare Modelle zur Maschinensimulation für den Steuerungstest. München: Utz 2003. (Informationstechnik im Maschinenwesen 22).

Zäh u. a. 2003

Zäh, M. F.; Wünsch, G.; Pörnbacher, C.; Ehrenstraßer, M.: Emerging Virtual Machine Tools. In: American Society of Mechanical Engineers (Hrsg.): Proceedings of the 29th ASME Design Engineering Technical Conferences and Computers and Information Engineering Conference (ASME/DETC '03), Chicago (Ill.). New York: American Society of Mechanical Engineers 2003.

Zäh & Lercher 2004

Zäh, M. F.; Lercher, B.: Einsatz von Metamodellen in der virtuellen Werkzeugmaschine. In: VDI Wissensforum (Hrsg.): Mechatronischer Systementwurf, Darmstadt. Düsseldorf: VDI-Verlag 2004 (VDI-Berichte 1842).

Zäh & Lercher 2006

Zäh, M. F.; Lercher, B.: Towards Data Integration in the Virtual Machine Tool. Production Engineering Research and Development – Annals of the German Academic Society for Production Engineering 13 (2006) 1, S. 207-210.

Zäh & Oertli 2004

Zäh, M. F.; Oertli, T.: FEM Simulation of machine tools in the scope of process planning. In: Teti, R. (Hrsg.): Intelligent Computation in Manufacturing Engineering - Proceedings of the 4th CIRP International Seminar on Intelligent Computation in Manufacturing Engineering (CIRP ICME '04), Sorrento (I). Fisciano: CUES 2004.

Zäh & Pörnbacher 2005

Zäh M. F.; Pörnbacher, C.: A Model-Based Method to Develop PLC Software for Machine Tools. In: CIRP Annals 2005 Manufacturing Technology 54/1, Antalya (Turkey). Oxford: Elsevier Ldt. 2005.

Zeigler u. a. 2005

Zeigler, B. P.; Praehofer, H.; Kim, T. G.: Theory of Modeling and Simulation – Integrating discrete event and continuous complex dynamic systems. 2. Aufl. Amsterdam: Academic Press 2005.

Zentner & Bertram 2005

Zentner, J.; Bertram, T.: Konzept einer Simulationsplattform zum domänenübergreifenden modellbasierten Entwurf mechatronischer Systeme. In: Gausemeier, J. (Hrsg.); Rammig, F. (Hrsg.); Schäfer, W. (Hrsg.); Wallascheck, J. (Hrsg.): 3. Paderborner Workshop Intelligente Mechatronische Systeme. Paderborn: Bonifatius 2005 (HNI-Verlagsschriftenreihe Bd. 163).

Zirn 2002

Zirn, O.: Modellbildung und Simulation mechatronischer Systeme. Renningen: Expert 2002.

ZVEI 2006

ZVEI – Zentralverband Elektrotechnik- und Elektronikindustrie e. V. (Hrsg.): Integrierte Technologie-Roadmap Automation 2015+. Frankfurt: Berthold 2006.

11 Anhang

11.1 Begriffsdefinitionen

In diesem Abschnitt werden die für die vorliegende Arbeit relevanten Begriffe beschrieben. Die Darstellung beschränkt sich auf die grundlegenden Definitionen und ihre Bedeutung innerhalb des Manuskripts. Begriffe von sekundärer Bedeutung werden an entsprechender Stelle im Text erläutert und gegebenenfalls mit Verweisen auf weiterführende Literaturquellen fundiert.

Automatisierungstechnik
Die Automatisierungstechnik ist ein fachübergreifendes Gebiet, das sich mit der Konzeption und Entwicklung von Automaten oder anderen automatisch ablaufenden Vorgängen befasst. Als Automat wird ein künstliches System bezeichnet, das selbstständig ein Programm befolgt *(DIN 19223)*. Auf Grund des Programms trifft das System Entscheidungen, die auf der Verknüpfung von Eingaben mit den jeweiligen Zuständen des Systems beruhen und Ausgaben zur Folge haben *(Schrüfer 1992)*. Die Definition nach *(Lunze 2003, S. 35)* stellt die eingesetzte Methodik zur Entwicklung der Steuerungsfunktionalität in den Fokus und hebt insbesondere die Unabhängigkeit von der gerätetechnischen Realisierung hervor. Der Begriff wird als Synonym zur „Technischen Kybernetik[1]“ definiert.

Automatisiertes System
Ein Automatisiertes System ist ein Steuerungssystem, in dem Speicherprogrammierbare Steuerungen (SPS) durch oder für die Anwenderinnen und Anwender eingesetzt werden, das aber auch andere Komponenten einschließlich ihrer Anwendungsprogramme enthält *(DIN EN 61131-1)*.

Automatisiertes Fertigungssystem
Ein Fertigungssystem ist ein technisches System zur Herstellung beliebiger Produkte aus Rohmaterial oder Halbzeugen *(Rooda & Vervoort 2006)*. Nach *(DIN 69651)* werden Einzelmaschinen und Mehrmaschinensysteme unterschieden. In *(CIRP 2004)* wird in Bezug auf automatisierte Fertigungssysteme insbesondere der vollständig programmgesteuerte Ablauf der Bearbeitung, des Werkzeugwechsels und des Werkstückflusses mit einem Minimum an manuellem Eingriff hervorgehoben. Im Rahmen dieser Arbeit werden in Anlehnung an *(Lutz 1999)* unter Fertigungssystemen zusammenfassend Anlagen verstanden, die aus einer oder mehreren Werkzeugmaschinen bestehen und zur Fertigung von Produkten oder Teilen davon zusammenwirken.

[1] Der Begriff „Kybernetik“ (griechisch: „Kunst des Steuerns“) wurde vom amerikanischen Mathematiker Norbert Wiener (1894-1964) geprägt. Die Technische Kybernetik beschäftigt sich mit der Planung und Umsetzung technischer Systeme.

Speicherprogrammierbare Steuerung
Eine Speicherprogrammierbare Steuerung (SPS) ist nach *(DIN 19226-5)* eine digital arbeitende elektronische Steuerung, deren Programm in einem Programmspeicher abgelegt wird. Die Art des Speichers sowie die mechanische und elektrische Integration des Speichers in die Steuerung bestimmt, in welchem Umfang und wie das Programm der Steuerung verändert werden kann.

Systems Engineering
Systems Engineering ist eine auf bestimmten Grundprinzipien beruhende Methodik zur zweckmäßigen und zielgerichteten Gestaltung komplexer Systeme *(Haberfellner u. a. 2002)*. Es ist ein iterativer Prozess zur Synthese, Entwicklung und zum Betrieb realer Systeme, die in nahezu optimaler Art und Weise die an sie gestellten Anforderungen erfüllen *(Eisner 2002)*. Die Definition von *(INCOSE 2006)* hebt insbesondere die Interdisziplinarität des Systems Engineering hervor.

Domäne
Als Domäne wird im Allgemeinen ein eingegrenztes Fach- oder Wissensgebiet bezeichnet. In der Softwaretechnik wird darunter ein abgrenzbares Problemfeld bzw. ein bestimmter Einsatzbereich für Computersysteme oder Software verstanden. Eine Domäne stellt typischerweise sehr spezielle Anforderungen an ein technisches System, welches zur Bewältigung der domänenspezifischen Aufgaben und Probleme eingesetzt werden soll. Diese Anforderungen fließen insbesondere im Rahmen der Anforderungsanalyse, die einer Systementwicklung vorausgeht, und während des Entwurfs des Systems in den Entwicklungsprozess ein und bestimmen maßgeblich die Modellbildung oder auch Modellierung, die der späteren Realisierung zugrunde liegt. Eine Domäne ist durch eine weitgehend eigenständige Begriffs- und Vorstellungswelt gekennzeichnet.

Notation
Eine Notation ist ein System von Zeichen oder Symbolen zur quantitativen und qualitativen Repräsentation von Sachverhalten *(Duden 2005)*. Dadurch wird es möglich, komplizierte Zusammenhänge kurz und eindeutig zu vermitteln.

Semantik
Als Semantik wird die Bedeutung sprachlicher Ausdrücke beziehungsweise von Zeichen im Allgemeinen verstanden. Die Aufgabe der formalen Semantik besteht darin, mithilfe logisch-mathematischer Methoden Regeln zu formulieren, durch die Ausdrücke und Sätze künstlicher (logischer) Sprachen gedeutet werden können *(Brockhaus 2006)*. Dies ist erforderlich, um Programme (z. B. Steuerungsprogramme) eindeutig zu interpretieren.

Syntax
In der Informatik bezeichnet die Syntax die Gesamtheit aller Regeln für die Bildung erlaubter Ausdrücke und damit korrekter Programme *(Brockhaus 2006)*. Dadurch ist es in Kombination mit der semantischen Bedeutung der Ausdrücke möglich, ge-

wünschte Softwarefunktionalitäten in Form einer formalen Sprache (z. B. einer Programmiersprache) rechnerinterpretierbar abzubilden.

Beschreibungsmittel
Ein Beschreibungsmittel deklariert (in graphischer Form) bestimmte Sachverhalte zur visuellen Wahrnehmung und Speicherung. Beschreibungsmittel umfassen alphanumerische Zeichen, Symbole oder sonstige graphische Darstellungselemente (Notation) sowie Konventionen bzgl. deren Kombination (Syntax). Den einzelnen Darstellungselementen sowie ihren Kombinationen und Zuordnungen werden definierte Bedingungen oder Konzepte aus einem bestimmten fachlichen Bereich zugeordnet, die mehr oder weniger detailliert und formal spezifiziert sind (Semantik) *(Schnieder u. a. 1998)*. Es werden formale, semiformale und informale Beschreibungsmittel unterschieden.

Formales Beschreibungsmittel
Ein formales Beschreibungsmittel verfügt über eine mathematische Basis und eine definierte, vollständige Syntax. Darüber hinaus verfügt es über eine eindeutige semantische Interpretation *(VDI/VDE 3681)*.

Semiformales Beschreibungsmittel
Ein semiformales Beschreibungsmittel besitzt eine definierte, vollständige Syntax sowie eine eindeutige semantische Interpretation. Im Gegensatz zum formalen Beschreibungsmittel fehlt jedoch die mathematische Basis *(VDI/VDE 3681)*.

Informales Beschreibungsmittel
Ein informales Beschreibungsmittel verfügt über die Merkmale eines Beschreibungsmittels (Semantik, Syntax und Notation). Syntax und Semantik müssen jedoch nicht grundsätzlich vollständig sein. Ebenso ist eine eindeutige semantische Interpretation nicht möglich *(VDI/VDE 3681)*. Ein Beispiel für ein derartiges Beschreibungsmittel ist die Sprache.

Artefakt
Bezeichnet ein durch menschliche oder technische Einwirkung entstandenes Produkt oder Phänomen, in Abgrenzung zu den unbeeinflussten bzw. natürlichen Phänomen *(Brockhaus 2006)*. In der Domäne der Informationstechnik bezeichnet der Begriff ein Produkt (Dokument, Quellcode, Programm), welches als Zwischen- oder Endergebnis der Softwareentwicklung entsteht.

Methode/Methodik
Unter einer Methode wird eine planmäßige, regelbasierte Vorgehensweise zum Lösen einer definierten Aufgabenstellung und zum Erreichen festgelegter Ziele verstanden. Die Kombination mehrerer Einzelmethoden zur Klärung spezifischer Fragestellungen wird dagegen unter dem Begriff Methodik eingeordnet *(Lindemann 2005)*.

System
Ein System ist eine in einem betrachteten Zusammenhang gegebene Anordnung von Komponenten bzw. Gebilden, die miteinander in Beziehung stehen. Diese Anordnung

wird aufgrund definierter Vorgaben gegenüber ihrer Umgebung abgegrenzt (Systemgrenze) *(DIN 19226-1)*. Ein System wird neben seiner Systemgrenze vor allem durch seinen Aufbau (Struktur) und sein Verhalten charakterisiert.

Struktur

Die Struktur ist die wirkungsmäßige Art und Zuordnung der Komponenten in einem System *(VDI 3633-1)*. Sie beschreibt die Gesamtheit der Beziehungen zwischen den Teilen eines Systems *(DIN 19226-1)*.

Modul

Als Modul wird ein definierter und klar abgegrenzter Teil eines größeren Gesamtsystems bezeichnet. Besonderes Kennzeichen ist die Realisierung spezifischer Aufgaben sowie einer definierten Schnittstelle. Die Komposition von Gesamtsystemen aus Modulen (Baukastenprinzip) bietet den Vorteil der einfachen Anpassung an unterschiedliche Aufgaben *(Brockhaus 2006)*.

Mechatronik

Für den Begriff Mechatronik existiert keine eindeutige Definition. Die international am weitesten verbreitete Definition geht nach *(Kallmeyer 1998)* auf die IFTMM[2] zurück. Demnach wird Mechatronik als eine synergetische Kombination aus Feinmechanik, Steuerungs- und Regelungstechnik sowie Informationstechnik verstanden. Inhaltlich entspricht dies auch der Definition des IRDAC[3]. Nach *(Wernicke 1996)* charakterisiert der Begriff eine Systementwurfstechnologie mit dem Ziel einer effizienten Integration elektrischer, informationstechnischer und mechanischer Elemente und wird somit als interdisziplinäres Entwicklungsvorgehen bezeichnet.

Mechatronisches System

Als mechatronische Systeme werden technische Systeme bezeichnet, in denen elektrische, mechanische und elektronische Funktionselemente miteinander verknüpft sind *(Kallenbach u. a. 2001)*. Diese verschmelzen zu einem untrennbaren Gesamtsystem, in dem der mechanische Teilprozess dominierend ist *(Isermann 2003, S. 11)*. Zur Einordnung mechatronischer Systeme wurde von der JSPMI[4] ein Ordnungsschema mit vier Klassen erarbeitet *(Kyura & Oho 1996)*. Fertigungssysteme sind demnach der Klasse 1 zuzuordnen, da sie traditionell rein mechanische Produkte sind, die durch den Einsatz von Informationstechnik wesentlich verbessert wurden.

Kausalität

Ein kausaler Zusammenhang wird durch eine Relation „Ereignis schafft Bedingung für das Stattfinden des Folgeereignisses“ beschrieben. Ein System ist daher als kausal zu bezeichnen, wenn die Ausgangsgrößen nur von den aktuellen oder vergangenen Ein-

[2] IFTMM = International Federation for the Theory of Machines and Mechanism.

[3] IRDAC = Industrial Research and Development Advisory Committee of the European Union.

[4] JSPMI = Japan Society for the Promotion of Machine Industry.

gangsgrößen abhängen. Im Kontext dieser Arbeit wird in Bezug auf Ein- und Ausgangsgrößen eines Systems auch deren Wirkrichtung verstanden. Größen ohne eindeutige Wirkrichtung werden auch als akausal bezeichnet.

Prozess/Fertigungsprozess

Ein Prozess ist die Gesamtheit von aufeinander einwirkenden Vorgängen in einem System, durch die Materie, Energie oder auch Informationen umgeformt, transportiert oder gespeichert wird *(DIN V 19233)*. Unter einem Fertigungsprozess wird die Gesamtheit aller Verfahren zur Herstellung eines ein- oder mehrteiligen Gebrauchsgegenstandes mit geometrisch bestimmter Gestalt in einer genau festgelegten Reihenfolge verstanden. Diese erfolgt aus formlosen Werkstoffen oder aus Formteilen unter Nutzung von Werkzeugen, Maschinen, Energie und Daten *(Duden 2004, S. 61)*.

Modell

Ein Modell ist die vereinfachte Nachbildung eines geplanten oder real existierenden Originalsystems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System. Es unterscheidet sich hinsichtlich der untersuchungsrelevanten Eigenschaften nur innerhalb eines vom Untersuchungsziel abhängigen Toleranzrahmens vom Vorbild *(VDI 3633-1)*. Die Erstellung von Modellen erfolgt aufgrund bekannter Gesetzmäßigkeiten, durch Identifikation oder aufgrund von getroffenen Annahmen.

Modellierung/Modellbildung

Nach *(Otter 1995)* ist unter Modellierung die Spezifizierung eines Modells in einer anwendernahen Form zu verstehen. Modellbildung hingegen wird als die Überführung einer solchen Definition in eine durch Rechnerwerkzeuge effizient ausführbare Standardform mittels entsprechender Transformationsalgorithmen angesehen. Da dieser Vorgang in der Regel automatisiert durch entsprechende Softwarewerkzeuge erfolgt und aufgrund der synonymen Verwendung der beiden Definitionen in der Praxis, wird im Rahmen der Arbeit der Ausdruck „Modellbildung“ gleichbedeutend mit dem Begriff „Modellierung“ verwendet.

Vorgehensmodell/Geschäftsprozess

Vorgehensmodelle dienen zum Abbilden von Geschäftsprozessen in strukturierte Phasen anhand einer abstrakten Darstellung. Unter einem Geschäftsprozess wird dabei ein Vorgang verstanden, der unter Zuhilfenahme von Regeln und Sachmitteln von den beteiligten Personen gesteuert wird, mit dem Ziel, definierte Leistungen (Produkte oder Dienstleistungen) zu erzeugen *(DIN 16566-3)*. Für einzelne Domänen (z. B. die Softwareentwicklung) wurden unterschiedliche Vorgehensmodelle entwickelt. Diese variieren in der Anzahl der differenzierten Phasen und ihrer Bedeutung. Allen Vorgehensmodellen gemeinsam ist jedoch die Darstellung der einzelnen Phasen des Geschäftsprozesses, der empfohlenen Reihenfolge der durchzuführenden Aktivitäten sowie ihrer Ergebnisse.

Modellzweck
Der Modellzweck als wichtigste Vorgabe der Modellbildung bestimmt Art und Umfang des Modells. Je genauer dieser spezifiziert wird, desto präziser kann die Modellierung durchgeführt werden. Gleichzeitig hat er einen entscheidenden Einfluss auf die Auswahl der hierzu geeigneten und notwendigen Beschreibungsmittel bzw. Modellbildungsmethoden. Dies bedeutet im Umkehrschluss aber auch, dass das gleiche System für unterschiedliche Modellzwecke durch verschiedene Modelle abgebildet werden muss *(Bossel 1994)*.

Simulation
Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Im weiteren Sinne wird unter Simulation auch das Vorbereiten, Durchführen und Auswerten gezielter Experimente mit dem Simulationsmodell verstanden *(VDI 3633-1)*. In der Definition von *(Brockhaus 2006)* wird insbesondere die Abbildung des Zeitverhaltens im Modell hervorgehoben.

Simulator/Simulationssystem
Ein Simulator ist das Werkzeug zur Simulation, das ein Modell zur Nachbildung des dynamischen Verhaltens des Systems und seiner Prozesse lauf- und nutzungsfähig macht *(VDI 3633-1)*. Erfolgt die Erstellung des Modells und die Durchführung der Experimente auf einem Rechner, so wird der Simulator auch als Simulationsprogramm bezeichnet *(Brockhaus 2006)*. Im Rahmen der Arbeit wird dies vorausgesetzt und synonym auch der Begriff Simulationssystem verwendet.

11.2 Prinzipielle Methoden zur Modellierung technischer Systeme

11.2.1 Kontinuierliche Systeme

Die mathematische Abstraktion eines technischen Produktes in Form einer kontinuierlichen Systembeschreibung beruht üblicherweise auf bekannten physikalischen Gesetzmäßigkeiten, die es erlauben, die Ausgangsgrößen als Funktionen der Zeit darzustellen. Die meist auf einfachen algebraischen Ausdrücken, Bilanzgleichungen oder Erhaltungssätzen (z. B. Prinzip der virtuellen Arbeit) basierenden Zusammenhänge werden in Form einfacher oder differential-algebraischer Gleichungen (ODE/DAE[5]) abgebildet. Für viele technische Problemstellungen resultiert als Ergebnis ein nichtlineares, verkoppeltes Gleichungssystem. Die Systemgesetzmäßigkeiten sind somit nur implizit abbildbar und demzufolge in den meisten Fällen nicht durch allgemeine Simulationspakete lösbar. Eine Möglichkeit zur Behandlung derartiger Fragestellungen bietet die Überführung des Gleichungssystems in die Zustandsraumdarstellung. Dabei handelt es sich um eine standardisierte Darstellungsform für lineare und nichtlineare

[5] ODE = "Ordinary Differential Equation", DAE = "Differential Algebraic Equation".

Differentialgleichungssysteme, in denen maximal erste Ableitungen nach der Zeit auftreten.

$$\dot{z}_k^k = f\left(x_i^k(t), z_k^k(t)\right)$$
$$y_j^k = g\left(x_i^k(t), z_k^k(t)\right)$$

Die Umformung der systembeschreibenden Gleichungen in die Zustandsdarstellung erfolgt in mehreren Schritten. Zunächst werden die Eingangsgrößen x_i^k, die Zustandsgrößen z_k^k und die Ausgangsgrößen y_j^k identifiziert. Hierzu gilt es, alle zeitveränderlichen Größen, die im ursprünglichen ODE/DAE-System in erster Ableitung auftreten als Zustandsgrößen zu deklarieren und mit $\dot{z}_k^k$ zu bezeichnen. Zeitabhängige Größen, deren Ableitung höherer Ordnung ist, sind durch Einführung zusätzlicher Zustandsvariablen auf solche erster Ordnung zurückzuführen. Unabhängig auftretende, veränderliche Eingangsgrößen werden mit x_i^k, abhängige Ausgangsgrößen mit y_j^k bezeichnet.

Ziel des zweiten Schrittes ist es, dass in jeder Gleichung höchstens eine Ableitung erster Ordnung auftritt. Nach dieser können die Gleichungen dann aufgelöst und anschließend übersichtlich geordnet werden. Diese sogenannten Zustandsgleichungen können nun in vektorieller Form mit dem Zustandsvektor $\vec{z}^k$, dem Eingangsvektor $\vec{x}^k$ und dem Ausgangsvektor $\vec{y}^k$ beschrieben werden und bilden das Zustandsraummodell des kontinuierlichen Systems. In Abbildung 11.1 wird die Vorgehensweise zur Überführung einer allgemeinen Beschreibung in ein Zustandsraummodell anhand eines einfachen Feder-Masse-Dämpfer-Systems mit geschwindigkeitsproportionaler Dämpfung aufgezeigt.

Für viele technische Fragestellungen ist es möglich, die Lösung der systembeschreibenden Differentialgleichungen nach der Überführung in die Zustandsraumdarstellung weiter zu vereinfachen. Das hierzu am häufigsten angewandte Verfahren ist die Linearisierung. Dazu werden die nichtlinearen Zustandsgleichungen durch eine für einen definierten Arbeitspunkt gültige Approximation ersetzt. Diese Methode ist beispielsweise bei der simulationsgestützten Auslegung und Implementierung von Regelungen anwendbar, wenn die Regelgröße und auch andere zeitveränderliche Größen nur wenig vom definierten Arbeitspunkt abweichen. Weitere Vereinfachungen können durch die Lösung der Gleichungen im Bildbereich und die anschließende Rücktransformation in den Zeitbereich realisiert werden. Nähere Informationen zu den dazu notwendigen mathematischen Verfahren der Laplace-Transformation bzw. Z-Transformation sind der diesbezüglichen Fachliteratur, z. B. *(Preuß 2002)* zu entnehmen.

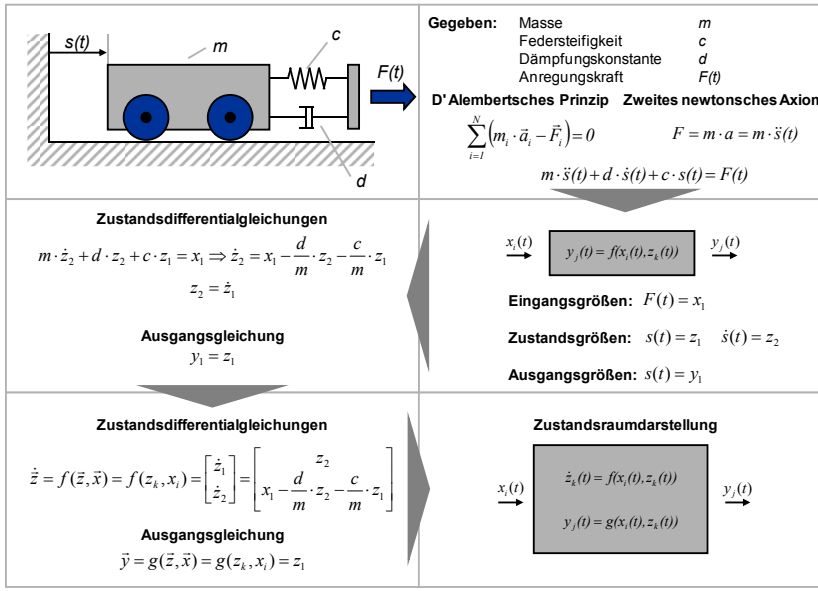


Abbildung 11.1: Feder-Masse-Dämpfer-System in Zustandsraumdarstellung

Bereits das einfache Beispiel aus Abbildung 11.1 zeigt, dass zur effizienten Untersuchung komplexer Problemstellungen generell gültige Werkzeuge benötigt werden. Für die rechnergestützte Modellierung wurden daher allgemeine Simulationssprachen entwickelt. Bekannte kommerzielle Systeme sind SIMULA oder ACSL[6] *(Mitchell & Gauthier 1995)*. Diese setzen jedoch voraus, dass die systembeschreibenden Gleichungen gegeben, durch einfaches Sortieren in eine korrekte Abarbeitungsreihenfolge gebracht und in eine Zustandsraumbeschreibung überführbar sind. Bei fast allen komplexeren physikalischen Systemen besteht die Hauptschwierigkeit aber gerade darin, diese aus einer anwendernahen Modelldefinition abzuleiten.

Blockschaltbild-Editoren[7] und -Simulatoren ermöglichen eine anschauliche und fachbereichsübergreifende Modellbildung und Simulation. Durch das graphisch unterstützte Verbinden von rückwirkungsfreien Ein/Ausgangsblöcken werden die Modelle schrittweise zusammengesetzt. Kommerzielle Systeme, wie beispielsweise MATLAB/Simulink stellen hierzu eine große Zahl vordefinierter Funktionsblöcke zur Verfügung, erlauben aber auch die Definition neuer Blocktypen. Abbildung 11.2 zeigt das Blockschaltbild des beschriebenen Feder-Masse-Dämpfer-Systems.

[6] ACSL = Advanced Continuous Simulation Language.

[7] Blockschaltbilder werden auch als Wirkschemata oder Strukturbilder bezeichnet.

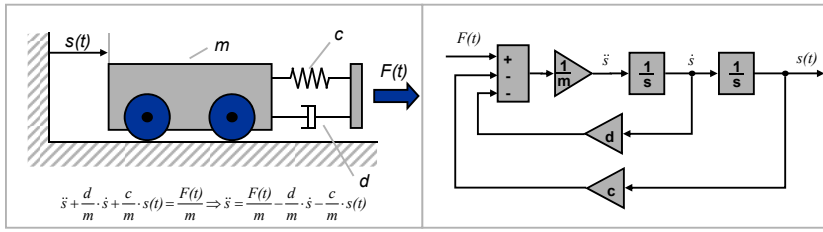


Abbildung 11.2: Feder-Masse-Dämpfer-System als Blockschaltbild

Funktionell sind sich allgemeine Simulationssprachen und Blockschaltbildeditoren sehr ähnlich. Ein Block entspricht einem Satz von Gleichungen mit definierten Eingangs- und Ausgangsgrößen sowie Zustandsvariablen. Vor der Simulation wird die Abarbeitungsreihenfolge durch (automatisches) Sortieren ermittelt. Dies entspricht dem Sortieren der Gleichungen bei Simulationssprachen. Häufig werten Blockschaltbildeditoren die Modelle während einer Simulation interpretativ aus und sind daher deutlich langsamer als Simulatoren, die in Maschinencode übersetzte Modelle nutzen.

Eine wesentliche Eigenschaft der Blockschaltbilder ist ihre Beschränkung auf die Abbildung kausaler Wirkzusammenhänge und damit die unmittelbare Umsetzung mathematischer Gleichungen in eine graphische Darstellung. Für regelungstechnische Systeme weist diese Form der Modellbildung daher eine sehr gute Eignung auf. Für die Modellierung physikalischer Systeme ist eine kausale (deklarative) Modellbildung jedoch nur bedingt einsetzbar. Dies ist damit zu begründen, dass die Überführung physikalischer Systemmodelle in Blockschaltbilder schwierig und aufwendig ist, während regelungstechnische Modelle meist schon in dieser Form vorliegen *(Otter 1995, S. 5)*. Abbildung 11.3 zeigt dies anhand der Darstellung des Modells eines primitiven elektrischen Schaltkreises, wie er beispielsweise zum Aufbau von Filtern eingesetzt wird.

Die genannten Nachteile können durch die Formulierung akausaler Modelle vermieden werden. Diese haben den Vorteil, dass zur Modellbildung lediglich die zugrunde liegenden physikalischen Gesetzmäßigkeiten beschrieben werden müssen, ohne explizit Ein- und Ausgangsgrößen zu definieren. Das bekannteste Beispiel für entsprechende Sprachen ist Modelica *(Modelica 2005)*, und deren kommerzielle Nutzung im Simulationssystem Dymola. Einzelne mathematisch formulierte Komponenten können graphisch unter Nutzung geeigneter Symbole dargestellt werden. Die Verknüpfung der notwendigen Bausteine zu einem Gesamtmodell erfolgt ebenfalls graphisch, indem die Symbole anhand von Linien verbunden werden. Akausale Modelle haben jedoch den Nachteil, dass sie nicht interpretativ auswertbar sind, sondern in eine Zustandsraumdarstellung überführt und numerisch gelöst werden müssen.

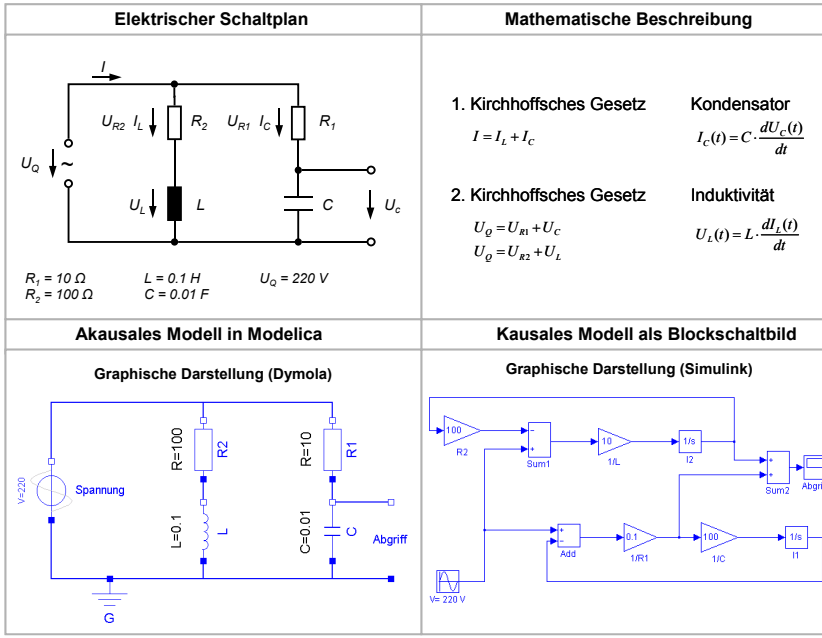


Abbildung 11.3 Akausale und kausale Modellbildung

Die Berechnung der Simulationsergebnisse erfolgt bei den meisten kommerziell verfügbaren Systemen unter Zuhilfenahme numerischer Näherungsverfahren. Als Beispiel sei an dieser Stelle als einfachster Ansatz das explizite Euler-Verfahren[8] genannt. Die Auswahl einer geeigneten Methode in Bezug auf die gegebene Problemstellung bestimmt direkt die Genauigkeit der Berechnungsergebnisse und richtet sich daher nach der Art der zugrunde liegenden Differentialgleichungen. Einen guten Überblick über gängige Verfahren der numerischen Mathematik und deren Einsatz zur Behandlung linearer und nichtlinearer Zustandsdifferentialgleichungen bietet beispielsweise *(Otter 1995, S. 26-43)* oder *(Fritzson 2004, S. 668-688)*.

11.2.2 Ereignisdiskrete Systeme

Wie in Abschnitt 2.2.4 erläutert, haben ereignisdiskrete Systeme die Eigenschaft, dass sich das Systemverhalten als Funktion eingehender Ereignisse und einer endlichen Menge möglicher Systemzustände charakterisieren lässt. Dabei wird davon ausgegan-

[8] Das explizite Euler-Verfahren ist ein Einschrittverfahren zur numerischen Lösung von Anfangswertproblemen mit konstanter Schrittweite Δt. Ausgehend von einem Zeitpunkt t wird der Zustandsvektor zum Zeitpunkt $t+\Delta t$ berechnet. Grundlage des Verfahrens ist die Taylorentwicklung der Zustandsdifferentialgleichungen, die nach dem zweiten Summanden abgebrochen wird.

gen, dass sich der Informationsgehalt der eingehenden Signale, Zustands- und Ausgangsgrößen nicht in ihrem konkreten Wert, sondern qualitativ durch das Über- bzw. Unterschreiten eines vordefinierten Grenzwertes widerspiegelt. Eingangs- und Zustandsgrößen ereignisdiskreter Systeme werden daher häufig durch binäre Wertzuweisungen oder symbolisch deklariert (z. B. Sensor betätigt oder nicht betätigt) und sind somit als wertdiskret zu betrachten. Genauso ist der genaue Zeitpunkt des Eintretens von Signal- bzw. Zustandsänderungen von untergeordneter Bedeutung. Lediglich dass sich diese ändern und in welcher Reihenfolge dies geschieht, ist für die Betrachtung relevant. Es ist demgemäß möglich, die Ein- und Ausgangsereignisse sowie die erreichbaren Systemzustände als nicht zeitbewertete, kausale Folge zu betrachten.

Ob durch die Vernachlässigung der Zeit eine wichtige Beschreibungsgröße verloren geht hängt vom Modellzweck ab. Ist lediglich interessant, welche Zustände ein System annehmen kann und wie diese durchlaufen werden, spielt der zeitliche Abstand keine Rolle und die logische Zustandsfolge ist zur Verhaltensbeschreibung hinreichend. Dies ist nach *(Lunze 2003, S. 300)* für viele praktische Fragestellungen gegeben.

Ereignisse am Systemeingang und -ausgang sowie Zustandsänderungen können zu beliebigen Zeitpunkten auftreten. Derartige, als asynchron bezeichnete Ereignisfolgen stellen bei technischen Systemen den Regelfall dar. Begründet liegt dies in der Tatsache, dass diese eine gewisse Zeit brauchen, um in Bezug auf Eingangsereignisse Zustandsänderungen bzw. resultierende Ausgangsereignisse zu generieren. Ist jedoch die Reaktionszeit des Systems im Verhältnis zum zeitlichen Abstand der Zustandsänderungen und Ausgangsereignisse wesentlich kleiner, so kann diese vernachlässigt werden. Eingangsereignisse, Zustandsänderungen und davon abgeleitete Ausgangsereignisse treten unter Anwendung einer derartigen Idealisierung gleichzeitig auf und werden als synchron bezeichnet.

In Bezug auf die genannten Eigenschaften soll das Modell eines ereignisdiskreten Systems spezifizieren, in welchen Nachfolgezustand $Z_{k+1}^d(n+1)$ dieses übergeht, wenn es sich zum diskreten Zeitpunkt *n* im Zustand $Z_k^d(n)$ befindet und die Eingangssignale $x_i^d(n)$ anliegen. Außerdem soll es festlegen, welche Ausgabe $y_j^d(n)$ in dieser Situation erzeugt wird. Das Systemmodell muss also die folgende prinzipielle Form aufweisen:

$$Z_{k+1}^d(n+1) = f\left(Z_k^d(n),\, x_i^d(n)\right), \quad Z^d(0) = Z_0^d$$
$$y_j^d(n) = g\left(Z_k^d(n),\, x_i^d(n)\right)$$

Ein Vergleich dieser formalen Darstellung mit der Zustandsraumdarstellung für kontinuierliche Systeme zeigt, dass die Modellbildungsaufgabe im Wesentlichen dieselbe ist. Allerdings werden die Zustandsänderungen bei kontinuierlichen Systemen durch den aktuellen Zustand $Z_k^k(t)$ sowie den Differenzialquotienten $\dot{Z}_k^k(t)$ definiert. Eine derartige kompakte Beschreibung der Zustandsübergänge existiert für die meisten er-

eignisdiskreten Systeme nicht. Alle Zustandsänderungen müssen folglich einzeln betrachtet und beschrieben werden.

Aus den dargestellten Unterschieden resultiert eine gänzlich andere mathematische Behandlung ereignisdiskreter Systeme und in der Folge der Einsatz vollkommen anderer Modellformen. Die Schwierigkeit das Systemverhalten durch analytische Ausdrücke zu beschreiben ist auch die Ursache dafür, dass zur Modellierung ereignisdiskreter Systeme eine Vielzahl von Beschreibungsmitteln entwickelt wurden und eingesetzt werden. Im Folgenden werden einige wichtige Grundformen derartiger Systemspezifikationen vorgestellt. Eine darauf aufbauende, eingehendere Betrachtung dezidierter, für den Kontext der Arbeit relevanter Derivate, ist Abschnitt 2.3 zu entnehmen.

Endliche Automaten mit Eingabe und Ausgabe

Nach der in der Automatentheorie *(Hopcroft u. a. 2002)* üblichen Mengen-Funktionen-Darstellung lässt sich ein endlicher Automat mit Ein- und Ausgabe als ein 6-Tupel definieren *(Litz 2005, S. 199)*, in dem sich die obige prinzipielle Darstellungsform ereignisdiskreterer Systeme wiederfindet.

$$A(Z^d, X^d, Y^d, f, g, Z_0^d)$$

Dabei bezeichnet:

$Z^d = \{Z_1^d, \dots, Z_k^d\}$	die Menge aller Systemzustände (Zustandsmenge),
$X^d = \{x_1^d, \dots, x_i^d\}$	die Menge aller Eingangssignale (Eingabemenge)[9],
$Y^d = \{y_1^d, \dots, y_j^d\}$	die Menge aller Ausgangssignale (Ausgabemenge),
$f\colon Z^d \times X^d \to Z^d$	die Zustandsübergangsfunktion,
$g\colon Z^d \times X^d \to Y^d$	die Ausgabefunktion und
$Z_0^d \in Z^d$	den Anfangszustand der Betrachtung.

Bei einer überschaubaren Anzahl von Systemzuständen bietet sich die intuitivere Darstellung als Automatengraph[10] an (siehe Abbildung 11.4). In diesem gerichteten Graph manifestieren sich die möglichen Systemzustände als sogenannte Knoten (Kreise). Die als Kanten bezeichneten Übergänge (Transitionen) zwischen den Zuständen werden als Pfeile dargestellt. Sie entsprechen einer graphischen Repräsentation der Zustandsübergangsfunktion *f*. Zusätzlich wird jede Kante mit dem Eingabe-/Ausgabepaar beschriftet, das aus der Zustandsübergangsfunktion und der Ausgabefunktion *g* hervorgeht. Die Darstellung des Anfangszustandes erfolgt durch einen unbeschrifteten, zum Knoten hinführenden Pfeil. Spezielle Arten von Zustandsübergangsfunktionen sind zum einen die sogenannte Selbsttransition, die dadurch charakterisiert ist, dass der Start- und Zielzustand derselbe sind. Des Weiteren die als ε-Transition bezeichnete

[9] Die Eingabe- und Ausgabemenge werden auch als Eingabe- bzw. Ausgabealphabet bezeichnet.

[10] Weitere in der Fachliteratur verwendete Bezeichnungen sind Statechart oder State-Transition-Diagramm.

Kante mit der ein Zustandsübergang ohne Eingangs- bzw. Ausgangssignale beschrieben wird.

Bezüglich der Eindeutigkeit der mathematischen Repräsentation werden zwei Automatenarten unterschieden. Der deterministische Automat ist dadurch gekennzeichnet, dass Zustandsübergänge eindeutig bestimmten Eingangssignalen zugeordnet sind ($Z_{k+1}^d(n+1) = f\left(Z_k^d(n),\, x_i^d(n)\right)$). Nichtdeterministische Automaten ermöglichen für ein bestimmtes Eingangssignal einen Übergang auf mehrere Zustände. Diese Form erlaubt es daher nicht, das Verhalten eines Systems eindeutig zu beschreiben. Determinismus ist aber gerade bei der Modellierung technischer Systeme und vor allem der Software zur Realisierung der Steuerungs- und Überwachungsfunktionalität Voraussetzung *(Sabbah 2000, S. 15)* und zur Systembeherrschung notwendig *(Lunze 2003, S. 332)*.

Ein weiteres Unterscheidungsmerkmal bezieht sich auf die Form der Ausgabefunktion g des Automaten. In Abhängigkeit davon, ob diese von der Eingabe $x_i^d(n)$ abhängt oder nicht werden Automaten in Mealy- bzw. Moore-Maschinen klassifiziert. Bei der oben beschriebenen Mealy-Notation hängt die Ausgabe $y_j^d(n)$ vom aktuellen Zustand $Z_k^d(n)$ und den Eingangssignalen $x_i^d(n)$ ab. Bei der Darstellung nach Moore wird die Ausgabe nur vom aktuellen Zustand bestimmt. Sie wird daher in der graphischen Darstellung direkt dem entsprechenden Zustand zugeordnet (siehe Abbildung 11.4). Als erweitere Form existiert der sogenannte Harel-Automat, der die beiden Konzepte kombiniert. Ausgangsereignisse können demnach den Transitionen und den Zuständen zugeordnet werden.

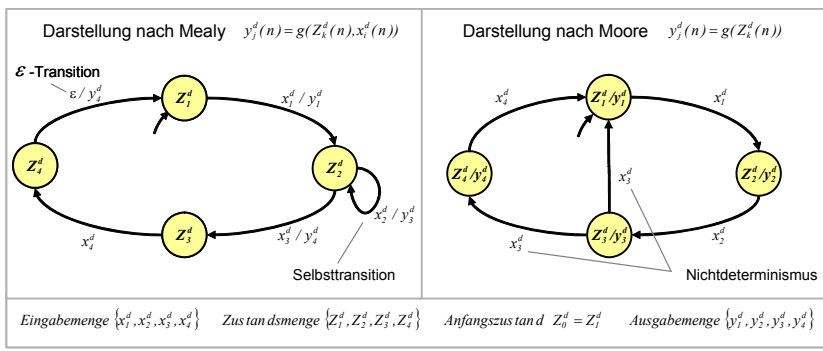
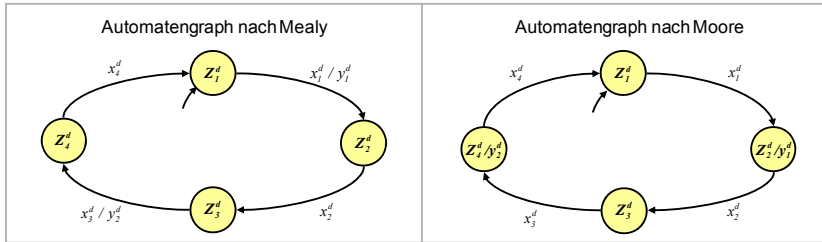


Abbildung 11.4: Automatengraph als Mealy- und Moore-Maschine

Neben dem Automatengraphen hat sich insbesondere für die Behandlung komplexerer Systeme, aber auch zur formalen Analyse sowie als Basis zur Simulation, die Automatentabelle als adäquat erwiesen. Bei dieser Darstellungsform werden die erreichbaren Folgezustände in Bezug auf einen gegebenen Momentanzustand und die aktuellen Eingangssignale anhand einer Tabelle abgebildet. Je nachdem ob es sich um die Ab-

bildung eines Mealy- oder Moore-Automaten handelt, erfolgt die Zuordnung der Ausgabegrößen zu den Folgezuständen bzw. zum momentanen Zustand. In Abbildung 11.5 wird die in Abschnitt 2.2.4 beschriebene hydraulische Positionierachse aus Sicht der Ansteuerung in den entsprechenden Notationsformen dargestellt.

Ob ein Automat vollständig definiert ist lässt sich anhand der Automatentabelle besonders gut bestimmen. Dies ist dann der Fall, wenn in jedem Zustand auf jedes Eingabesignal reagiert werden kann. Ist für einen Zustand kein Folgezustand definiert (leere Zeile), so kann der Zustand nicht mehr verlassen werden. Wenn dagegen in einem Tabellenfeld mehrere Folgezustände auftreten, ist der Automat als nicht deterministisch zu betrachten.



x_1^d = Steuerbefehl: „Achse ausfahren“
x_2^d = Sensorsignal: „Achse ausgefahren“
x_3^d = Steuerbefehl: „Achse einfahren“
x_4^d = Sensorsignal: „Achse eingefahren“

Z_1^d = Zustand: „eingefahren“
Z_2^d = Zustand: „fährt aus“
Z_3^d = „Zustand ausgefahren“
Z_4^d = „Zustand: „fährt ein“

y_1^d = „Magnetventil einschalten“
y_2^d = „Magnetventil ausschalten“

Momentan-Zustand Z_k^d	Folgezustand Z_{k+1}^d, momentaner Ausgang y_j^d bei momentaner Eingabe x_i^d			
	x_1^d	x_2^d	x_3^d	x_4^d
Z_1^d	Z_2^d / y_1^d			
Z_2^d		Z_3^d		
Z_3^d			Z_4^d / y_2^d	
Z_4^d				Z_1^d

Automatentabelle nach Mealy

Momentan-zustand Z_k^d, Ausgang y_j^d	Folgezustand Z_{k+1}^d bei momentaner Eingabe x_i^d			
	x_1^d	x_2^d	x_3^d	x_4^d
Z_1^d	Z_2^d			
Z_2^d / y_1^d		Z_3^d		
Z_3^d			Z_4^d	
Z_4^d / y_2^d				Z_1^d

Automatentabelle nach Moore

Abbildung 11.5: Automatengraph und Automatentabelle

Petri-Netze

Die Beschreibung größerer Systeme mit Automaten wird aufgrund der vielen Zustände relativ schnell sehr komplex. Vor allem bei der Beschreibung nebenläufiger Prozesse wird die Anzahl der möglichen Zustände dadurch exponentiell erhöht, dass die unabhängig voneinander arbeitenden Systemteile in unterschiedlicher Reihenfolge neue Zustände erreichen können und jede mögliche Reihenfolge zu einem anderen Gesamtzustand führt. Durch die Darstellung ereignisdiskreter Systeme in Form von Petri-Netzen wird die Modellkomplexität reduziert. Dies wird realisiert, indem parallel ablaufende Änderungen im System auch im entsprechenden Modell abgebildet werden.

Petri-Netze sind zwar bezüglich ihrer Mächtigkeit zur Abbildung des Systemverhaltens mit den Automaten vergleichbar, erlauben aber aufgrund zusätzlicher Modellelemente eine kompaktere und damit komfortablere Beschreibung.

Formal lässt sich ein Petri-Netz nach *(Abel 1990)* als 6-Tupel der Form

$$PN(S, T, F, K, W, M_0^d)$$

definieren. Dabei bezeichnet:

$S = \{S_1, \ldots, S_n\}$	die Menge aller Stellen,
$T = \{T_1, \ldots, T_k\}$	die Menge aller Transitionen,
$K \subseteq T \times S \cup S \times T$	die Menge aller Kanten (Flussrelation),
$C: S \to \mathbb{N} \setminus \{0\}$	die Abbildung der Kapazität einer Stelle,
$W: K \to \mathbb{N} \setminus \{0\}$	die Abbildung des Gewichts einer Kante und
$M_0^d: S \to \mathbb{Z}$	die Anfangsmarkierung.

Eine intuitive graphische Darstellung erfolgt, ähnlich den Automaten, als ein gerichteter Graph (siehe Abbildung 11.6). Die Knoten des Graphen werden durch die Stellen (Kreise) und Transitionen (Rechtecke) dargestellt. Aus der abwechselnden Verschaltung von Stellen und Transitionen mittels Kanten resultiert die Struktur und Flussrichtung des Graphen. Jede Stelle wird mit einer Kapazität beschriftet, die angibt wie viele Marken diese aufnehmen kann. Ebenso werden Kanten mit einer Gewichtung versehen. Nicht beschriftete Stellen und Kanten haben vereinbarungsgemäß die Kapazität C=1 bzw. das Gewicht W=1.

Trotz der Ähnlichkeit mit den Automaten werden die Systemzustände aber auch die Übergänge im Petri-Netz vollkommen anders dargestellt. Die Knoten repräsentieren nicht die möglichen Zustände. Diese entsprechen vielmehr der Verteilung der Marken auf die Stellen des Netzes zu einem bestimmten Zeitpunkt. Während Stellen eine rein passive Funktion haben, sind die Transitionen als aktive Elemente zu betrachten und verursachen durch sogenannte Schaltvorgänge den Markenfluss (W Marken fließen von einer der Transition vorgelagerten zur nachgelagerten Stelle) im Netz. Wann dies geschieht – und damit eine Zustandsänderung ausgelöst wird – bestimmt die Schaltregel, welche besagt, dass alle der Transition vorgelagerten Stellen markiert und alle nachgelagerten Stellen frei sein müssen. Transitionen korrespondieren folglich mit den Zustandsübergängen bei den Automaten.

Bei Petri-Netzen wird eine Vielzahl von Netztypen unterschieden. Die Klassifizierung beruht im Wesentlichen auf der Kapazität der Stellen, dem Gewicht der einzelnen Kanten, der Unterscheidung von Markenarten und teilweise auch adaptierten Schaltregeln. Bezüglich näherer Informationen hierzu sei auf die entsprechende Fachliteratur *(z. B. Pickhardt 2000; Niebert 2003a; Niebert 2003b)* verwiesen. Die im Folgenden betrachtete Klasse der Bedingungs-/Ereignisnetze (B/E-Netz) ist durch die Gewichtung und Kapazität W=C=1 gekennzeichnet. Sie stellt aufgrund der formalen Nähe zu

den in der Domäne der Automatisierungstechnik verbreiteten Fachsprachen die für die Arbeit relevanteste Modellform dar.

Spezielle Strukturen von B/E-Netzen sind die sogenannte Zustandsmaschine und der Synchronisationsgraph. Die erstgenannte Netzform zeichnet sich dadurch aus, dass jede Transition höchstens eine vorhergehende und eine nachfolgende Stelle hat. In Zustandsmaschinen ist also immer genau ein Platz markiert, wodurch ein derartiges Netz einem endlichen Automaten am nächsten kommt. Das damit einhergehende Problem, dass Nebenläufigkeiten nicht abbildbar sind, löst der Synchronisationsgraph. Bei dieser Netzart verfügt jede Stelle über maximal eine vor- und nachrangige Transition. Indem Marken aufgeteilt (Eröffnung) bzw. zusammengeführt werden (Synchronisation), lassen sich die parallelen Abläufe anschaulich darstellen. Abbildung 11.6 zeigt die genannten Netzstrukturen anhand einzelner Abläufe an einer Werkzeugmaschine.

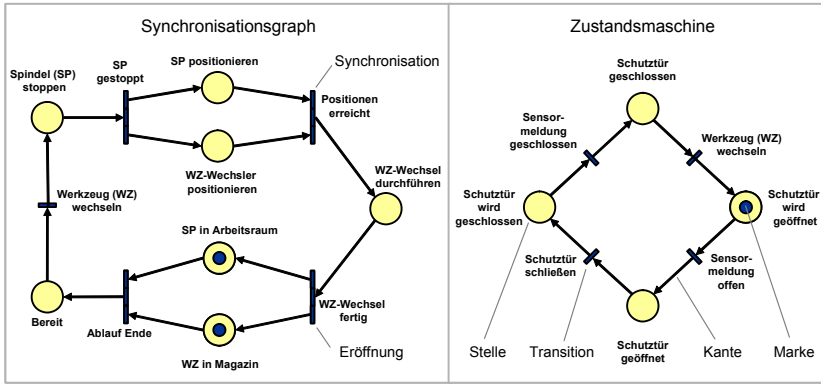


Abbildung 11.6: Petri-Netze der Klasse B/E-Netz

Petri-Netze haben die Fähigkeit zur kompakten Repräsentation diskreter Systeme und bauen auf einer präzise formulierten theoretischen Grundlage auf, die zahlreiche Analyseverfahren bereitstellt. An dieser Stelle sei nur auf die Erreichbarkeitsanalyse hingewiesen, bei der die Frage beantwortet wird, ob aus einer gegebenen Anfangsmarkierung des Netzes eine bestimmte Zielmarkierung und damit ein bestimmter Zustand erreicht werden kann.

Die formale Definition eines Petri-Netzes zeigt, dass diese im Gegensatz zu den Automaten über keine expliziten Ein- und Ausgangsgrößen verfügen. Die generelle Form dieses Beschreibungsmittels und auch die zugrunde liegende Theorie schließt derartige Größen nicht ein. Daher ist der Informationsfluss zu anderen Systemteilen nicht allgemein darstellbar. Aufgrund dieser Einschränkung wurden speziell für die Klasse der B/E-Netze entsprechende Erweiterungen entwickelt. Die sogenannten interpretierten Petri-Netze weisen den Stellen und Transitionen in Sachen Informationsfluss (Eingaben/Ausgaben) eine eindeutige Semantik zu. Eine für das Umfeld der Arbeit besonders

geeignete Beschreibungsform, das Signalinterpretierte Petri-Netz (SIPN), wird in Abschnitt 2.3.3 genauer beschrieben.

Boolesche Algebra

Diese Methode zur Beschreibung des Verhaltens eignet sich für Systeme, die rein binäre Größen verarbeiten. Eingangssignale und Zustandsgrößen werden gemäß den Gesetzen der Booleschen Algebra miteinander verknüpft und in entsprechende Ausgangsgrößen umgeformt. Die Darstellung kann zum einen mit der hierfür bekannten mathematischen Verknüpfungsnotation erfolgen (siehe Abbildung 11.7). Speziell für die Informationsverarbeitung wurden aber auch graphische Notationsformen entwickelt. So sind beispielsweise zwei der international genormten graphischen Programmiersprachen für die Entwicklung Speicherprogrammierbarer Steuerungen *(DIN EN 61131-3)* dieser Beschreibungsmethodik zuzuordnen (siehe Abschnitt 2.3.4).

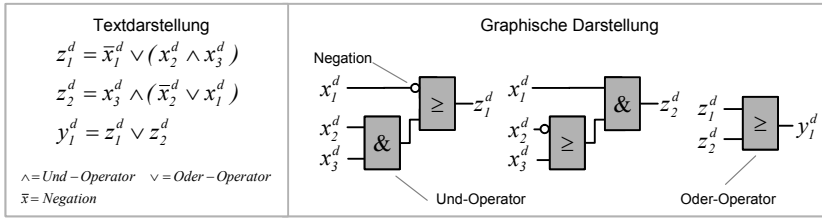


Abbildung 11.7: Systemverhalten mit Boolescher Algebra

11.2.3 Hybride Systeme

Die Betrachtung eines Systems als hybrid ist dann notwendig, wenn kontinuierliche und ereignisdiskrete Elemente kombiniert auftreten, ohne dass durch Abstraktion der eine oder andere Teil eliminierbar ist und zu einer einheitlichen Modellform übergegangen werden kann. Das Modell muss daher in der Lage sein, die Verarbeitung sowohl ereignisdiskreter wie auch kontinuierlicher Signale abzubilden. Um dies zu realisieren ist es notwendig, beide Beschreibungsformen miteinander zu kombinieren. Dies erfordert jedoch die Umwandlung ereignisdiskreter Signale in kontinuierliche Signale sowie die Generierung von Ereignissen aus den kontinuierlichen Größen. Dem entsprechend sind Schnittstellen zwischen kontinuierlichen und ereignisdiskreten Systemteilen notwendig. Diese werden als Injektor *I* bzw. Quantisierer *Q* bezeichnet. Abbildung 11.8 zeigt das hybride Modell einer hydraulischen Positionierachse[11]. Der Injektor generiert in Abhängigkeit davon, ob der Zylinder ein- oder ausfährt, eine kontinuierliche Größe (wirksame Kolbenfläche), die zur Berechnung der Achsposition herangezogen wird. Durch den Quantisierer hingegen werden in Abhängigkeit von der aktu-

[11] Diese Form der Spezifikation hybrider Systeme wird auch als Netz-Zustandsraum-Modell bezeichnet.

ellen Position die Ereignisse generiert, die dem Erreichen der Endlagen (Sensoren) entsprechen.

Obwohl der Reifegrad der formalen Beschreibung hybrider noch nicht mit dem ereignisdiskreter und insbesondere kontinuierlicher Systeme konkurrieren kann *(Litz 2005, S. 408)*, haben sich dennoch einige Formalismen auf der Basis von Automaten und Petri-Netzen in der Forschung etabliert. Diese basieren auf dem oben dargelegten Prinzip von Quantisierer und Injektor, integrieren aber diese (und teilweise auch den gesamten kontinuierlichen Systemanteil) meist direkt in die ereignisdiskrete Beschreibung. Umgekehrt existieren auch Methoden um ereignisdiskrete Systemanteile in kontinuierliche Beschreibungsformen aufzunehmen.

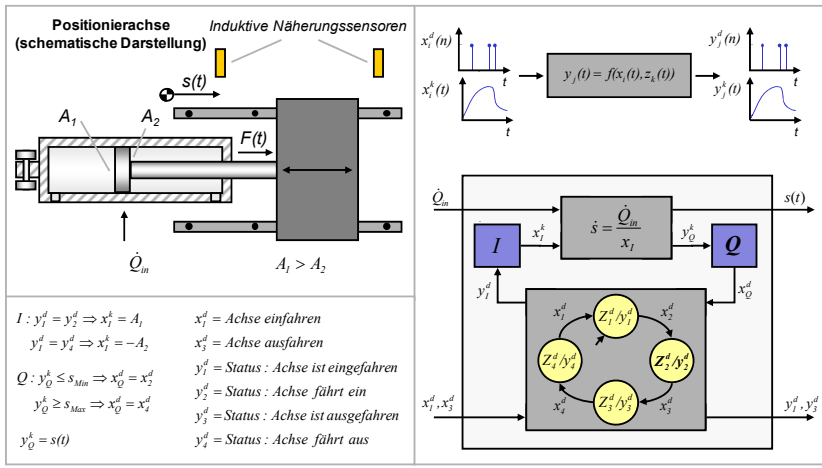


Abbildung 11.8: Basiskonzept der hybriden Systembeschreibung

Hybride und zeitbewertete Automaten

Hybride Automaten sind dadurch gekennzeichnet, dass jeder ereignisdiskrete Zustand $Z_k^d(n)$ durch genau eine kontinuierliche Zustandsbeschreibung $Z_k^k(t)$ verfeinert wird. Der kontinuierliche Zustand Z_k^k gilt dann genau so lange, wie sich der Automat im diskreten Zustand Z_k^d aufhält. Aufgrund dieses Zusammenhangs resultiert eine hybride Zustandsdarstellung:

$$Z_k = \begin{bmatrix} Z_k^d(n) \\ Z_k^k(t) \end{bmatrix}$$

Des Weiteren wird zur Komplexitätsreduktion bei trotzdem ausreichender Modellierungsmächtigkeit festgelegt, dass die Verarbeitung kontinuierlicher Ein- und Ausgangssignale ausschließlich der kontinuierlichen Zustandsbeschreibung $Z_k^k(t)$ zugeordnet wird. Ebenso gilt der Bezug der diskreten Ein- und Ausgangsgrößen zu den Zustandsübergängen des Automaten. Aufgrund dieser klaren Trennung ist es möglich,

für die mathematische Beschreibung der kontinuierlichen Systemeigenschaften eine Zustandsraumdarstellung zu verwenden. Die bei dieser Art der Systembeschreibung noch fehlende Verkoppelung von ereignisdiskretem und kontinuierlichem Modellteil erfolgt über die Transitionsstruktur des hybriden Automaten. Diese wird durch folgende Regeln definiert:

- Eine Transition findet nur statt, wenn die hierfür geltende Transitionsbedingung erfüllt ist. Diese ist eine logische Funktion der ereignisdiskreten und kontinuierlichen Eingangssignale und des kontinuierlichen Zustandes.
- Der Zustandsübergang löst (falls notwendig) eine ereignisdiskrete Ausgabe aus. Gleichzeitig definiert er die Randbedingungen der kontinuierlichen Beschreibung $Z_k^k(t)$ des Zielzustandes.

Zur Veranschaulichung dieser Zusammenhänge zeigt Abbildung 11.9 einen hybriden Automaten der beschriebenen Positionierachse. Eine detaillierte theoretische Betrachtung findet sich beispielsweise in *(Henzinger 1996)*.

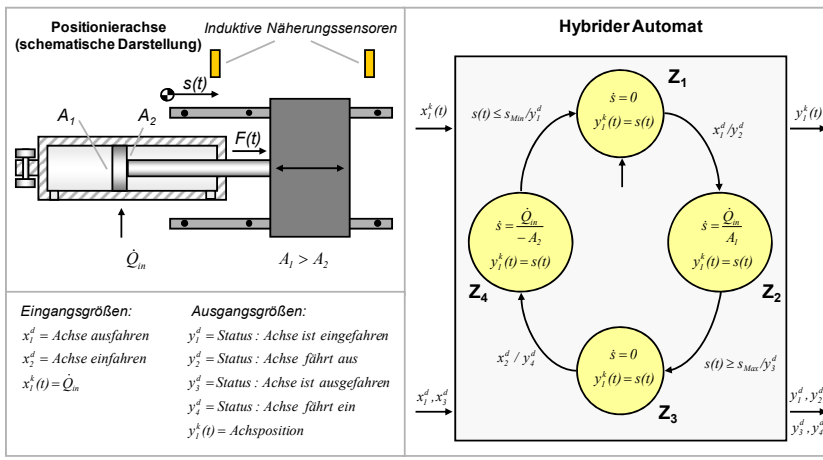


Abbildung 11.9: Hybrider Automat

Zeitbewertete Automaten stellen insofern eine Erweiterung dar, als dass die Zeit direkt in die Transitionsbedingung mit eingeht und diese einschränkt. Durch die Angabe eines oberen und unteren Schwellwertes wird ein Zeitraum festgelegt, innerhalb dessen die Transition (bei Erfüllung aller anderen Randbedingungen) schalten kann. Werden keine weiteren Restriktionen für den Zustandsübergang angegeben, so gilt das definierte Zeitfenster als alleinige Bedingung. Diese Erweiterung der Modellierungsmächtigkeit, die auch bei endlichen Automaten angewandt werden kann, ist bei der praktischen Umsetzung von Steueralgorithmen weit verbreitet. Der Ablauf einer bestimmten Zeitspanne ersetzt oft nur schwer oder mit hohem Aufwand erfassbare Sensorwerte

oder dient zum Überwachen von Bewegungen oder anderen Abläufen von festgelegter Dauer. Bezüglich einer detaillierten Betrachtung der Zeit bei der Modellierung ereignisdiskreter und hybrider Systeme wird auf entsprechende Literaturquellen *(z. B. Lee & Varaiya 2003)* verwiesen.

Hybride Petri-Netze

Analog zu den endlichen Automaten lassen sich auch mit Petri-Netzen hybride Systembeschreibungen aufbauen. Hierzu finden sich in der Fachliteratur eine Vielzahl von Varianten. Dies hat zum einen den Hintergrund, dass wegen der vielen Möglichkeiten zur Definition der Semantik von Marken und Transitionen ein breites Instrumentarium zur Verfügung steht. Zum anderen erlaubt es eine zusätzliche Erweiterung, auch kontinuierliches Verhalten im Petri-Netz darzustellen. Diese erfolgt, indem der Markeninhalt einer Stelle sowie die Markenflussrate (Gewichtung) als reelle Zahlen definiert werden. Zur Veranschaulichung der entsprechenden formalen Hintergründe wird im Folgenden exemplarisch das sogenannte hybride dynamische Netz (HDN) näher betrachtet *(Engell u. a. 2002, S. 15-36)*. Einen guten Überblick bezüglich der weiteren bislang entwickelten Netztypen bietet *(Chouika u. a. 2000)*.

Eine wesentliche Charakteristik eines HDN ist die Beschreibung der kontinuierlichen Eingangs-, Zustands- und Ausgangsgrößen mittels doppelt umrandeter Stellen (siehe Abbildung 11.10).

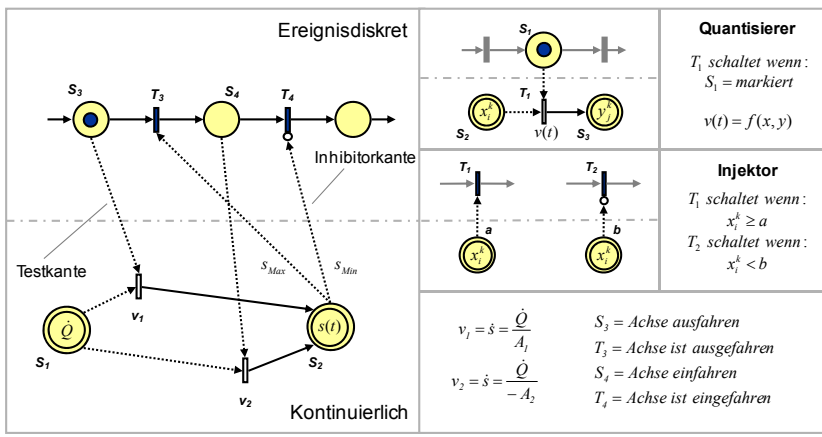


Abbildung 11.10: Hybrides Dynamisches Netz

Die Beziehung zwischen den einzelnen Größen wird durch Transitionen mit kontinuierlicher Markenflussrate dargestellt und beschreibt die Zusammenhänge zwischen den unterschiedlichen Variablen analog zur Zustandsraumdarstellung als Differenzialgleichung erster Ordnung. Die Koppelung zum in üblicher Petri-Netz-Notation dargestellten diskreten Systemanteil erfolgt mittels sogenannter Test- und Inhibitorkanten. Diese

wirken als einschränkende Randbedingungen für das Schalten der Transitionen. Eine Testkante verbindet eine ereignisdiskrete Stelle und eine kontinuierliche Transition und implementiert somit die Funktion des Injektor. Die Markierung der Stelle ist Voraussetzung für die Aktivierung des Markenflusses über die Transition. Umgekehrt übernimmt eine Inhibitorkante die Funktion des Quantisierers. Das Über- bzw. Unterschreiten eines bestimmten Grenzwertes ist für das Schalten der Transition im ereignisdiskreten Systemteil notwendig.

Auch für kontinuierliche Systembeschreibungen wurden Erweiterungen entwickelt, um diskretes Verhalten abbilden zu können. Begründet liegt dies in der Tatsache, dass viele Modelle technischer Systeme Unstetigkeiten enthalten, die dazu führen, dass sich die beschreibenden Gleichungen unstetig ändern. Die wesentlichen Ansätze hierzu beruhen auf der Einführung von ergänzenden Sprachkonstrukten bei der Erstellung der Modelle und werden im Folgenden dargestellt.

Unstetige Modellgleichungen

Unstetige Modellgleichungen beschreiben ein System, das in Abhängigkeit vom Wertebereich der Eingangsgrößen ein unterschiedliches Systemverhalten aufweist. Entsprechend müssen die mathematischen Zusammenhänge durch alternative Gleichungssysteme abgebildet werden. Die Herausforderungen, die sich bei dieser einfachen und nahe liegenden Modellbildungsmöglichkeit ergeben, sind insbesondere im Konvergenzverhalten bei der Lösung der Gleichungssysteme zu suchen. Die eingesetzten Solver müssen über geeignete Indikatorfunktionen[12] verfügen, um den exakten Zeitpunkt zur Wahl der richtigen Systemzusammenhänge zu ermitteln. Dies erfolgt dadurch, dass die numerische Integration nach jedem Schritt angehalten wird, anhand der Iteratorfunktion der richtige Bereich ermittelt und anschließend die Integration neu gestartet wird. Abbildung 11.11 zeigt die Modellierung mit unstetigen Gleichungen anhand des einfachen Beispiels eines Begrenzers. Für detailliertere Informationen bezüglich der Indikatorfunktionen und der numerischen Herausforderungen bei der Problemlösung wird auf *(Otter 1995, S. 45-51)* verwiesen.

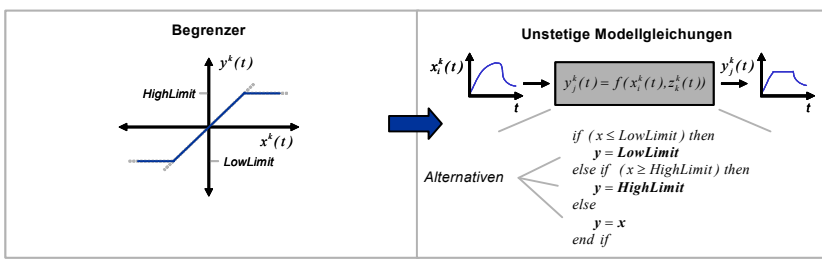


Abbildung 11.11: Unstetige Modellgleichungen

[12] Eine Indikatorfunktion ermittelt den Umschaltpunkt zwischen verschiedenen Bereichen.

Instant-Modellgleichungen

Mit den obigen Modellgleichungen sind nicht alle möglichen, unstetigen Effekte beschreibbar. Es werden auch Mechanismen benötigt, die plötzliche Änderungen innerhalb des Modells erfassen und mehr als eine Bereichsumschaltung erfordern. Dies kann durch sogenannte Instant-Modellgleichungen erfolgen. Dabei handelt es sich um ergänzende Gleichungen, die nur dann bei der Lösung des Gleichungssystems berücksichtigt werden, wenn bestimmte Randbedingungen[13] erfüllt sind. Abbildung 11.12 zeigt die Anwendung dieses Beschreibungselements anhand einer einfachen Hysteresie. Die innerhalb des Bedingungsblocks stehenden Gleichungen werden bei der numerischen Lösung nur ausgewertet, wenn die angegebene Prämisse erfüllt (Schalten von false auf true) wird. Ansonsten bleibt die Ausgabegröße unverändert.

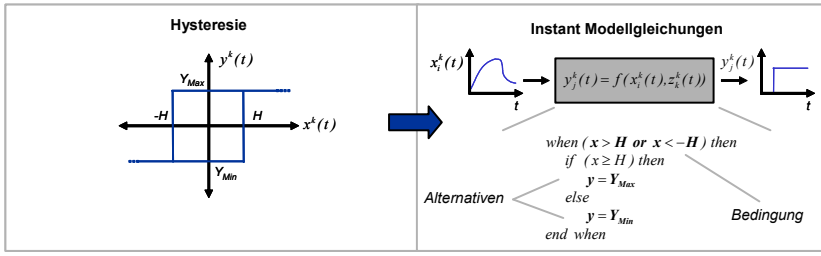


Abbildung 11.12: Instant Modellgleichungen

11.3 Automatisierungstechnik in Fertigungssystemen

11.3.1 Steuerungsarchitekturen

Auf der Aktor-/Sensorebene, der Steuerungsebene und der Zellenebene eines automatisierten Fertigungssystems werden zentrale Steuerungen, zentrale Steuerungen mit dezentraler Ein- und Ausgabe, hierarchisch organisierte Steuerungsverbünde, hierarchisch organisierte Verbünde mit direkter Synchronisation und kooperative Steuerungssysteme als grundsätzliche Architekturkonzepte unterschieden.

Zentrale Steuerungsarchitekturen mit dezentraler Ein- und Ausgabe (siehe Abbildung 11.13) sind insbesondere auf der Steuerungsebene und der Aktor-/Sensorebene stark verbreitet. Dabei wird die Signalverarbeitung in der Regel von einer einzigen Steuerung übernommen. Die Verknüpfung mit den Sensoren und Aktoren erfolgt über ein entsprechend den Anforderungen gestaltetes Bussystem und örtlich verteilten Ein- und Ausgangsmodulen.

[13] Die Erfüllung der Randbedingungen wird auch als Ereignis bezeichnet.

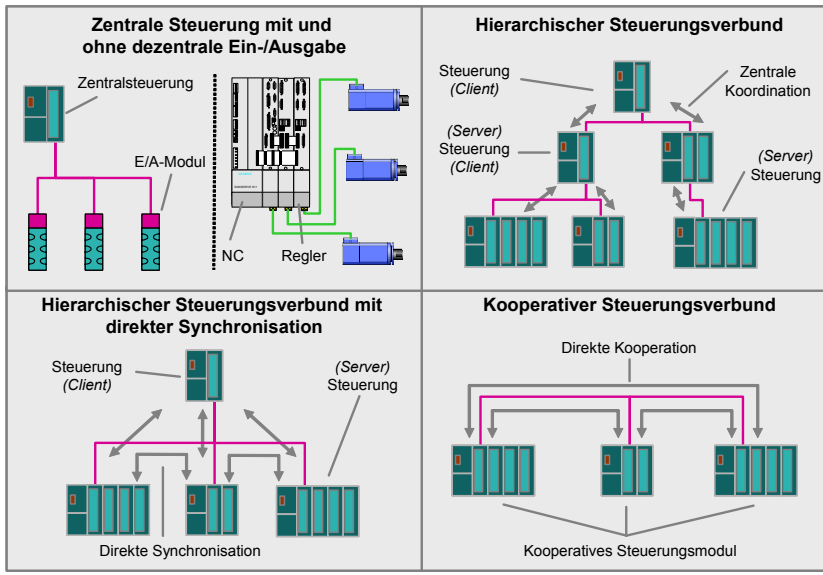


Abbildung 11.13: Steuerungsarchitekturen in Fertigungssystemen

Der wesentliche Vorteil gegenüber zentralen Steuerungsarchitekturen, bei denen sich alle erforderlichen Komponenten im Schaltschrank befinden und über ein integriertes Kommunikationssystem verbunden sind, ergibt sich durch die Einsparung von Kabelkosten, reduzierten Aufwendungen hinsichtlich der Montage und eine vereinfachte Elektrokonstruktion. Ein typisches Beispiel für eine derartige Architektur stellt die PLC-Steuerung einer Werkzeugmaschine dar. Diese befindet sich in der Regel in einem Schaltschrank, während die Module zum Anschluss der Sensoren und Aktoren in der Maschine verteilt sind. Bezüglich zentraler Steuerungsarchitekturen sei das numerische Steuerungs- und Regelsystem eines Bearbeitungszentrums als exemplarische Umsetzung genannt, bei der sich die NC-Steuerung wie auch die Achsregelmodule im Schaltschrank befinden.

Hierarchische Steuerungsverbünde werden beispielsweise zur Koordination komplexer Anlagen eingesetzt. Dabei übernimmt eine Steuerung auf einer übergeordneten Ebene (Zellenebene) die zentrale Koordination einzelner Anlagenteile. Diese verfügen wiederum über eine eigene Steuerung, welche das jeweilige Subsystem bedient. Die Vorteile einer derartigen Architektur sind hauptsächlich in den geringen Abhängigkeiten zwischen den einzelnen Subsystemen und der Reduktion der Gesamtkomplexität begründet. Schwächen hat das Konzept bei der Umsetzung zeitkritischer Koordinations- bzw. Synchronisationsfunktionen, da die hierzu notwendigen Signale über die übergeordnete Steuerung weitergeleitet werden müssen.

Durch die Einführung direkter Synchronisationsfunktionalität zwischen den Steuerungen kann dieser Nachteil abgemildert werden. Zeitkritische Signale werden direkt zwischen den jeweiligen Subsystemen ausgetauscht. Als Beispiel für solche Funktionalitäten seien elektronische Kurvenscheibensteuerungen genannt. Bei derartigen Systemen wird die Funktionalität einer Kurvenscheibe nachgeahmt, indem eine Gesamtbewegung aus mehreren unabhängigen Teilbewegungen zusammengesetzt wird. Die Koordination der Einzelbewegungen erfolgt durch die direkte Synchronisation der Antriebe. Dazu gibt ein Antrieb (Masterantrieb) die Führungsgröße vor. Die Bewegung der weiteren Antriebe wird realisiert, indem diese direkt mit dem Masterantrieb kommunizieren und in Abhängigkeit seines Bewegungszustandes ihr Bewegungsverhalten berechnen. Nachteile des beschriebenen Architekturkonzeptes sind eine steigende Gesamtkomplexität durch die zusätzlichen Schnittstellen sowie eine Einschränkung der Wiederverwendbarkeit entsprechender Softwarebausteine.

Bei kooperativen Steuerungsverbünden wird auf eine übergeordnete Steuerung verzichtet. Die Steuerungsaufgaben einzelner Module werden durch direkte Kommunikation synchronisiert und koordiniert. Derartige Architekturen entstehen beispielsweise bei der Integration mittelkomplexer Anlagen unterschiedlicher Hersteller (z. B. Bearbeitungszentrum und automatisiertes Werkstück-Beladesystem), die jeweils über eine eigene Steuerung verfügen und miteinander kommunizieren, indem sie Signale über eine gemeinsame Schnittstelle austauschen.

11.3.2 Steuerungen in Fertigungssystemen

Die im Werkzeugmaschinenbau, aber auch in anderen Gebieten des Maschinenbaus, eingesetzten Steuerungen basieren aufgrund der hohen Flexibilität und Leistungsfähigkeit sowie der geringen Herstellkosten heute vorwiegend auf der Mikroprozessortechnik. Ihre Aufgaben reichen von der Verrichtung einfacher Schaltvorgänge über die Umsetzung von Handhabungs- und Transportaufgaben hin bis zur präzisen räumlichen Koordination mehrerer Bewegungsachsen. Aufgrund der mannigfaltigen Anforderungen an die zu erfüllenden Funktionen haben sich verschiedene Steuerungsarten entwickelt, die sich durch eine unterschiedliche Funktionsweise, Programmierung und Bedienung auszeichnen. Im Produktionsanlagenbau werden prinzipiell als wesentliche Systemtypen Speicherprogrammierbare Steuerungen, Numerische Steuerungen und Robotersteuerungen unterschieden.

Speicherprogrammierbare Steuerungen (SPS)

Speicherprogrammierbare Steuerungen werden in Fertigungssystemen vorwiegend zur Umsetzung von Nebenfunktionen eingesetzt. Diese umfassen unter anderem die Steuerung der Medienversorgung, des Werkzeug- und Werkstücktransports sowie sonstiger Peripheriefunktionen. Häufig erfolgt auch die Kommunikation zu übergeordneten Leitsystemen und zu Bedien- und Beobachtungssystemen über die Speicherprogrammierbare Steuerung. Die vorwiegend eingesetzten Systeme arbeiten zyklusorientiert,

wobei ein vom Hersteller fest eingespeichertes Betriebssystem diesen Zyklus kontrolliert. Aufgrund der bedingten Ausführung einzelner Programmteile weisen die Steuerungsprogramme unterschiedliche Laufzeiten auf. Daher ist die Umsetzung von Regelungen mit einer SPS nur eingeschränkt möglich. Neuere Systeme verfügen zwar zum Teil über entsprechende Funktionen. Mittlerweile gibt es beispielsweise Produkte, die geregelte Verfahrbewegungen umsetzen. Derartige Baugruppen verfügen aber über einen eigenen Prozessor, bei dem entsprechende Funktionen fest implementiert sind. Die SPS kann Positionier- und Verfahraufgaben lediglich über eine definierte Schnittstelle anfordern. Die eigentliche Funktion – die Regelung der Bewegung – ist daher nicht der Speicherprogrammierbaren Steuerung zuzuordnen.

Numerische Steuerungen (NC-Steuerungen)

Numerische Steuerungen werden dann verwendet, wenn mehrere Bewegungsachsen koordiniert zueinander bewegt werden müssen. Folglich werden sie vorwiegend in Werkzeugmaschinen eingesetzt. Ihre wesentliche Funktion ist die Ausführung der für die Punkt-, Strecken- und Bahnbewegung notwendigen Interpolations- und Regelungsaufgaben. Kennzeichen Numerischer Steuerungen ist das Vorhandensein von direkten oder indirekten Weg- bzw. Winkelmesssystemen an den Bewegungsachsen. Die aktuellen Achspositionen werden erfasst und an die Steuerung zum Zwecke der Regelung zurückgeführt. Die Regelung selbst ist typischerweise kaskadierend aus einem Stromregler, einem Geschwindigkeitsregler und einem Lageregler aufgebaut. Während die erstgenannten Regelkreise meist direkt auf einem eigenen Antriebsregelmodul integriert sind, kann der Lageregelkreis auch durch das Steuerungsmodul implementiert werden.

Zur Beschreibung der Steuerungsaufgabe dient ein mittels alphanumerischer Zeichen aufgebautes Steuerungsprogramm (NC-Programm). Der Befehlssatz ist durch entsprechende Normen *(DIN 66025-1; DIN66025-2)* vorgegeben. Da dieser die vielfältigen Aufgaben und Möglichkeiten moderner NC-Steuerungen nur unzureichend abdeckt, werden vermehrt herstellerspezifische Befehle eingesetzt. Ebenso ist darauf aufbauend der Trend zur Entwicklung spezieller Funktionen durch die Hersteller von Fertigungssystemen zu beobachten. Die Erstellung der Programme erfolgt im Rahmen der Arbeitsvorbereitung durch das Bedienpersonal. Dieses hat die Aufgabe, die Fertigungszeichnungen zu analysieren und die zur Bearbeitung notwendigen Anweisungen in das Programm zu integrieren. Aufgrund der zunehmenden Komplexität der zu fertigenden Werkstücke sowie dem hohen Aufwand zur Programmerstellung, kommen vermehrt Rechnerprogramme zum Einsatz, welche die NC-Anweisungen direkt aus einem dreidimensionalen Modell des Bauteils ableiten (CAD-CAM-Koppelung).

Die Interpretation der Steuerungsprogramme erfolgt durch ein eigenes Modul (NC-Interpreter). Dieses liest die Teileprogramme satzweise ein und analysiert die dort enthaltenen Weg- und Geschwindigkeitsangaben sowie zusätzliche technologische Anweisungen und Schaltbefehle. Geometrieanweisungen werden an den Interpolator wei-

tergeleitet, der die Lagesollwerte für die einzelnen Antriebe generiert. Schaltbefehle gehen an eine meist direkt in die NC-Steuerung integrierte SPS. Die für die gezielte Ausführung der Steuerungsprogramme notwendige Synchronisation von NC-Steuerung und SPS erfolgt über eine vordefinierte Schnittstelle, die häufig in Form eines gemeinsamen Speicherbereichs implementiert ist.

Industrieroboter-Steuerungen

Industrieroboter besitzen analog zu Werkzeugmaschinen numerisch gesteuerte Achsen. Die speziell auf Industrieroboter ausgerichteten RC-Steuerungen sind daher in ihrer Funktion weitestgehend mit NC-Steuerungen vergleichbar. Ein wesentlicher Unterschied besteht darin, dass die einzelnen Achsen eines Industrieroboters oft seriell angeordnet sind und kinematische Ketten bilden (z. B. Sechs-Achs-Knickarmroboter). Daher sind die Berechungsalgorithmen für die Bahnsteuerung wesentlich komplexer und rechenintensiver als bei NC-Steuerungen. Weiterhin verfügen RC-Steuerungen über spezielle Funktionen zur Teach-In Programmierung und erlauben es, binäre und serielle Ein- und Ausgabeschnittstellen direkt aus dem Anwenderprogramm heraus anzusprechen. Analog zu NC-Steuerungen existiert mit der Industrial Robot Language (IRL) eine genormte Programmiersprache. Diese hat jedoch den Nachteil, dass sie von den Herstellern der Systeme nicht oder nur teilweise unterstützt wird bzw. nur einen Teil des verfügbaren Sprachumfangs darstellt.

11.4 Ergänzende Sprechelemente der Domain Specific Language

11.4.1 Datentypen

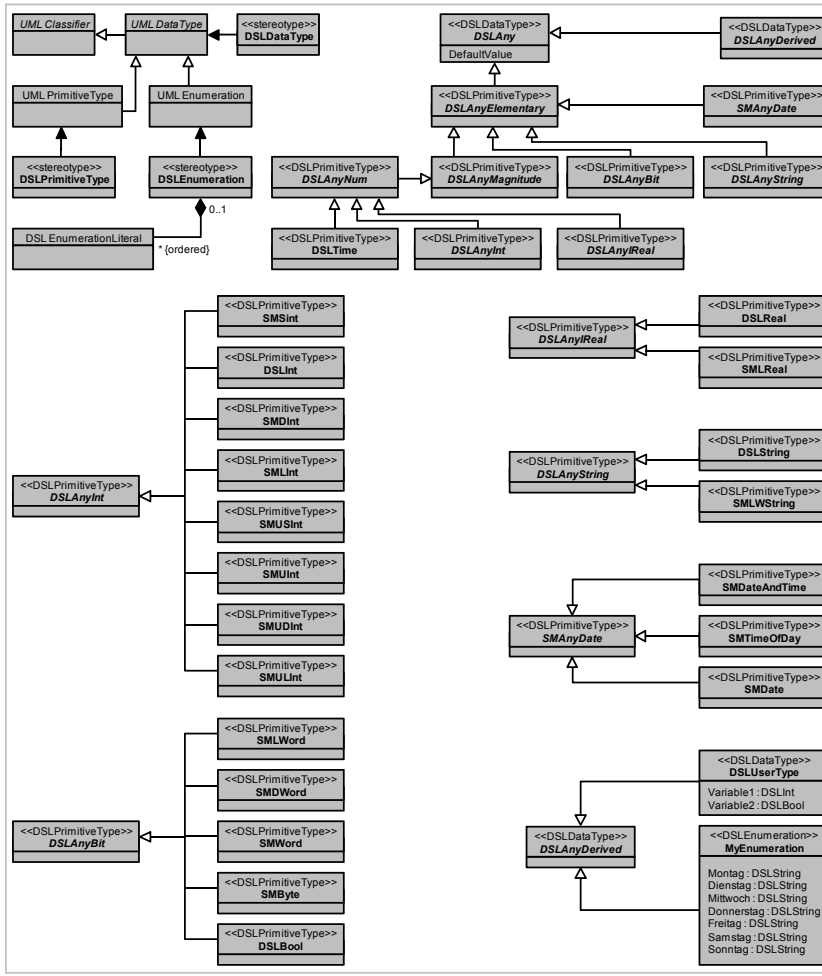


Abbildung 11.14: Datentypen der Domain Specific Language (DSL)

11.4.2 Operatoren und elementare Strukturbausteine

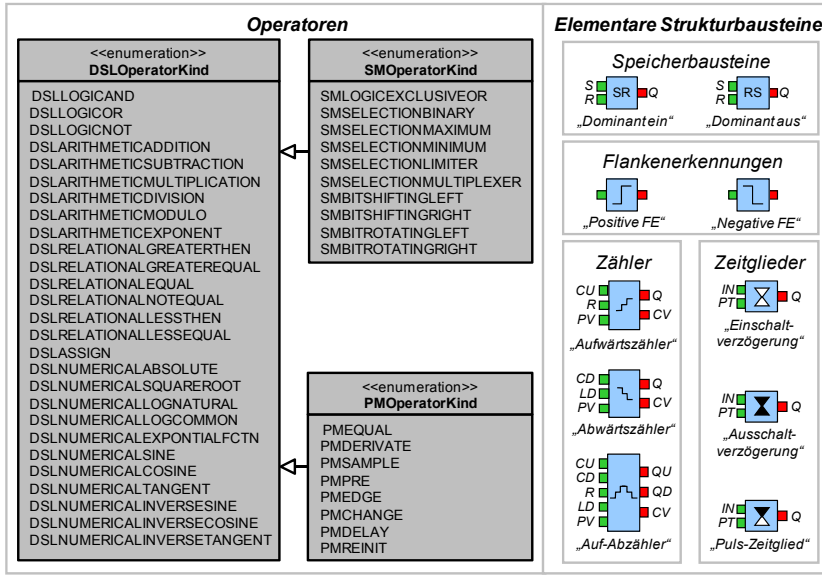


Abbildung 11.15: Operatoren und elementare Strukturbausteine

11.5 Transformationen präskriptiver Modelle

Beschreibung Modellelement	Interpretation	Umsetzung
Ausgehende synchrone Nachricht von einer Komponente des Steuerungsmodells; Nachricht ohne Attribute oder mit n Attributen eingehender und m Attributen ausgehender Kausalität.	Aktion zum Aufruf anderer Verhaltensbestandteile des Nachrichtenempfängers.	Hinzufügen einer Operation zu den entsprechenden Schnittstellen der kommunizierenden Ports; einfügen einer Aktion vom Typ „SMCallBehaviorAction“, die den Operationsaufruf realisiert; n +m Attribute als Parameter der Operation bzw. n+m Pins des Aktionsaufrufs mit entsprechender Kausalität.
Ausgehende asynchrone Nachricht von einer Komponente des Steuerungsmodells an eine Komponente des Steuerungsmodells; Nachricht ohne Attribute.	Schreibender Zugriff[14] auf eine Boolesche Variable des Nachrichtenempfängers.	Hinzufügen einer Operation zu den entsprechenden Schnittstellen der kommunizierenden Ports; einfügen einer Aktion vom Typ „SMWriteVariableValueAction“; einfügen einer entsprechenden Variable beim Empfänger (Variablenname = Name der Nachricht).
Ausgehende asynchrone Nachricht von einer Komponente des Steuerungsmodells an eine Komponente des Steuerungsmodells; Nachricht ohne Attribute.	Lesender Zugriff[15] auf eine Boolesche Variable des Nachrichtensenders.	Hinzufügen einer Operation zu den entsprechenden Schnittstellen der kommunizierenden Ports.
Ausgehende asynchrone Nachricht von einer Komponente des Steuerungsmodells an eine Komponente des Steuerungsmodells; Nachricht mit n Attributen ausgehender und eingehender Kausalität.	Zuweisen des Wertes der referenzierten Variablen des Senders an die referenzierten Variablen des Empfängers.	Hinzufügen von n Operationen zu den entsprechenden Schnittstellen der kommunizierenden Ports; einfügen von n Aktionen vom Typ „SMRead-VariableValueAction“ und „SMWriteVariableValueAction“ in eine neue Aktivität, die den Namen der Nachricht trägt.
Ausgehende asynchrone Nachricht von einer Komponente des Steuerungsmodells an eine Komponente des Steuerungsmodells; Nachricht mit n Attributen ausgehender Kausalität.	Lesender Zugriff auf n Variablen des Nachrichtensenders.	Hinzufügen von n Operationen zu den entsprechenden Schnittstellen der kommunizierenden Ports.
Ausgehende asynchrone Nachricht von einer Komponente des Steuerungsmodells an eine Komponente des Physikalischen Teilmodells oder eine andere Ressource; Nachricht ohne Attribute.	Schreibender Zugriff auf eine durch ein Signal referenzierte Boolesche Variable.	Hinzufügen eines Signals vom Typ „DSLBOOL“ zu den entsprechenden Schnittstellen der kommunizierenden Ports; einfügen einer Aktion vom Typ „SMWriteVariableValueAction“ sowie einer Booleschen Variable vom Typ „DSLPUBLIC“ (Variablenname = Name der Nachricht).
Ausgehende asynchrone Nachricht von einer Komponente des Steuerungsmodells an eine Komponente des Physikalischen Teilmodells oder eine andere Ressource; Nachricht mit n Attributen ausgehender Kausalität.	Schreibender Zugriff auf n durch Signale referenzierte Variablen.	Hinzufügen von n Signalen zu den entsprechenden Schnittstellen der kommunizierenden Ports; einfügen von n Aktionen vom Typ „SMWriteVariableValueAction“ in eine neue Aktivität, die den Namen der Nachricht trägt.
Ausgehende asynchrone Nachricht von einer Komponente des Physikalischen Teilmodells an eine Komponente des Physikalischen Teilmodells oder des Steuerungsmodells; Nachricht ohne Attribute.	Schreibender Zugriff auf eine durch ein Signal referenzierte Boolesche Variable.	Hinzufügen eines Signals vom Typ „DSLBOOL“ zu den entsprechenden Schnittstellen der kommunizierenden Ports, sowie einer Booleschen Variablen vom Typ „DSLPUBLIC“ (Variablenname = Name der Nachricht); die Kausalität des Signals wird als ausgehend definiert.
Ausgehende asynchrone Nachricht von einer Komponente des Physikalischen Teilmodells an eine Komponente des Physikalischen Teilmodells; Nachricht mit n Attributen ausgehender und l Attributen ohne Kausalität.	Schreibender Zugriff auf n/l durch Signale referenzierte Variablen.	Hinzufügen von n+l Signalen zu den entsprechenden Schnittstellen der kommunizierenden Ports; die Kausalität der Signale entspricht der Kausalität der Attribute der Nachricht.
Ausgehende asynchrone Nachricht von einer Komponente des Physikalischen Teilmodells an eine Komponente des Steuerungsmodells; Nachricht mit n Attributen ausgehender Kausalität.	Schreibender Zugriff auf n durch Signale referenzierte Variablen.	Hinzufügen von n Signalen zu den entsprechenden Schnittstellen der kommunizierenden Ports; die Kausalität der Signale entspricht der Kausalität der Attribute der Nachricht.

[14] Nur möglich, wenn die Boolesche Variable durch eine Markierung als Teil der empfangenden Lebenslinie deklariert wird.

[15] Nur möglich, wenn die Boolesche Variable durch eine Markierung als Teil der sendenden Lebenslinie deklariert wird.

Beschreibung Modellelement	Interpretation	Umsetzung
Eingehende synchrone Nachricht an einer Komponente des Steuerungsmodells; Nachricht ohne Attribute oder mit n Attributen eingehender und m Attributen ausgehender Kausalität.	Aufruf von Verhaltensbestandteilen des Nachrichtenempfängers.	Verknüpfung mit einer Funktion bzw. der von dieser referenzierten Aktivitäten; hinzufügen von m Parameterknoten eingehender und n Parameterknoten ausgehender Kausalität zu den Aktivitäten; falls bereits eine indirekte Verknüpfung des Empfangsereignisses zu Aktivitäten besteht, erfolgt keine Transformation.
Eingehende asynchrone Nachricht an einer Komponente des Steuerungsmodells von einer Komponente des Steuerungsmodells; Nachricht ohne Attribute.	Lesender Zugriff[16] auf eine Boolesche Variable des Nachrichtenempfängers.	Lesender Zugriff auf die Variable des Nachrichtenempfängers im Rahmen der Spezifikation von Überwachungsbedingungen; die Übersetzung ist davon abhängig, ob die Trace der Lebenslinie in Teile eines Zustandsgraphen oder eine koordinierende Aktivität übersetzt werden soll.
Eingehende asynchrone Nachricht an einer Komponente des Steuerungsmodells von einer Komponente des Steuerungsmodells; Nachricht ohne Attribute.	Lesender Zugriff[17] auf eine Boolesche Variable des Nachrichtensenders.	Lesender Zugriff auf die Variable des Nachrichtensenders im Rahmen der Spezifikation von Überwachungsbedingungen; die Übersetzung ist davon abhängig, ob die Trace der Lebenslinie in Teile eines Zustandsgraphen oder eine koordinierende Aktivität übersetzt werden soll.
Eingehende asynchrone Nachricht an einer Komponente des Steuerungsmodells von einer Komponente des Steuerungsmodells; Nachricht mit n Attributen ausgehender und eingehender Kausalität.	Zuweisen des Wertes der referenzierten Variablen des Senders an die referenzierten Variablen des Empfängers.	Wird nicht transformiert, da das entsprechende Verhalten vollständig vom Nachrichtensender implementiert wird.
Eingehende asynchrone Nachricht an einer Komponente des Steuerungsmodells von einer Komponente des Steuerungsmodells; Nachricht mit n Attributen ausgehender Kausalität.	Lesender Zugriff auf n Variablen des Nachrichtensenders.	Lesender Zugriff auf n Variablen des Nachrichtensenders im Rahmen der Spezifikation von Überwachungsbedingungen; die Übersetzung ist davon abhängig, ob die Trace der Lebenslinie in Teile eines Zustandsgraphen oder eine koordinierende Aktivität übersetzt werden soll.
Eingehende asynchrone Nachricht an einer Komponente des Physikalischen Teilmodells; Nachricht ohne Attribute.	Lesender Zugriff auf eine durch ein Signal referenzierte Boolesche Variable .	Bei einer existierenden indirekten Verknüpfung des Empfangsereignisses zu Verhaltenselementen erfolgt keine Transformation; alternativ Einfügen einer Sektion, Zuweisung und einer Variablen die den Namen der Nachricht trägt; hinzufügen eines Signalknoten.
Eingehende asynchrone Nachricht an einer Komponente des Physikalischen Teilmodells; Nachricht mit n Attributen ausgehender und l Attributen ohne Kausalität[18].	Lesender Zugriff auf n/l durch Signale referenzierte Variablen.	Bei einer existierenden indirekten Verknüpfung des Empfangsereignisses zu Verhaltenselementen erfolgt keine Transformation; alternativ Einfügen einer Sektion, Zuweisung und einer Variablen die den Namen der Nachricht trägt; hinzufügen von n+l Signalknoten.
Eingehende asynchrone Nachricht an einer Komponente des Steuerungsmodells von einer Komponente des Physikalischen Teilmodells oder einer anderen Ressource; Nachricht ohne Attribute.	Lesender Zugriff auf eine durch ein Signal referenzierte Boolesche Variable.	Lesender Zugriff auf die Variable des Nachrichtenempfängers im Rahmen der Spezifikation von Überwachungsbedingungen; die Übersetzung ist davon abhängig, ob die Trace der Lebenslinie in Teile eines Zustandsgraphen oder eine koordinierende Aktivität übersetzt werden soll.
Eingehende asynchrone Nachricht an einer Komponente des Steuerungsmodells von einer Komponente des Physikalischen Teilmodells oder einer anderen Ressource; Nachricht mit n Attributen ausgehender Kausalität.	Lesender Zugriff auf n durch Signale referenzierte Variablen.	Lesender Zugriff auf die Variable des Nachrichtenempfängers im Rahmen der Spezifikation von Überwachungsbedingungen oder von Aktivitäten; die Übersetzung ist davon abhängig, ob die Trace der Lebenslinie in Teile eines Zustandsgraphen oder eine koordinierende Aktivität übersetzt werden soll.

[16] Nur möglich, wenn die Boolesche Variable durch eine Markierung als Teil der empfangenden Lebenslinie deklariert wird.

[17] Nur möglich, wenn die Boolesche Variable durch eine Markierung als Teil der sendenden Lebenslinie deklariert wird.

[18] Akausale Attribute sind nur bei Nachrichten zwischen Komponenten des Physikalischen Teilmodells erlaubt.

Beschreibung Modellelement	Interpretation	Umsetzung
Zustandsinvariante einer Komponente des Physikalischen Teilmodells.	Interpretation als Kontrollstruktur zur alternativen Steuerung des Verhaltens.	Hinzufügen einer Kontrollstruktur „PMControlStructure“ mit einem Operanden vom Typ „PMIF“ in das Modell; der Boolesche Ausdruck, der die Zustandsinvariante beschreibt, wird als zur prüfender Ausdruck zur Ausführungsteuerung des Operanden eingefügt.
Zustandsinvariante einer Komponente des Steuerungsmodells.	Interpretation als Überwachungsbedingung einer Transition eines Zustandsautomaten.	Hinzufügen einer Transition zwischen zwei Zuständen; der Boolesche Ausdruck, der die Zustandsinvariante beschreibt, wird als Randbedingung (Guard) für das Feuern der Transition eingefügt.
Zustandsinvariante einer Komponente des Steuerungsmodells.	Interpretation als Zustand eines Zustandsautomaten.	Hinzufügen eines Zustands in das Modell; der Boolesche Ausdruck, der die Zustandsinvariante beschreibt, wird als Randbedingung (Guard) für das Feuern derjenigen Transitionen eingefügt, die zum Zustand hinführen.
Zustandsinvariante einer Komponente des Steuerungsmodells.	Interpretation als Überwachungsbedingung einer Kontrollflusskante eines Aktivitätsdiagramms.	Hinzufügen einer Kontrollflusskante zwischen zwei Aktionen; der Boolesche Ausdruck, der die Zustandsinvariante beschreibt, wird als Randbedingung (Guard) für die Weitergabe der Kontrolltoken eingefügt.
Zustandsinvariante einer Komponente des Steuerungsmodells.	Interpretation als generische Aktion eines Aktivitätsdiagramms.	Hinzufügen einer generischen Aktion in ein Aktivitätsdiagramm; die generische Aktion erzeugt einen Booleschen Wert als Ausgang, der weitere Aktionen antriggern kann; damit ist semantisch eine Analogie zu einem Schritt der Ablaufsprache nach IEC 61131-3 gegeben; der Boolesche Ausdruck, der die Zustandsinvariante beschreibt, wird als Randbedingung (Guard) für die Weitergabe der Kontrolltoken an zur Aktion hinführenden Aktivitätskanten eingefügt.
Kombiniertes Fragment vom Typ „DSLALT“.	Interpretation als alternativer Teilgraph eines Zustandsautomaten.	Übersetzung der einzelnen Interaktionsoperanden als Teile eines Zustandsautomaten; die Kontrollbedingungen der Operanden werden als Randbedingungen (Guard) für das Feuern derjenigen Transitionen eingefügt, die vom vorhergehenden Zustand zu den jeweiligen Teilgraphen hinführen.
Kombiniertes Fragment vom Typ „DSLLOOP“.	Interpretation als wiederholt ablaufender Teilgraph in einem Aktivitätsdiagramm.	Übersetzung des Interaktionsoperanden als Teil eines Aktivitätsdiagramms; hinzufügen eines Vereinigungsknoten vor dem ersten und eines Entscheidungsknoten nach dem letzten Aktionsknoten der Teilaktivität; einfügen einer rückführenden Kontrollflusskante vom Entscheidungs- zum Vereinigungsknoten; die Kontrollbedingung des Operanden fungiert als Randbedingung für die Weitergabe von Kontrolltoken.
Kombiniertes Fragment vom Typ „DSLPAR“.	Interpretation als parallel ablaufender Teilgraph in einem Aktivitätsdiagramm.	Übersetzung des Interaktionsoperanden als Teil eines Aktivitätsdiagramms; hinzufügen eines Parallelisierungsknoten vor den ersten und eines Synchronisierungsknoten nach den letzten Aktionsknoten der Teilaktivitäten; einfügen von Kontrollflusskanten zwischen dem Parallelisierungsknoten und den ersten Aktionsknoten; die Kontrollbedingungen der Operanden fungieren als Randbedingungen für die Weitergabe von Token.
Ausführungsspezifikation einer Komponente des Physikalischen Teilmodells.	Interpretation als Aufruf von Gleichungen oder Zuweisungen.	Keine Transformation, da üblicherweise eine direkte oder indirekte (über eine Funktion) Verknüpfung zu einer oder mehreren Sektionen, Kontrollstrukturen Gleichungen oder Zuweisungen im Modell besteht.
Ausführungsspezifikation einer Komponente des Steuerungsmodells.	Interpretation als Aktivitätsaufruf.	Keine Transformation, da üblicherweise eine direkte oder indirekte (über eine Funktion) Verknüpfung zu einer oder mehreren Aktivitäten besteht.

11.6 Modellbildungsregeln in Modelica

Regel in Modelica	Beschreibung/Begründung
Die Anzahl der Gleichungen entspricht der Anzahl der Unbekannten. Bekannte Größen sind Parameter, Konstanten oder feste Zahlenwerte.	Das resultierende Gleichungssystem ist nur dann eindeutig lösbar, wenn das System weder unter- noch überbestimmt ist. Diese Regel wird als „Single Assignment Rule“ bezeichnet.
Die Namen der Variablen müssen sich voneinander und von den Namen der Komponenten unterscheiden.	Dies ist erforderlich, um eine eindeutige Übersetzung in Modelica-Code zu gewährleisten. Transformatoren müssen dies bei auftretenden Konflikten durch entsprechende Umbenennungen sicherstellen.
Die Namen der Signale verbundener Ports müssen identisch und vom gleichen Datentyp sein. Die Ordnung der einzelnen Signale eines Ports ist nicht relevant.	Ports werden im Rahmen der Transformation in Modelica in Konnektoren übersetzt. Signale bzw. damit verbundene Variablen sind Teil der Konnektoren. Für diese wird von Modelica Namensgleichheit und Typkonformität gefordert. Engineeringwerkzeuge müssen dies durch entsprechende Konsistenzprüfungen sicherstellen.
Ein Signal ausgehender Kausalität „DSLOUT“ kann nur mit einem akausalen Signal oder einem Signal eingehender Kausalität „DSLIN“ mittels eines Konnektors verknüpft werden.	Die ist aufgrund der Prämisse eines definierten Datenflusses erforderlich. Bei der Umwandlung des Modells in ein Gleichungssystem können nicht auflösbare Inkonsistenzen entstehen.
Ein Signal eingehender Kausalität „DSLIN“ kann nur mit einem akausalen Signal oder einem Signal ausgehender Kausalität „DSLOUT“ mittels eines Konnektors verknüpft werden.	Die ist aufgrund der Prämisse eines definierten Datenflusses erforderlich. Bei der Umwandlung des Modells in ein Gleichungssystem können nicht auflösbare Inkonsistenzen entstehen.
Kausale Signale gleichen Typs (DSLIN, DSLOUT) können mittels eines Konnektors miteinander verknüpft werden, wenn eines der beiden Signale Teil der Definition eines Durchgangsports ist.	Konnektoren einer Komponente mit eingehender bzw. ausgehender Kausalität werden innerhalb der Komponente als Konnektoren mit inverser Kausalität betrachtet.
Signale mit dem Attribut „IsFlow“ werden als positive Flüsse in die Komponente definiert	Dies ist notwendig, um Konnektoren eindeutig in Gleichungen umwandeln zu können. Die Summe aller Flüsse über einen Konnektor kann damit stets zu Null gesetzt werden.
Für bestimmte Operatoren sind im Sinne einer korrekten Modellierung ergänzende Rahmenbedingungen zu beachten. So darf beispielsweise bei der Anwendung des Ableitungsoperators „PMDER“ lediglich eine einzelne Variable als Operand verwendet werden.	Ergänzende Rahmenbedingungen sind notwendig, um die Entwicklung konsistenter und simulierbarer Modelle sicherzustellen. Die für die jeweiligen Operatoren geltenden Einschränkungen sind durch entsprechende Konsistenzprüfungen abzusichern und sollten durch die eingesetzten Entwicklungswerkzeuge unterstützt und implementiert werden.
Die Variabilität der Variablen der rechten Seite eines Ausdruckes muss geringer oder gleich sein, wie die Variabilität der Variablen der linken Seite.	Damit wird sichergestellt, dass Ausdrücke konsistent und eindeutig sind. So kann beispielsweise einer Konstanten zur Laufzeit nicht der Wert einer Variablen zugewiesen werden.
Werden Kontrollstrukturen zur Modellierung bedingt gültiger Gleichungen mit einem Operand vom Typ „PMIF“ verwendet, so muss auch ein Operand vom Typ „PMELSE“ vorhanden sein, wenn die Ausdrücke zur Formulierung der Gültigkeitsbedingung nicht ausschließlich aus Konstanten, festen Zahlenwerten und Parametern bestehen.	Nur wenn die Ausdrücke zur Formulierung der Gültigkeitsbedingung ausschließlich aus Konstanten, festen Zahlenwerten und Parametern bestehen, ist stets eindeutig festgelegt, ob die nachfolgenden Gleichungen als gültig zu betrachten sind oder nicht. Im Alternativfall würde die „Single Assignment Rule“ verletzt, was in einem über- oder unterbestimmten Gleichungssystem für das Modell resultieren würde.
Jeder Operand zur Formulierung alternativ gültiger Gleichungen muss die gleiche Anzahl von Gleichungen beinhalten.	Im Alternativfall würde die „Single Assignment Rule“ verletzt, was in einem nicht eindeutig bestimmten Gleichungssystem für das Modell resultieren würde.
Gleichungen innerhalb eines Operanden vom Typ „PMWHEN“ oder „PMELSEWHEN“ sind nur zu dem Zeitpunkt gültig, an dem die Kontrollbedingung erfüllt wird.	Nur beim Erfüllen der Kontrollbedingung (positive Flanke) werden die innerhalb des Operanden stehenden Gleichungen als Teil des Modells ausgewertet. Anschließend werden die innerhalb des Operanden zugewiesenen Werte konstant gehalten. Dadurch wird sichergestellt, dass die „Single Assignment Rule“ nicht verletzt wird.
Bei Operanden vom Typ „PMWHEN“ oder „PMELSEWHEN“ darf an mindestens einer Seite der Gleichung nur eine einzelne Variable stehen.	Somit kann der Solver beim Auswerten des Gleichungssystems feststellen, welche Variablen während der kontinuierlichen Berechnung konstant gehalten werden müssen.

Regel in Modelica	Beschreibung/Begründung
Zwei unterschiedliche Operanden vom Typ „PMWHEN“ dürfen nicht die selbe Variable durch eine Gleichung definieren.	Dadurch würde die „Single Assignment Rule“ verletzt, da bei gleichzeitiger Erfüllung der Kontrollbedingung der Operanden, zwei Gleichungen für eine Unbekannte existieren. Als Alternative wird die Anwendung eines Operanden vom Typ „PMELSEWHEN“ empfohlen.
Kontrollstrukturen, die Operanden vom Typ „PMWHEN“ oder „PMELSEWHEN“ enthalten, dürfen nicht verschachtelt werden.	Dies ist ebenfalls in einer potentiellen Verletzung der „Single Assignment Rule“ begründet.
Werden Kontrollstrukturen zur Modellierung bedingt gültiger Zuweisungen mit einem Operand vom Typ „PMIF“ verwendet, so muss nicht zwangsläufig ein Operand vom Typ „PMELSE“ vorhanden sein.	Da allen Variablen einer Sektion vom Typ „PMALGORITHM“ vor der Ausführung der Anweisungen ihre Initialisierungswerte zugewiesen werden, erfolgt keine Verletzung der „Single Assignment Rule“, auch wenn einzelne konditionelle Verhaltenselemente nicht ausgeführt werden.
Die Zuweisung eines Wertes an eine Variable darf innerhalb einer Sektion mehrfach, jedoch nicht in unterschiedlichen Sektionen erfolgen.	Dies würde zu einer Verletzung der „Single Assignment Rule“ führen.

iwb Forschungsberichte Band 1–121

Herausgeber: Prof. Dr.-Ing. J. Milberg und Prof. Dr.-Ing. G. Reinhart, Institut für Werkzeugmaschinen und Betriebswissenschaften der Technischen Universität München

Band 1–121 erschienen im Springer Verlag, Berlin, Heidelberg und sind im Erscheinungsjahr und den folgenden drei Kalenderjahren erhältlich im Buchhandel oder durch Lange & Springer, Otto-Suhr-Allee 26–28, 10585 Berlin

1 *Streifinger, E.*
Beitrag zur Sicherung der Zuverlässigkeit und Verfügbarkeit moderner Fertigungsmittel
1986 · 72 Abb. · 167 Seiten · ISBN 3-540-16391-3

2 *Fuchsberger, A.*
Untersuchung der spanenden Bearbeitung von Knochen
1986 · 90 Abb. · 175 Seiten · ISBN 3-540-16392-1

3 *Maier, C.*
Montageautomatisierung am Beispiel des Schraubens mit Industrierobotern
1986 · 77 Abb. · 144 Seiten · ISBN 3-540-16393-X

4 *Summer, H.*
Modell zur Berechnung verzweigter Antriebsstrukturen
1986 · 74 Abb. · 197 Seiten · ISBN 3-540-16394-8

5 *Simon, W.*
Elektrische Vorschubantriebe an NC-Systemen
1986 · 141 Abb. · 198 Seiten · ISBN 3-540-16693-9

6 *Büchs, S.*
Analytische Untersuchungen zur Technologie der Kugelbearbeitung
1986 · 74 Abb. · 173 Seiten · ISBN 3-540-16694-7

7 *Hunzinger, I.*
Schneiderodierte Oberflächen
1986 · 79 Abb. · 162 Seiten · ISBN 3-540-16695-5

8 *Pilland, U.*
Echtzeit-Kollisionsschutz an NC-Drehmaschinen
1986 · 54 Abb. · 127 Seiten · ISBN 3-540-17274-2

9 *Barthelmeß, P.*
Montagegerechtes Konstruieren durch die Integration von Produkt- und Montageprozeßgestaltung
1987 · 70 Abb. · 144 Seiten · ISBN 3-540-18120-2

10 *Reithofer, N.*
Nutzungssicherung von flexibel automatisierten Produktionsanlagen
1987 · 84 Abb. · 176 Seiten · ISBN 3-540-18440-6

11 *Diess, H.*
Rechnerunterstützte Entwicklung flexibel automatisierter Montageprozesse
1988 · 56 Abb. · 144 Seiten · ISBN 3-540-18799-5

12 *Reinhart, G.*
Flexible Automatisierung der Konstruktion und Fertigung elektrischer Leitungssätze
1988 · 112 Abb. · 197 Seiten · ISBN 3-540-19003-1

13 *Bürstner, H.*
Investitionsentscheidung in der rechnerintegrierten Produktion
1988 · 74 Abb. · 190 Seiten · ISBN 3-540-19099-6

14 *Groha, A.*
Universelles Zellenrechnerkonzept für flexible Fertigungssysteme
1988 · 74 Abb. · 153 Seiten · ISBN 3-540-19182-8

15 *Riese, K.*
Klipsmontage mit Industrierobotern
1988 · 92 Abb. · 150 Seiten · ISBN 3-540-19183-6

16 *Lutz, P.*
Leitsysteme für rechnerintegrierte Auftragsabwicklung
1988 · 44 Abb. · 144 Seiten · ISBN 3-540-19260-3

17 *Klippel, C.*
Mobiler Roboter im Materialfluß eines flexiblen Fertigungssystems
1988 · 86 Abb. · 164 Seiten · ISBN 3-540-50468-0

18 *Rascher, R.*
Experimentelle Untersuchungen zur Technologie der Kugelherstellung
1989 · 110 Abb. · 200 Seiten · ISBN 3-540-51301-9

19 *Heusler, H.-J.*
Rechnerunterstützte Planung flexibler Montagesysteme
1989 · 43 Abb. · 154 Seiten · ISBN 3-540-51723-5

20 *Kirchknopf, P.*
Ermittlung modaler Parameter aus Übertragungsfrequenzgängen
1989 · 57 Abb. · 157 Seiten · ISBN 3-540-51724-3

21 *Sauerer, Ch.*
Beitrag für ein Zerspanprozeßmodell Metallbandsägen
1990 · 89 Abb. · 166 Seiten · ISBN 3-540-51868-1

22 *Karstedt, K.*
Positionsbestimmung von Objekten in der Montage- und Fertigungsautomatisierung
1990 · 92 Abb. · 157 Seiten · ISBN 3-540-51879-7

23 *Peiker, St.*
Entwicklung eines integrierten NC-Planungssystems
1990 · 66 Abb. · 180 Seiten · ISBN 3-540-51880-0

24 *Schugmann, R.*
Nachgiebige Werkzeugaufhängungen für die automatische Montage
1990 · 71 Abb. · 155 Seiten · ISBN 3-540-52138-0

25 *Wrba, P.*
Simulation als Werkzeug in der Handhabungstechnik
1990 · 125 Abb. · 178 Seiten · ISBN 3-540-52231-X

26 *Eibelshäuser, P.*
Rechnerunterstützte experimentelle Modalanalyse mittels gestufter Sinusanregung
1990 · 79 Abb. · 156 Seiten · ISBN 3-540-52451-7

27 *Prasch, J.*
Computerunterstützte Planung von chirurgischen Eingriffen in der Orthopädie
1990 · 113 Abb. · 164 Seiten · ISBN 3-540-52543-2

28 *Teich, K.*
Prozeßkommunikation und Rechnerverbund in der Produktion
1990 · 52 Abb. · 158 Seiten · ISBN 3-540-52764-8

29 *Pfrang, W.*
Rechnergestützte und graphische Planung manueller und teilautomatisierter Arbeitsplätze
1990 · 59 Abb. · 153 Seiten · ISBN 3-540-52829-6

30 *Tauber, A.*
Modellbildung kinematischer Strukturen als Komponente der Montageplanung
1990 · 93 Abb. · 190 Seiten · ISBN 3-540-52911-X

31 *Jäger, A.*
Systematische Planung komplexer Produktionssysteme
1991 · 75 Abb. · 148 Seiten · ISBN 3-540-53021-5

32 *Hartberger, H.*
Wissensbasierte Simulation komplexer Produktionssysteme
1991 · 58 Abb. · 154 Seiten · ISBN 3-540-53326-5

33 *Tuczek, H.*
Inspektion von Karosseriepreßteilen auf Risse und Einschnürungen mittels Methoden der Bildverarbeitung
1992 · 125 Abb. · 179 Seiten · ISBN 3-540-53965-4

34 *Fischbacher, J.*
Planungsstrategien zur stömungstechnischen Optimierung von Reinraum-Fertigungsgeräten
1991 · 60 Abb. · 166 Seiten · ISBN 3-540-54027-X

35 *Moser, O.*
3D-Echtzeitkollisionsschutz für Drehmaschinen
1991 · 66 Abb. · 177 Seiten · ISBN 3-540-54076-8

36 *Naber, H.*
Aufbau und Einsatz eines mobilen Roboters mit unabhängiger Lokomotions- und Manipulationskomponente
1991 · 85 Abb. · 139 Seiten · ISBN 3-540-54216-7

37 *Kupec, Th.*
Wissensbasiertes Leitsystem zur Steuerung flexibler Fertigungsanlagen
1991 · 68 Abb. · 150 Seiten · ISBN 3-540-54260-4

38 *Maulhardt, U.*
Dynamisches Verhalten von Kreissägen
1991 · 109 Abb. · 159 Seiten · ISBN 3-540-54365-1

39 *Götz, R.*
Strukturierte Planung flexibel automatisierter Montagesysteme für flächige Bauteile
1991 · 86 Abb. · 201 Seiten · ISBN 3-540-54401-1

40 *Koepfer, Th.*
3D-grafisch-interaktive Arbeitsplanung - ein Ansatz zur Aufhebung der Arbeitsteilung
1991 · 74 Abb. · 126 Seiten · ISBN 3-540-54436-4

41 *Schmidt, M.*
Konzeption und Einsatzplanung flexibel automatisierter Montagesysteme
1992 · 108 Abb. · 168 Seiten · ISBN 3-540-55025-9

42 *Burger, C.*
Produktionsregelung mit entscheidungsunterstützenden Informationssystemen
1992 · 94 Abb. · 186 Seiten · ISBN 3-540-55187-5

43 *Hoßmann, J.*
Methodik zur Planung der automatischen Montage von nicht formstabilen Bauteilen
1992 · 73 Abb. · 168 Seiten · ISBN 3-540-5520-0

44 *Petry, M.*
Systematik zur Entwicklung eines modularen Programmbaukastens für robotergeführte Klebeprozesse
1992 · 106 Abb. · 139 Seiten · ISBN 3-540-55374-6

45 *Schönecker, W.*
Integrierte Diagnose in Produktionszellen
1992 · 87 Abb. · 159 Seiten · ISBN 3-540-55375-4

46 *Bick, W.*
Systematische Planung hybrider Montagesysteme unter Berücksichtigung der Ermittlung des optimalen Automatisierungsgrades
1992 · 70 Abb. · 156 Seiten · ISBN 3-540-55377-0

47 *Gebauer, L.*
Prozeßuntersuchungen zur automatisierten Montage von optischen Linsen
1992 · 84 Abb. · 150 Seiten · ISBN 3-540-55378-9

48 *Schrüfer, N.*
Erstellung eines 3D-Simulationssystems zur Reduzierung von Rüstzeiten bei der NC-Bearbeitung
1992 · 103 Abb. · 161 Seiten · ISBN 3-540-55431-9

49 *Wisbacher, J.*
Methoden zur rationellen Automatisierung der Montage von Schnellbefestigungselementen
1992 · 77 Abb. · 176 Seiten · ISBN 3-540-55512-9

50 *Garnich, F.*
Laserbearbeitung mit Robotern
1992 · 110 Abb. · 184 Seiten · ISBN 3-540-55513-7

51 *Eubert, P.*
Digitale Zustandesregelung elektrischer Vorschubantriebe
1992 · 89 Abb. · 159 Seiten · ISBN 3-540-44441-2

52 *Glaas, W.*
Rechnerintegrierte Kabelsatzfertigung
1992 · 67 Abb. · 140 Seiten · ISBN 3-540-55749-0

53 *Helml, H.J.*
Ein Verfahren zur On-Line Fehlererkennung und Diagnose
1992 · 60 Abb. · 153 Seiten · ISBN 3-540-55750-4

54 *Lang, Ch.*
Wissensbasierte Unterstützung der Verfügbarkeitsplanung
1992 · 75 Abb. · 150 Seiten · ISBN 3-540-55751-2

55 *Schuster, G.*
Rechnergestütztes Planungssystem für die flexibel automatisierte Montage
1992 · 67 Abb. · 135 Seiten · ISBN 3-540-55830-6

56 *Bomm, H.*
Ein Ziel- und Kennzahlensystem zum Investitionscontrolling komplexer Produktionssysteme
1992 · 87 Abb. · 195 Seiten · ISBN 3-540-55964-7

57 *Wendt, A.*
Qualitätssicherung in flexibel automatisierten Montagesystemen
1992 · 74 Abb. · 179 Seiten · ISBN 3-540-56044-0

58 *Hansmaier, H.*
Rechnergestütztes Verfahren zur Geräuschminderung
1993 · 67 Abb. · 156 Seiten · ISBN 3-540-56053-2

59 *Dilling, U.*
Planung von Fertigungssystemen unterstützt durch Wirtschaftssimulationen
1993 · 72 Abb. · 146 Seiten · ISBN 3-540-56307-5

60 *Strohmayr, R.*
Rechnergestützte Auswahl und Konfiguration von Zubringeeinrichtungen
1993 · 80 Abb. · 152 Seiten · ISBN 3-540-56652-X

61 *Glas, J.*
Standardisierter Aufbau anwendungsspezifischer Zellenrechnersoftware
1993 · 80 Abb. · 145 Seiten · ISBN 3.540-56890-5

62 *Stetter, R.*
Rechnergestütze Simulationswerkzeuge zur Effizienzsteigerung des Industrierobotereinsatzes
1994 · 91 Abb. · 146 Seiten · ISBN 3-540-56889-1

63 *Dirndorfer, A.*
Robotersysteme zur förderbandsynchronen Montage
1993 · 76 Abb. · 144 Seiten · ISBN 3-540-57031-4

64 *Wiedemann, M.*
Simulation des Schwingungsverhaltens spanender Werkzeugmaschinen
1993 · 81 Abb. · 137 Seiten · ISBN 3-540-57177-9

65 *Woenckhaus, Ch.*
Rechnergestütztes System zur automatisierten 3D-Layoutoptimierung
1994 · 81 Abb. · 140 Seiten · ISBN 3-540-57284-8

66 *Kummetsteiner, G.*
3D-Bewegungssimulation als integratives Hilfsmittel zur Planung manueller Montagesysteme
1994 · 62 Abb. · 146 Seiten · ISBN 3-540-57535-9

67 *Kugelmann, F.*
Einsatz nachgiebiger Elemente zur wirtschaftlichen Automatisierung von Produktionssystemen
1993 · 76 Abb. · 144 Seiten · ISBN 3-540-57549-9

68 *Schwarz, H.*
Simulationsgestützte CAD/CAM-Kopplung für die 3D-Laserbearbeitung mit integrierter Sensorik
1994 · 96 Abb. · 148 Seiten · ISBN 3-540-57577-4

69 *Viethen, U.*
Systematik zum Prüfen in flexiblen Fertigungssystemen
1994 · 70 Abb. · 142 Seiten · ISBN 3-540-57794-7

70 *Seehuber, M.*
Automatische Inbetriebnahme geschwindigkeitsadaptiver Zustandsregler
1994 · 72 Abb. · 155 Seiten · ISBN 3-540-57896-X

71 *Amann, W.*
Eine Simulationsumgebung für Planung und Betrieb von Produktionssystemen
1994 · 71 Abb. · 129 Seiten · ISBN 3-540-57924-9

72 *Schöpf, M.*
Rechnergestütztes Projektinformations- und Koordinationssystem für das Fertigungsvorfeld
1997 · 63 Abb. · 130 Seiten · ISBN 3-540-58052-2

73 *Welling, A.*
Effizienter Einsatz bildgebender Sensoren zur Flexibilisierung automatisierter Handhabungsvorgänge
1994 · 66 Abb. · 139 Seiten · ISBN 3-540-580-0

74 *Zetlmayer, H.*
Verfahren zur simulationsgestützten Produktionsregelung in der Einzel- und Kleinserienproduktion
1994 · 62 Abb. · 143 Seiten · ISBN 3-540-58134-0

75 *Lindl, M.*
Auftragsleittechnik für Konstruktion und Arbeitsplanung
1994 · 66 Abb. · 147 Seiten · ISBN 3-540-58221-5

76 *Zipper, B.*
Das integrierte Betriebsmittelwesen · Baustein einer flexiblen Fertigung
1994 · 64 Abb. · 147 Seiten · ISBN 3-540-58222-3

77 *Raith, P.*
Programmierung und Simulation von Zellenabläufen in der Arbeitsvorbereitung
1995 · 51 Abb. · 130 Seiten · ISBN 3-540-58223-1

78 *Engel, A.*
Strömungstechnische Optimierung von Produktionssystemen durch Simulation
1994 · 69 Abb. · 160 Seiten · ISBN 3-540-58258-4

79 *Zäh, M. F.*
Dynamisches Prozeßmodell Kreissägen
1995 · 95 Abb. · 186 Seiten · ISBN 3-540-58624-5

80 *Zwanzer, N.*
Technologisches Prozeßmodell für die Kugelschleifbearbeitung
1995 · 65 Abb. · 150 Seiten · ISBN 3-540-58634-2

81 *Romanow, P.*
Konstruktionsbegleitende Kalkulation von Werkzeugmaschinen
1995 · 66 Abb. · 151 Seiten · ISBN 3-540-58771-3

82 *Kahlenberg, R.*
Integrierte Qualitätssicherung in flexiblen Fertigungszellen
1995 · 71 Abb. · 136 Seiten · ISBN 3-540-58772-1

83 *Huber, A.*
Arbeitsfolgenplanung mehrstufiger Prozesse in der Hartbearbeitung
1995 · 87 Abb. · 152 Seiten · ISBN 3-540-58773-X

84 *Birkel, G.*
Aufwandsminimierter Wissenserwerb für die Diagnose in flexiblen Produktionszellen
1995 · 64 Abb. · 137 Seiten · ISBN 3-540-58869-8

85 *Simon, D.*
Fertigungsregelung durch zielgrößenorientierte Planung und logistisches Störungsmanagement
1995 · 77 Abb. · 132 Seiten · ISBN 3-540-58942-2

86 *Nedeljkovic-Groha, V.*
Systematische Planung anwendungsspezifischer Materialflußsteuerungen
1995 · 94 Abb. · 188 Seiten · ISBN 3-540-58953-8

87 *Rockland, M.*
Flexibilisierung der automatischen Teilbereitstellung in Montageanlagen
1995 · 83 Abb. · 168 Seiten · ISBN 3-540-58999-6

88 *Linner, St.*
Konzept einer integrierten Produktentwicklung
1995 · 67 Abb. · 168 Seiten · ISBN 3-540-59016-1

89 *Eder, Th.*
Integrierte Planung von Informationssystemen für rechnergestützte Produktionssysteme
1995 · 62 Abb. · 150 Seiten · ISBN 3-540-59084-6

90 *Deutschle, U.*
Prozeßorientierte Organisation der Auftragsentwicklung in mittelständischen Unternehmen
1995 · 80 Abb. · 188 Seiten · ISBN 3-540-59337-3

91 *Dieterle, A.*
Recyclingintegrierte Produktentwicklung
1995 · 68 Abb. · 146 Seiten · ISBN 3-540-60120-1

92 *Hechl, Chr.*
Personalorientierte Montageplanung für komplexe und variantenreiche Produkte
1995 · 73 Abb. · 158 Seiten · ISBN 3-540-60325-5

93 *Albertz, F.*
Dynamikgerechter Entwurf von Werkzeugmaschinen - Gestellstrukturen
1995 · 83 Abb. · 156 Seiten · ISBN 3-540-60608-8

94 *Trunzer, W.*
Strategien zur On-Line Bahnplanung bei Robotern mit 3D-Konturfolgesensoren
1996 · 101 Abb. · 164 Seiten · ISBN 3-540-60961-X

95 *Fichtmüller, N.*
Rationalisierung durch flexible, hybride Montagesysteme
1996 · 83 Abb. · 145 Seiten · ISBN 3-540-60960-1

96 *Trucks, V.*
Rechnergestützte Beurteilung von Getriebestrukturen in Werkzeugmaschinen
1996 · 64 Abb. · 141 Seiten · ISBN 3-540-60599-8

97 *Schäffer, G.*
Systematische Integration adaptiver Produktionssysteme
1996 · 71 Abb. · 170 Seiten · ISBN 3-540-60958-X

98 *Koch, M. R.*
Autonome Fertigungszellen - Gestaltung, Steuerung und integrierte Störungsbehandlung
1996 · 67 Abb. · 138 Seiten · ISBN 3-540-61104-5

99 *Moctezuma de la Barrera, J.L.*
Ein durchgängiges System zur computer- und rechnergestützten Chirurgie
1996 · 99 Abb. · 175 Seiten · ISBN 3-540-61145-2

100 *Geuer, A.*
Einsatzpotential des Rapid Prototyping in der Produktentwicklung
1996 · 84 Abb. · 154 Seiten · ISBN 3-540-61495-8

101 *Ebner, C.*
Ganzheitliches Verfügbarkeits- und Qualitätsmanagment unter Verwendung von Felddaten
1996 · 67 Abb. · 132 Seiten · ISBN 3-540-61678-0

102 *Pischeltsrieder, K.*
Steuerung autonomer mobiler Roboter in der Produktion
1996 · 74 Abb. · 171 Seiten · ISBN 3-540-61714-0

103 *Köhler, R.*
Disposition und Materialbereitstellung bei komplexen variantenreichen Kleinprodukten
1997 · 62 Abb. · 177 Seiten · ISBN 3-540-62024-9

104 *Feldmann, Ch.*
Eine Methode für die integrierte rechnergestützte Montageplanung
1997 · 71 Abb. · 163 Seiten · ISBN 3-540-62059-1

105 *Lehmann, H.*
Integrierte Materialfluß- und Layoutplanung durch Kopplung von CAD- und Ablaufsimulationssystem
1997 · 96 Abb. · 191 Seiten · ISBN 3-540-62202-0

106 *Wagner, M.*
Steuerungsintegrierte Fehlerbehandlung für maschinennahe Abläufe
1997 · 94 Abb. · 164 Seiten · ISBN 3-540-62656-5

107 *Lorenzen, J.*
Simulationsgestützte Kostenanalyse in produktorientierten Fertigungsstrukturen
1997 · 63 Abb. · 129 Seiten · ISBN 3-540-62794-4

108 *Krönert, U.*
Systematik für die rechnergestützte Ähnlichteilsuche und Standardisierung
1997 · 53 Abb. · 127 Seiten · ISBN 3-540-63338-3

109 *Pfersdorf, I.*
Entwicklung eines systematischen Vorgehens zur Organisation des industriellen Service
1997 · 74 Abb. · 172 Seiten · ISBN 3-540-63615-3

110 *Kuba, R.*
Informations- und kommunikationstechnische Integration von Menschen in der Produktion
1997 · 77 Abb. · 155 Seiten · ISBN 3-540-63642-0

111 *Kaiser, J.*
Vernetztes Gestalten von Produkt und Produktionsprozeß mit Produktmodellen
1997 · 67 Abb. · 139 Seiten · ISBN 3-540-63999-3

112 *Geyer, M.*
Flexibles Planungssystem zur Berücksichtigung ergonomischer Aspekte bei der Produkt- und Arbeitssystemgestaltung
1997 · 85 Abb. · 154 Seiten · ISBN 3-540-64195-5

113 *Martin, C.*
Produktionsregelung - ein modularer, modellbasierter Ansatz
1998 · 73 Abb. · 162 Seiten · ISBN 3-540-64401-6

114 *Löffler, Th.*
Akustische Überwachung automatisierter Fügeprozesse
1998 · 85 Abb. · 136 Seiten · ISBN 3-540-64511-X

115 *Lindermaier, R.*
Qualitätsorientierte Entwicklung von Montagesystemen
1998 · 84 Abb. · 164 Seiten · ISBN 3-540-64686-8

116 *Koehrer, J.*
Prozeßorientierte Teamstrukturen in Betrieben mit Großserienfertigung
1998 · 75 Abb. · 185 Seiten · ISBN 3-540-65037-7

117 *Schuller, R. W.*
Leitfaden zum automatisierten Auftrag von hochviskosen Dichtmassen
1999 · 76 Abb. · 162 Seiten · ISBN 3-540-65320-1

118 *Debuschewitz, M.*
Integrierte Methodik und Werkzeuge zur herstellungsorientierten Produktentwicklung
1999 · 104 Abb. · 169 Seiten · ISBN 3-540-65350-3

119 *Bauer, L.*
Strategien zur rechnergestützten Offline-Programmierung von 3D-Laseranlagen
1999 · 98 Abb. · 145 Seiten · ISBN 3-540-65382-1

120 *Pfob, E.*
Modellgestützte Arbeitsplanung bei Fertigungsmaschinen
1999 · 69 Abb. · 154 Seiten · ISBN 3-540-65525-5

121 *Spitznagel, J.*
Erfahrungsgeleitete Planung von Laseranlagen
1999 · 63 Abb. · 156 Seiten · ISBN 3-540-65896-3

Seminarberichte iwb

herausgegeben von Prof. Dr.-Ing. Gunther Reinhart und Prof. Dr.-Ing. Michael Zäh,
Institut für Werkzeugmaschinen und Betriebswissenschaften
der Technischen Universität München

Seminarberichte iwb sind erhältlich im Buchhandel oder beim
Herbert Utz Verlag, München, Fax 089-277791-01, info@utz.de

1 Innovative Montagesysteme · Anlagengestaltung, -bewertung und -überwachung
115 Seiten · ISBN 3-931327-01-9
2 Integriertes Produktmodell · Von der Idee zum fertigen Produkt
82 Seiten · ISBN 3-931327-02-7
3 Konstruktion von Werkzeugmaschinen · Berechnung, Simulation und Optimierung
110 Seiten · ISBN 3-931327-03-5
4 Simulation · Einsatzmöglichkeiten und Erfahrungsberichte
134 Seiten · ISBN 3-931327-04-3
5 Optimierung der Kooperation in der Produktentwicklung
95 Seiten · ISBN 3-931327-05-1
6 Materialbearbeitung mit Laser · von der Planung zur Anwendung
86 Seiten · ISBN 3-931327-76-0
7 Dynamisches Verhalten von Werkzeugmaschinen
80 Seiten · ISBN 3-931327-77-9
8 Qualitätsmanagement · der Weg ist das Ziel
130 Seiten · ISBN 3-931327-78-7
9 Installationstechnik an Werkzeugmaschinen · Analysen und Konzepte
120 Seiten · ISBN 3-931327-79-5
10 3D-Simulation · Schneller, sicherer und kostengünstiger zum Ziel
90 Seiten · ISBN 3-931327-10-8
11 Unternehmensorganisation · Schlüssel für eine effiziente Produktion
110 Seiten · ISBN 3-931327-11-6
12 Autonome Produktionssysteme
100 Seiten · ISBN 3-931327-12-4
13 Planung von Montageanlagen
130 Seiten · ISBN 3-931327-13-2
14 Nicht erschienen – wird nicht erscheinen
15 Flexible fluide Kleb/Dichtstoffe · Dosierung und Prozeßgestaltung
80 Seiten · ISBN 3-931327-15-9
16 Time to Market · Von der Idee zum Produktionsstart
80 Seiten · ISBN 3-931327-16-7
17 Industriekeramik in Forschung und Praxis · Probleme, Analysen und Lösungen
80 Seiten · ISBN 3-931327-17-5
18 Das Unternehmen im Internet · Chancen für produzierende Unternehmen
165 Seiten · ISBN 3-931327-18-3
19 Leittechnik und Informationslogistik · mehr Transparenz in der Fertigung
85 Seiten · ISBN 3-931327-19-1
20 Dezentrale Steuerungen in Produktionsanlagen · Plug & Play · Vereinfachung von Entwicklung und Inbetriebnahme
105 Seiten · ISBN 3-931327-20-5
21 Rapid Prototyping · Rapid Tooling · Schnell zu funktionalen Prototypen
95 Seiten · ISBN 3-931327-21-3
22 Mikrotechnik für die Produktion · Greifbare Produkte und Anwendungspotentiale
95 Seiten · ISBN 3-931327-22-1
24 EDM Engineering Data Management
195 Seiten · ISBN 3-931327-24-8
25 Rationelle Nutzung der Simulationstechnik · Entwicklungstrends und Praxisbeispiele
152 Seiten · ISBN 3-931327-25-6
26 Alternative Dichtungssysteme · Konzepte zur Dichtungsmontage und zum Dichtmittelauftrag
110 Seiten · ISBN 3-931327-26-4
27 Rapid Prototyping · Mit neuen Technologien schnell vom Entwurf zum Serienprodukt
111 Seiten · ISBN 3-931327-27-2
28 Rapid Tooling · Mit neuen Technologien schnell vom Entwurf zum Serienprodukt
154 Seiten · ISBN 3-931327-28-0
29 Installationstechnik an Werkzeugmaschinen · Abschlußseminar
156 Seiten · ISBN 3-931327-29-9
30 Nicht erschienen – wird nicht erscheinen
31 Engineering Data Management (EDM) · Erfahrungsberichte und Trends
183 Seiten · ISBN 3-931327-31-0
32 Nicht erschienen – wird nicht erscheinen
33 3D-CAD · Mehr als nur eine dritte Dimension
181 Seiten · ISBN 3-931327-33-7
34 Laser in der Produktion · Technologische Randbedingungen für den wirtschaftlichen Einsatz
102 Seiten · ISBN 3-931327-34-5
35 Ablaufsimulation · Anlagen effizient und sicher planen und betreiben
129 Seiten · ISBN 3-931327-35-3
36 Moderne Methoden zur Montageplanung · Schlüssel für eine effiziente Produktion
124 Seiten · ISBN 3-931327-36-1
37 Wettbewerbsfaktor Verfügbarkeit · Produktivitätsteigerung durch technische und organisatorische Ansätze
95 Seiten · ISBN 3-931327-37-X
38 Rapid Prototyping · Effizienter Einsatz von Modellen in der Produktentwicklung
128 Seiten · ISBN 3-931327-38-8
39 Rapid Tooling · Neue Strategien für den Werkzeug- und Formenbau
130 Seiten · ISBN 3-931327-39-6
40 Erfolgreich kooperieren in der produzierenden Industrie · Flexibler und schneller mit modernen Kooperationen
160 Seiten · ISBN 3-931327-40-X
41 Innovative Entwicklung von Produktionsmaschinen
146 Seiten · ISBN 3-89675-041-0
42 Stückzahlflexible Montagesysteme
139 Seiten · ISBN 3-89675-042-9
43 Produktivität und Verfügbarkeit · ...durch Kooperation steigern
120 Seiten · ISBN 3-89675-043-7
44 Automatisierte Mikromontage · Handhaben und Positionieren von Mikrobauteilen
125 Seiten · ISBN 3-89675-044-5
45 Produzieren in Netzwerken · Lösungsansätze, Methoden, Praxisbeispiele
173 Seiten · ISBN 3-89675-045-3
46 Virtuelle Produktion · Ablaufsimulation
108 Seiten · ISBN 3-89675-046-1

47 Virtuelle Produktion · Prozeß- und Produktsimulation
131 Seiten · ISBN 3-89675-047-X

48 Sicherheitstechnik an Werkzeugmaschinen
106 Seiten · ISBN 3-89675-048-8

49 Rapid Prototyping · Methoden für die reaktionsfähige Produktentwicklung
150 Seiten · ISBN 3-89675-049-6

50 Rapid Manufacturing · Methoden für die reaktionsfähige Produktion
121 Seiten · ISBN 3-89675-050-X

51 Flexibles Kleben und Dichten · Produkt-& Prozeßgestaltung, Mischverbindungen, Qualitätskontrolle
137 Seiten · ISBN 3-89675-051-8

52 Rapid Manufacturing · Schnelle Herstellung von Klein- und Prototypenserien
124 Seiten · ISBN 3-89675-052-6

53 Mischverbindungen · Werkstoffauswahl, Verfahrensauswahl, Umsetzung
107 Seiten · ISBN 3-89675-054-2

54 Virtuelle Produktion · Integrierte Prozess- und Produktsimulation
133 Seiten · ISBN 3-89675-054-2

55 e-Business in der Produktion · Organisationskonzepte, IT-Lösungen, Praxisbeispiele
150 Seiten · ISBN 3-89675-055-0

56 Virtuelle Produktion – Ablaufsimulation als planungsbegleitendes Werkzeug
150 Seiten · ISBN 3-89675-056-9

57 Virtuelle Produktion – Datenintegration und Benutzerschnittstellen
150 Seiten · ISBN 3-89675-057-7

58 Rapid Manufacturing · Schnelle Herstellung qualitativ hochwertiger Bauteile oder Kleinserien
169 Seiten · ISBN 3-89675-058-7

59 Automatisierte Mikromontage · Werkzeuge und Fügetechnologien für die Mikrosystemtechnik
114 Seiten · ISBN 3-89675-059-3

60 Mechatronische Produktionssysteme · Genauigkeit gezielt entwickeln
131 Seiten · ISBN 3-89675-060-7

61 Nicht erschienen – wird nicht erscheinen

62 Rapid Technologien · Anspruch – Realität – Technologien
100 Seiten · ISBN 3-89675-062-3

63 Fabrikplanung 2002 · Visionen – Umsetzung – Werkzeuge
124 Seiten · ISBN 3-89675-063-1

64 Mischverbindungen · Einsatz und Innovationspotenzial
143 Seiten · ISBN 3-89675-064-X

65 Fabrikplanung 2003 – Basis für Wachstum · Erfahrungen Werkzeuge Visionen
136 Seiten · ISBN 3-89675-065-8

66 Mit Rapid Technologien zum Aufschwung · Neue Rapid Technologien und Verfahren, Neue Qualitäten, Neue Möglichkeiten, Neue Anwendungsfelder
185 Seiten · ISBN 3-89675-066-6

67 Mechatronische Produktionssysteme · Die Virtuelle Werkzeugmaschine: Mechatronisches Entwicklungsvorgehen, Integrierte Modellbildung, Applikationsfelder
148 Seiten · ISBN 3-89675-067-4

68 Virtuelle Produktion · Nutzenpotenziale im Lebenszyklus der Fabrik
139 Seiten · ISBN 3-89675-068-2

69 Kooperationsmanagement in der Produktion · Visionen und Methoden zur Kooperation – Geschäftsmodelle und Rechtsformen für die Kooperation – Kooperation entlang der Wertschöpfungskette
134 Seiten · ISBN 3-98675-069-0

70 Mechatronik · Strukturdynamik von Werkzeugmaschinen
161 Seiten · ISBN 3-89675-070-4

71 Klebtechnik · Zerstörungsfreie Qualitätssicherung beim flexibel automatisierten Kleben und Dichten
ISBN 3-89675-071-2 · vergriffen

72 Fabrikplanung 2004 Ergfolgsfaktor im Wettbewerb · Erfahrungen – Werkzeuge – Visionen
ISBN 3-89675-072-0 · vergriffen

73 Rapid Manufacturing Vom Prototyp zur Produktion · Erwartungen – Erfahrungen – Entwicklungen
179 Seiten · ISBN 3-89675-073-9

74 Virtuelle Produktionssystemplanung · Virtuelle Inbetriebnahme und Digitale Fabrik
133 Seiten · ISBN 3-89675-074-7

75 Nicht erschienen – wird nicht erscheinen

76 Berührungslose Handhabung · Vom Wafer zur Glaslinse, von der Kapsel zur aseptischen Ampulle
95 Seiten · ISBN 3-89675-076-3

77 ERP-Systeme - Einführung in die betriebliche Praxis · Erfahrungen, Best Practices, Visionen
153 Seiten · ISBN 3-89675-077-7

78 Mechatronik · Trends in der interdisziplinären Entwicklung von Werkzeugmaschinen
155 Seiten · ISBN 3-89675-078-X

79 Produktionsmanagement
267 Seiten · ISBN 3-89675-079-8

80 Rapid Manufacturing · Fertigungsverfahren für alle Ansprüche
154 Seiten · ISBN 3-89675-080-1

81 Rapid Manufacturing · Heutige Trends – Zukünftige Anwendungsfelder
172 Seiten · ISBN 3-89675-081-X

82 Produktionsmanagement · Herausforderung Variantenmanagement
100 Seiten · ISBN 3-89675-082-8

83 Mechatronik · Optimierungspotenzial der Werkzeugmaschine nutzen
160 Seiten · ISBN 3-89675-083-6

84 Virtuelle Inbetriebnahme · Von der Kür zur Pflicht?
104 Seiten · ISBN 978-3-89675-084-6

85 3D-Erfahrungsforum · Innovation im Werkzeug- und Formenbau
375 Seiten · ISBN 978-3-89675-085-3

86 Rapid Manufacturing · Erfolgreich produzieren durch innovative Fertigung
162 Seiten · ISBN 978-3-89675-086-0

87 Produktionsmanagement · Schlank im Mittelstand
102 Seiten · ISBN 978-3-89675-087-7

88 Mechatronik · Vorsprung durch Simulation
134 Seiten · ISBN 978-3-89675-088-4

89 RFID in der Produktion · Wertschöpfung effizient gestalten
122 Seiten · ISBN 978-3-89675-089-1

Forschungsberichte iwb

herausgegeben von Prof. Dr.-Ing. Gunther Reinhart und Prof. Dr.-Ing. Michael Zäh,
Institut für Werkzeugmaschinen und Betriebswissenschaften
der Technischen Universität München

Forschungsberichte iwb ab Band 122 sind erhältlich im Buchhandel oder beim
Herbert Utz Verlag, München, Fax 089-277791-01, info@utz.de

122 Schneider, Burghard
Prozesskettenorientierte Bereitstellung nicht formstabiler Bauteile
1999 · 183 Seiten · 98 Abb. · 14 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-559-5

123 Goldstein, Bernd
Modellgestützte Geschäftsprozeßgestaltung in der Produktentwicklung
1999 · 170 Seiten · 65 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-546-3

124 Mößmer, Helmut E.
Methode zur simulationsbasierten Regelung zeitvarianter Produktionssysteme
1999 · 164 Seiten · 67 Abb. · 5 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-585-4

125 Gräser, Ralf-Gunter
Ein Verfahren zur Kompensation temperaturinduzierter Verformungen an Industrierobotern
1999 · 167 Seiten · 63 Abb. · 5 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-603-6

126 Trossin, Hans-Jürgen
Nutzung der Ähnlichkeitstheorie zur Modellbildung in der Produktionstechnik
1999 · 162 Seiten · 75 Abb. · 11 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-614-1

127 Kugelmann, Doris
Aufgabenorientierte Offline-Programmierung von Industrierobotern
1999 · 168 Seiten · 68 Abb. · 2 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-615-X

128 Diesch, Rolf
Steigerung der organisatorischen Verfügbarkeit von Fertigungszellen
1999 · 160 Seiten · 69 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-618-4

129 Lulay, Werner E.
Hybrid-hierarchische Simulationsmodelle zur Koordination teilautonomer Produktionsstrukturen
1999 · 182 Seiten · 51 Abb. · 14 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-620-6

130 Murr, Otto
Adaptive Planung und Steuerung von integrierten Entwicklungs- und Planungsprozessen
1999 · 178 Seiten · 85 Abb. · 3 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-636-2

131 Macht, Michael
Ein Vorgehensmodell für den Einsatz von Rapid Prototyping
1999 · 170 Seiten · 87 Abb. · 5 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-638-9

132 Mehler, Bruno H.
Aufbau virtueller Fabriken aus dezentralen Partnerverbünden
1999 · 152 Seiten · 44 Abb. · 27 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-645-1

133 Heitmann, Knut
Sichere Prognosen für die Produktionsptimierung mittels stochastischer Modelle
1999 · 146 Seiten · 60 Abb. · 13 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-675-3

134 Blessing, Stefan
Gestaltung der Materialflußsteuerung in dynamischen Produktionsstrukturen
1999 · 160 Seiten · 67 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-690-7

135 Abay, Can
Numerische Optimierung multivariater mehrstufiger Prozesse am Beispiel der Hartbearbeitung von Industriekeramik
2000 · 159 Seiten · 46 Abb. · 5 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-697-4

136 Brandner, Stefan
Integriertes Produktdaten- und Prozeßmanagement in virtuellen Fabriken
2000 · 172 Seiten · 61 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-715-6

137 Hirschberg, Arnd G.
Verbindung der Produkt- und Funktionsorientierung in der Fertigung
2000 · 165 Seiten · 49 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-729-6

138 Reek, Alexandra
Strategien zur Fokuspositionierung beim Laserstrahlschweißen
2000 · 193 Seiten · 103 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-730-X

139 Sabbah, Khalid-Alexander
Methodische Entwicklung störungstoleranter Steuerungen
2000 · 148 Seiten · 75 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-739-3

140 Schliffenbacher, Klaus U.
Konfiguration virtueller Wertschöpfungsketten in dynamischen, heterarchischen Kompetenznetzwerken
2000 · 187 Seiten · 70 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-754-7

141 Sprenzel, Andreas
Integrierte Kostenkalkulationsverfahren für die Werkzeugmaschinenentwicklung
2000 · 144 Seiten · 55 Abb. · 6 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-757-1

142 Gallasch, Andreas
Informationstechnische Architektur zur Unterstützung des Wandels in der Produktion
2000 · 150 Seiten · 69 Abb. · 6 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-781-4

143 Cuiper, Ralf
Durchgängige rechnergestützte Planung und Steuerung von automatisierten Montagevorgängen
2000 · 168 Seiten · 75 Abb. · 3 Tab. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-783-0

144 Schneider, Christian
Strukturmechanische Berechnungen in der Werkzeugmaschinenkonstruktion
2000 · 180 Seiten · 66 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-789-X

145 Jonas, Christian
Konzept einer durchgängigen, rechnergestützten Planung von Montageanlagen
2000 · 183 Seiten · 82 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-870-5

146 Willnecker, Ulrich
Gestaltung und Planung leistungsorientierter manueller Fließmontagen
2001 · 175 Seiten · 67 Abb. · broschiert · 20,5 x 14,5 cm · ISBN 3-89675-891-8

147 Lehner, Christof
Beschreibung des Nd:Yag-Laserstrahlschweißprozesses von Magnesiumdruckguss
2001 · 205 Seiten · 94 Abb. · 24 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0004-X

148 Rick, Frank
Simulationsgestützte Gestaltung von Produkt und Prozess am Beispiel Laserstrahlschweißen
2001 · 145 Seiten · 57 Abb. · 2 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0008-2

149 Höhn, Michael
Sensorgeführte Montage hybrider Mikrosysteme
2001 · 171 Seiten · 74 Abb. · 7 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0012-0

150 Böhl, Jörn
Wissensmanagement im Klein- und mittelständischen Unternehmen der Einzel- und Kleinserienfertigung
2001 · 179 Seiten · 88 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0020-1

151 Bürgel, Robert
Prozessanalyse an spanenden Werkzeugmaschinen mit digital geregelten Antrieben
2001 · 185 Seiten · 60 Abb. · 10 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0021-X

152 Stephan Dürrschmidt
Planung und Betrieb wandlungsfähiger Logistiksysteme in der variantenreichen Serienproduktion
2001 · 914 Seiten · 61 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0023-6

153 Bernhard Eich
Methode zur prozesskettenorientierten Planung der Teilebereitstellung
2001 · 132 Seiten · 48 Abb. · 6 Tabellen · 20,5 x 14,5 cm · ISBN 3-8316-0028-7

154 Wolfgang Rudorfer
Eine Methode zur Qualifizierung von produzierenden Unternehmen für Kompetenznetzwerke
2001 · 207 Seiten · 89 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0037-6

155 Hans Meier
Verteilte kooperative Steuerung maschinennaher Abläufe
2001 · 162 Seiten · 85 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0044-9

156 Gerhard Nowak
Informationstechnische Integration des industriellen Service in das Unternehmen
2001 · 203 Seiten · 95 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0055-4

157 Martin Werner
Simulationsgestützte Reorganisation von Produktions- und Logistikprozessen
2001 · 191 Seiten · 20,5 x 14,5 cm · ISBN 3-8316-0058-9

158 Bernhard Lenz
Finite Elemente-Modellierung des Laserstrahlschweißens für den Einsatz in der Fertigungsplanung
2001 · 150 Seiten · 47 Abb. · 5 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0094-5

159 Stefan Grunwald
Methode zur Anwendung der flexiblen integrierten Produktentwicklung und Montageplanung
2002 · 206 Seiten · 80 Abb. · 25 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0095-3

160 Josef Gartner
Qualitätssicherung bei der automatisierten Applikation hochviskoser Dichtungen
2002 · 165 Seiten · 74 Abb. · 21 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0096-1

161 Wolfgang Zeller
Gesamtheitliches Sicherheitskonzept für die Antriebs- und Steuerungstechnik bei Werkzeugmaschinen
2002 · 192 Seiten · 54 Abb. · 15 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0100-3

162 Michael Loferer
Rechnergestützte Gestaltung von Montagesystemen
2002 · 178 Seiten · 80 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0118-6

163 Jörg Fährer
Ganzheitliche Optimierung des indirekten Metall-Lasersinterprozesses
2002 · 176 Seiten · 69 Abb. · 13 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0124-0

164 Jürgen Höppner
Verfahren zur berührungslosen Handhabung mittels leistungsstarker Schallwandler
2002 · 132 Seiten · 24 Abb. · 3 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0125-9

165 Hubert Götte
Entwicklung eines Assistenzrobotersystems für die Knieendoprothetik
2002 · 258 Seiten · 123 Abb. · 5 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0126-7

166 Martin Weißenberger
Optimierung der Bewegungsdynamik von Werkzeugmaschinen im rechnergestützten Entwicklungsprozess
2002 · 210 Seiten · 86 Abb. · 2 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0138-0

167 Dirk Jacob
Verfahren zur Positionierung unterseitenstrukturierter Bauelemente in der Mikrosystemtechnik
2002 · 200 Seiten · 82 Abb. · 24 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0142-9

168 Ulrich Roßgoderer
System zur effizienten Layout- und Prozessplanung von hybriden Montageanlagen
2002 · 175 Seiten · 20,5 x 14,5 cm · ISBN 3-8316-0154-2

169 Robert Klingel
Anziehverfahren für hochfeste Schraubenverbindungen auf Basis akustischer Emissionen
2002 · 164 Seiten · 89 Abb. · 27 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0174-7

170 Paul Jens Peter Ross
Bestimmung des wirtschaftlichen Automatisierungsgrades von Montageprozessen in der frühen Phase der Montageplanung
2002 · 144 Seiten · 38 Abb. · 38 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0191-7

171 Stefan von Praun
Toleranzanalyse nachgiebiger Baugruppen im Produktentstehungsprozess
2002 · 250 Seiten · 62 Abb. · 7 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0202-6

172 Florian von der Hagen
Gestaltung kurzfristiger und unternehmensübergreifender Engineering-Kooperationen
2002 · 220 Seiten · 104 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0208-5

173 Oliver Kramer
Methode zur Optimierung der Wertschöpfungskette mittelständischer Betriebe
2002 · 212 Seiten · 84 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0211-5

174 Winfried Dohmen
Interdisziplinäre Methoden für die integrierte Entwicklung komplexer mechatronischer Systeme
2002 · 200 Seiten · 67 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0214-X

175 Oliver Anton
Ein Beitrag zur Entwicklung telepräsenter Montagesysteme
2002 · 158 Seiten · 85 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0215-8

176 Welf Broser
Methode zur Definition und Bewertung von Anwendungsfeldern für Kompetenznetzwerke
2002 · 224 Seiten · 122 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0217-4

177 Frank Breitinger
Ein ganzheitliches Konzept zum Einsatz des indirekten Metall-Lasersinterns für das Druckgießen
2003 · 156 Seiten · 20,5 x 14,5 cm · ISBN 3-8316-0227-1

178 Johann von Pieverling
Ein Vorgehensmodell zur Auswahl von Konturfertigungsverfahren für das Rapid Tooling
2003 · 163 Seiten · 88 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0230-1

179 Thomas Baudisch
Simulationsumgebung zur Auslegung der Bewegungsdynamik des mechatronischen Systems Werkzeugmaschine
2003 · 190 Seiten · 67 Abb. · 8 Tab. · 20,5 x 14,5 cm · ISBN 3-8316-0249-2

180 Heinrich Schieferstein
Experimentelle Analyse des menschlichen Kausystems
2003 · 132 Seiten · 20,5 x 14,5 cm · ISBN 3-8316-0251-4

181 Joachim Berlak
Methodik zur strukturierten Auswahl von Auftragsabwicklungssystemen
2003 · 244 Seiten · 20,5 x 14,5 cm · ISBN 3-8316-0258-1

182 Christian Meierlohr
Konzept zur rechnergestützten Integration von Produktions- und Gebäudeplanung in der Fabrikgestaltung
2003 · 181 Seiten · 84 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0292-1

183 Volker Weber
Dynamisches Kostenmanagement in kompetenzzentrierten Unternehmensnetzwerken
2004 · 210 Seiten · 64 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0330-8

184 Thomas Bongardt
Methode zur Kompensation betriebsabhängiger Einflüsse auf die Absolutgenauigkeit von Industrierobotern
2004 · 170 Seiten · 40 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0332-4

185 Tim Angerer
Effizienzsteigerung in der automatisierten Montage durch aktive Nutzung mechatronischer Produktkomponenten
2004 · 180 Seiten · 67 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0336-7

186 Alexander Krüger
Planung und Kapazitätsabstimmung stückzahlflexibler Montagesysteme
2004 · 197 Seiten · 83 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0371-5

187 Matthias Meindl
Beitrag zur Entwicklung generativer Fertigungsverfahren für das Rapid Manufacturing
2005 · 222 Seiten · 97 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0465-7

188 Thomas Fusch
Betriebsbegleitende Prozessplanung in der Montage mit Hilfe der Virtuellen Produktion am Beispiel der Automobilindustrie
2005 · 190 Seiten · 99 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0467-3

189 Thomas Mosandl
Qualitätssteigerung bei automatisiertem Klebstoffauftrag durch den Einsatz optischer Konturfolgesysteme
2005 · 182 Seiten · 58 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0471-1

190 Christian Patron
Konzept für den Einsatz von Augmented Reality in der Montageplanung
2005 · 150 Seiten · 61 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0474-6

191 Robert Cisek
Planung und Bewertung von Rekonfigurationsprozessen in Produktionssystemen
2005 · 200 Seiten · 64 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0475-4

192 Florian Auer
Methode zur Simulation des Laserstrahlschweißens unter Berücksichtigung der Ergebnisse vorangegangener Umformsimulationen
2005 · 160 Seiten · 65 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0485-1

193 Carsten Selke
Entwicklung von Methoden zur automatischen Simulationsmodellgenerierung
2005 · 137 Seiten · 53 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0495-9

194 Markus Seefried
Simulation des Prozessschrittes der Wärmebehandlung beim Indirekten-Metall-Lasersintern
2005 · 216 Seiten · 82 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0503-3

195 Wolfgang Wagner
Fabrikplanung für die standortübergreifende Kostensenkung bei marktnaher Produktion
2006 · 208 Seiten · 43 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0586-6

196 Christopher Ulrich
Erhöhung des Nutzungsgrades von Laserstrahlquellen durch Mehrfach-Anwendungen
2006 · 178 Seiten · 74 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0590-4

197 Johann Härtl
Prozessgaseinfluss beim Schweißen mit Hochleistungsdiodenlasern
2006 · 140 Seiten · 55 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0611-0

198 Bernd Hartmann
Die Bestimmung des Personalbedarfs für den Materialfluss in Abhängigkeit von Produktionsfläche und -menge
2006 · 208 Seiten · 105 Abb. · 20,5 x 14,5 cm · ISBN 3-8316-0615-3

199 Michael Schilp
Auslegung und Gestaltung von Werkzeugen zum berührungslosen Greifen kleiner Bauteile in der Mikromontage
2006 · 130 Seiten · 20,5 x 14,5 cm · ISBN 3-8316-0631-5

200 Florian Manfred Grätz
Teilautomatische Generierung von Stromlauf- und Fluidplänen für mechatronische Systeme
2006 · 192 Seiten · 20,5 x 14,5 cm · ISBN 3-8316-0643-9

201 Dieter Eireiner
Prozessmodelle zur statischen Auslegung von Anlagen für das Friction Stir Welding
2006 · 214 Seiten · 20,5 x 14,5 cm · ISBN 3-8316-0650-1

202 Gerhard Volkwein
Konzept zur effizienten Bereitstellung von Steuerungsfunktionalität für die NC-Simulation
2007 · 192 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0668-9

203 Sven Roeren
Komplexitätsvariable Einflussgrößen für die bauteilbezogene Struktursimulation thermischer Fertigungsprozesse
2007 · 224 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0680-1

204 Henning Rudolf
Wissensbasierte Montageplanung in der Digitalen Fabrik am Beispiel der Automobilindustrie
2007 · 200 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0697-9

205 Stella Clarke-Griebsch
Overcoming the Network Problem in Telepresence Systems with Prediction and Inertia
2007 · 150 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0701-3

206 Michael Ehrenstraßer
Sensoreinsatz in der telepräsenten Mikromontage
2008 · 160 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0743-3

207 Rainer Schack
Methodik zur bewertungsorientierten Skalierung der Digitalen Fabrik
2008 · 248 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0748-8

208 Wolfgang Sudhoff
Methodik zur Bewertung standortübergreifender Mobilität in der Produktion
2008 · 276 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0749-5

209 Stefan Müller
Methodik für die entwicklungs- und planungsbegleitende Generierung und Bewertung von Produktionsalternativen
2008 · 240 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0750-1

210 Ulrich Kohler
Methodik zur kontinuierlichen und kostenorientierten Planung produktionstechnischer Systeme
2008 · 232 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0753-2

211 Klaus Schlickenrieder
Methodik zur Prozessoptimierung beim automatisierten elastischen Kleben großflächiger Bauteile
2008 · 204 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0776-1

212 Niklas Möller
Bestimmung der Wirtschaftlichkeit wandlungsfähiger Produktionssysteme
2008 · 260 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0778-5

213 Daniel Siedl
Simulation des dynamischen Verhaltens von Werkzeugmaschinen während Verfahrbewegungen
2008 · 200 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0779-2

214 Dirk Ansorge
Auftragsabwicklung in heterogenen Produktionsstrukturen mit spezifischen Planungsfreiräumen
2008 · 146 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0785-3

215 Georg Wünsch
Methoden für die virtuelle Inbetriebnahme automatisierter Produktionssysteme
2008 · 224 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0795-2

216 Thomas Oertli
Strukturmechanische Berechnung und Regelungssimulation von Werkzeugmaschinen mit elektromechanischen Vorschubantrieben
2008 · 194 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0798-3

217 Bernd Petzold
Entwicklung eines Operatorarbeitsplatzes für die telepräsente Mikromontage
2008 · 234 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0805-8

218 Loucas Papadakis
Simulation of the Structural Effects of Welded Frame Assemblies in Manufacturing Process Chains
2008 · 260 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0813-3

219 Mathias Mörtl
Ressourcenplanung in der variantenreichen Fertigung
2008 · 210 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0820-1

220 Sebastian Weig
Konzept eines integrierten Risikomanagements für die Ablauf- und Strukturgestaltung in Fabrikplanungsprojekten
2008 · 232 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0823-2

221 Tobias Hornfeck
Laserstrahlbiegen komplexer Aluminiumstrukturen für Anwendungen in der Luftfahrtindustrie
2008 · 150 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0826-3

222 Hans Egermeier
Entwicklung eines Virtual-Reality-Systems für die Montagesimulation mit kraftrückkoppelnden Handschuhen
2008 · 210 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0833-1

223 Matthäus Sigl
Ein Beitrag zur Entwicklung des Elektronenstrahlsinterns
2008 · 185 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0841-6

224 Mark Harfensteller
Eine Methodik zur Entwicklung und Herstellung von Radiumtargets
2009 · 196 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0849-8

225 Jochen Werner
Methode zur roboterbasierten förderbandsynchronen Fließmontage am Beispiel der Automobilindustrie
2009 · 210 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0857-7

226 Florian Hagemann
Ein formflexibles Werkzeug für das Rapid Tooling beim Spritzgießen
2009 · 226 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0861-4

227 Haitham Rashidy
Knowledge-based quality control in manufacturing processes with application to the automotive industry
2009 · 212 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0862-1

228 Wolfgang Vogl
Eine interaktive räumliche Benutzerschnittstelle für die Programmierung von Industrierobotern
2009 · 200 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0869-0

229 Sonja Schedl
Integration von Anforderungsmanagement in den mechatronischen Entwicklungsprozess
2009 · 160 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0874-4

230 Andreas Trautmann
Bifocal Hybrid Laser Welding – A Technology for Welding of Aluminium and Zinc-Coated Steels
2009 · 268 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0876-8

231 Patrick Neise
Managing Quality and Delivery Reliability of Suppliers by Using Incentives and Simulation Models
2009 · 224 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0878-2

232 Christian Habicht
Einsatz und Auslegung zeitfensterbasierter Planungssysteme in überbetrieblichen Wertschöpfungsketten
2009 · 200 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0891-1

233 Michael Spitzweg
Methode und Konzept für den Einsatz eines physikalischen Modells in der Entwicklung von Produktionsanlagen
2009 · 180 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0931-4

234 Ulrich Munzert
Bahnplanungsalgorithmen für das robotergestützte Remote-Laserstrahlschweißen
2010 · 176 Seiten · 20,5 x 14,5 cm · ISBN 978-3-8316-0948-2

235 Georg Völlner
Rührreibschweißen mit Schwerlast-Industrierobotern
2010 · 232 Seiten · 20,5 x 14,5 cm · 978-3-8316-0955-0

236 Nils Müller
Modell für die Beherrschung und Reduktion von Nachfrageschwankungen
2010 · 270 Seiten · 20,5 x 14,5 cm · 978-3-8316-0992-5

237 Franz Decker
Unternehmensspezifische Strukturierung der Produktion als permanente Aufgabe
2010 · 180 Seiten · 20,5 x 14,5 cm · 978-3-8316-0996-3

238 Christian Lau
Methodik für eine selbstoptimierende Produktionssteuerung
2010 · 200 Seiten · 20,5 x 14,5 cm · 978-3-8316-4012-6

239 Christoph Rimpau
Wissensbasierte Risikobewertung in der Angebotskalkulation für hochgradig individualisierte Produkte
2010 · 200 Seiten · 20,5 x 14,5 cm · 978-3-8316-4015-7

240 Michael Loy
Modulare Vibrationswendelförderer zur flexiblen Teilezuführung
2010 · 169 Seiten · 20,5 x 14,5 cm · 978-3-8316-4027-0

241 Andreas Eursch
Konzept eines immersiven Assistenzsystems mit Augmented Reality zur Unterstützung manueller Aktivitäten in radioaktiven Produktionsumgebungen
2010 · 205 Seiten · 20,5 x 14,5 cm · 978-3-8316-4029-4

242 Florian Schwarz
Simulation der Wechselwirkungen zwischen Prozess und Struktur bei der Drehbearbeitung
2010 · 256 Seiten · 20,5 x 14,5 cm · 978-3-8316-4030-0

243 Martin Georg Prasch
Integration leistungsgewandelter Mitarbeiter in die variantenreiche Serienmontage
2010 · 261 Seiten · 20,5 x 14,5 cm · 978-3-8316-4033-1

244 Johannes Schilp
Adaptive Montagesysteme für hybride Mikrosysteme unter Einsatz von Telepräsenz
2011 · 160 Seiten · 20,5 x 14,5 cm · 978-3-8316-4063-8

245 Stefan Lutzmann
Beitrag zur Prozessbeherrschung des Elektronenstrahlschmelzens
2011 · 222 Seiten · 20,5 x 14,5 cm · 978-3-8316-4070-6

246 Gregor Branner
Modellierung transienter Effekte in der Struktursimulation von Schichtbauverfahren
2011 · 230 Seiten · 20,5 x 14,5 cm · 978-3-8316-4071-3

247 Josef Ludwig Zimmermann
Eine Methodik zur Gestaltung berührungslos arbeitender Handhabungssysteme
2011 · 184 Seiten · 20,5 x 14,5 cm · 978-3-8316-4091-1

248 Clemens Pörnbacher
Modellgetriebene Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme
2011 · 280 Seiten · 20,5 x 14,5 cm · 978-3-8316-4108-6