Matthias Hennig

Engineering-orientierte Steuerungsarchitektur auf der Basis von Aktionsprimitiven für Anwendungen in der Robotik

Beiträge aus der Automatisierungstechnik

Matthias Hennig

Engineering-orientierte Steuerungsarchitektur auf der Basis von Aktionsprimitiven für Anwendungen in der Robotik

VOGT

Dresden 2012

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

Bibliographic Information published by the Deutsche Bibliothek
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibiograpic data is available in the internet at http://dnb.ddb.de.

Zugl.: Dresden, Techn. Univ., Diss., 2012

Die vorliegende Arbeit stimmt mit dem Original der Dissertation „Engineering-orientierte Steuerungsarchitektur auf der Basis von Aktionsprimitiven für Anwendungen in der Robotik“ von Matthias Hennig überein.

Gesetzt vom Autor

ISBN 978-3-938860-54-0

Jörg Vogt Verlag
Niederwaldstr. 36
01277 Dresden
Germany

Phone: +49-(0)351-31403921
Telefax: +49-(0)351-31403918
e-mail: info@vogtverlag.de
Internet : www.vogtverlag.de

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik

Institut für Automatisierungstechnik

Engineering-orientierte Steuerungsarchitektur auf der Basis von Aktionsprimitiven für Anwendungen in der Robotik

Matthias Hennig

von der Fakultät Elektrotechnik und Informationstechnik der Technischen Universität Dresden

zur Erlangung des akademischen Grades eines

Doktoringenieurs

(Dr.-Ing.)

genehmigte Dissertation

Vorsitzender:	Prof. Dr.-Ing. habil. K. Röbenack
Gutachter:	Prof. Dr. techn. K. Janschek Prof. Dr.-Ing. A. Gräser
Tag der Einreichung:	02.04.2012
Tag der Verteidigung:	27.08.2012

Über den Autor

Matthias Hennig wurde am 25.05.1980 in Dessau geboren. Er absolvierte 1998 das Abitur am Gymnasium "Fürst Franz" in Dessau und studierte anschließend an der Technischen Universität Dresden Wirtschaftsingenieurwesen sowie ein Begleitstudium Regionalwissenschaften Lateinamerika. Dabei absolvierte er zahlreiche Praktika und arbeitete als studentischer Mitarbeiter an der Technischen Universität Dresden sowie am Fraunhofer Institut für Autonome Intelligente Systeme in Sankt Augustin bei Bonn. Nach seinem Diplomabschluss arbeitete er als wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik der Fakultät Elektrotechnik und Informationstechnik der Technischen Universität Dresden. Dort promovierte er im August 2012 mit der vorliegenden Arbeit zum Doktor-Ingenieur.

Danksagung

Mit der Drucklegung der Dissertation ist es an der Zeit, nochmals denjenigen zu danken, die mich während der Zeit meiner Promotion begleitet und tatkräftig unterstützt haben. Mein ganz besonderer Dank gilt dabei Herrn Prof. Dr. techn. Klaus Janschek für die Gelegenheit meine Promotion an seinem Institut anfertigen zu können. Bedanken möchte ich mich für seine stete Einsatzbereitschaft sowie die volle Unterstützung während meiner Zeit in Dresden.

Für die bereitwillige Übernahme des Zweitgutachtens sowie wertvolle ergänzende Hinweise bedanke ich mich vielmals bei Herrn Prof. Dr.-Ing. Axel Gräser.

Herrn Prof. Dr.-Ing. habil. Klaus Röbenack sowie Herrn Prof. Dr.-Ing. habil. Leon Urbas danke ich für die Mitwirkung an den Rigorosa sowie für die Durchführung der Verteidigung der Arbeit.

Herzlichen Dank an alle Kollegen meiner Zeit am Institut für Automatisierungstechnik. Ihr standet stets zu wissenschaftlichen und manchmal auch zu weniger wissenschaftlichen Diskussionen bereit. Für die kurze aber tiefe Freundschaft denke ich besonders an meinen Kollegen Martin Beck.

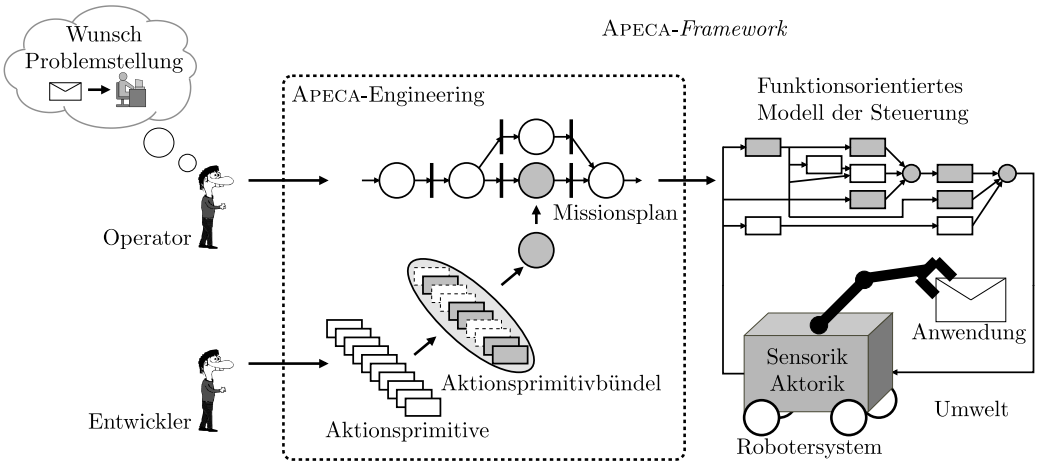
Mein Dank gilt auch meiner Familie sowie meinen Freunden, die mich während meiner Promotionszeit moralisch und auch tatkräftig unterstützt haben.

Ein besonderes Dankeschön geht an Herrn Wolfgang Weihmann. Die bei ihm besuchten Arbeitsgemeinschaften haben die frühen Grundlagen für diese Arbeit erst gelegt. Ihm sei die vorliegende Arbeit herzlich gewidmet.

Kurzfassung

Engineering-orientierte Steuerungsarchitektur auf der Basis von Aktionsprimitiven für Anwendungen in der Robotik

In der vorliegenden Arbeit wird die flexible Steuerungsarchitektur APECA für Systeme der Robotik sowie der robotergestützten Fertigungstechnik vorgestellt. Dafür werden verschiedene Anforderungen identifiziert und innerhalb eines Entwurfs vereint. Ein Hauptaugenmerk des dabei entstandenen Konzeptes ist es, einen vereinfachten Engineeringprozess für den Steuerungsentwurf zu ermöglichen. Dieser Ansatz wird durch die Verwendung von Aktionsprimitiven ermöglicht, die in Form atomarer Systemverhalten in einer speziellen Modulhierarchie eingesetzt werden. Hierzu erfolgt innerhalb der Steuerungsarchitektur eine Trennung zwischen einem funktionsorientierten verhaltensbasierten Modell zur hierarchischen sowie funktionell parallelen Ausführung von Aktionsprimitiven und einem ablauforientierten Modell zur aufgabenabhängigen Aktivierung derselben. Mit Hilfe eines Nutzerkonzepts werden diese Modelle verschiedenen Anwendern zugeordnet.

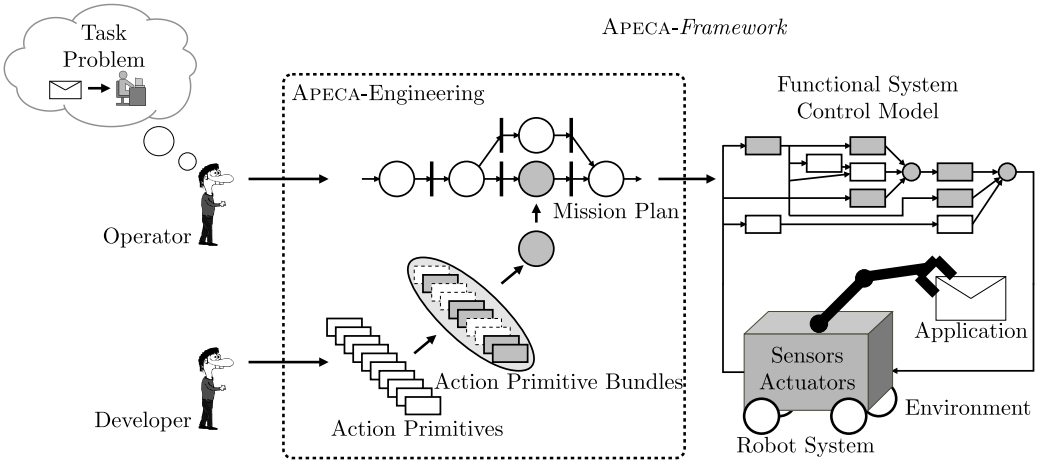


Die objektorientierte Realisierung dieses Entwurfs ermöglicht die Verwendung und Synchronisation von mehreren Teilsystemen innerhalb einer Steuerung. In der Arbeit wird sowohl der entstandene Entwurf diskutiert als auch eine prototypische Implementierung vorgestellt. Abschließend werden die Ergebnisse anhand verschiedener Demonstrationsszenarien präsentiert.

Abstract

Engineering-oriented Control Architecture based on Action Primitives for Applications in Robotics

In this present work, the APECA framework, a flexible control architecture for robotic systems, is introduced. The conceptual design combines different requirements identified in miscellaneous robotic control approaches. The main focus of the resulting concept is on a simplified engineering process for the controller design. This approach is supported by the use of atomic system behaviors, the so called action primitives, in a special module hierarchy. For this purpose a distinction between a functional behavior based system model with hierarchically and also parallelly executed action primitives and a sequential control system model with a task-dependent activation of the primitives is proposed. These models are assigned to different users through a distinct user concept.



An object-oriented implementation of the proposed architecture allows the utilization and synchronisation of multiple (sub-)systems within one framework. In this work the proposed framework will be discussed, a prototypical implementation will be presented and results based on different experimental scenarios will be shown.

Inhaltsverzeichnis

1 Problemstellung und Motivation **1**
1.1 Einleitung 1
1.2 Zielstellung und Aufbau der Arbeit 1

2 Stand der Technik **5**
2.1 Steuerungsarchitekturen für Systeme der Robotik 5
2.1.1 Funktionsorientierte Architekturen 7
2.1.2 Verhaltensbasierte Architekturen 12
2.1.2.1 Eigenschaften verhaltensbasierter Architekturen 13
2.1.2.2 Realisierung von Verhalten 19
2.1.2.3 Koordination paralleler Verhalten 20
2.1.3 Hybride Architekturen 21
2.2 Funktionsrealisierung über Aktionsprimitive 24
2.2.1 Übersicht über die Nutzung von Aktionsprimitiven . . 24
2.2.2 Formalisierung des Konzepts 27
2.3 Engineering von Robotersteuerungen 28
2.4 Offene Probleme und eigener Beitrag 30

3 Engineering-orientierte Architektur *APECA* **33**
3.1 Anforderungen an den Entwurf sowie Modellsystem 33
3.2 Modularer Aufbau der Architektur 36
3.2.1 Systembegriff 36
3.2.2 Aktionsprimitive 37
3.2.3 Funktionsorientiertes Modell der Aktionsprimitiv-Steuerung 41
3.2.4 Koordinierungsansatz mittels gewichteter Fusion . . . 43
3.2.5 Elementares Aktionsprimitiv-Ablaufmodell 46
3.2.6 Aktionsprimitivbündel 46
3.2.7 Ablaufmodell der Bündelsteuerung 49
3.2.8 Missionssteuerung 51
3.3 Integration innerhalb des *APECA-Frameworks* 54

4 Objektorientierte Softwarerealisierung der *APECA*-Module 59
4.1 Systementwurf 59
4.2 Entwurf der Softwarestruktur 60
4.3 Softwareverhalten in Aktivitätsdiagrammen 64
4.4 Resultierender Aufbau der Klassen 66
4.5 Komponenten des Softwareentwurfs 69
4.6 Implementierung des *Frameworks* 70

5 Engineering innerhalb der *APECA*-Architektur 71
5.1 Nutzerkonzept 71
5.2 Workflow zum Steuerungsentwurf 73
5.2.1 Allgemeines Vorgehen 73
5.2.2 Steuerungsarchitektur am Beispielszenario 74
5.2.3 Erstellung der anwendungsspezifischen Bibliotheken . 75
5.2.4 Konfiguration und Parametrierung des Missionsplans 78
5.3 Realisierung verteilter Systeme 79
5.3.1 Abgrenzung von Teilsystemen 82
5.3.2 Komponenten der Teilsysteme 85
5.3.3 Synchronisation von Teilsystemen 86

6 Demonstrationsbeispiele, Experimente und Verifikation 89
6.1 Vorgehen 89
6.2 Demonstrationsbeispiel A: Mobile Robotik 90
6.2.1 Mobiler Roboter *IfAbot* 90
6.2.2 Szenario „Transportaufgabe“ 92
6.2.2.1 Aufgabenstellung 93
6.2.2.2 Aktionsprimitive im funktionsorientierten Modell 93
6.2.2.3 Bündelsteuerung und Missionsplan 94
6.2.2.4 Simulation, Experimente und Auswertung . 96
6.2.3 Szenario „Search and Rescue“ 99
6.2.3.1 Aufgabenstellung 99
6.2.3.2 Aktionsprimitive im funktionsorientierten Modell 99
6.2.3.3 Bündelsteuerung und Missionsplan 102
6.2.3.4 Simulation, Experimente und Auswertung . 102
6.2.4 Szenario „MultiRobot“ 104
6.2.4.1 Aufgabenstellung 105

6.2.4.2 Aktionsprimitive im funktionsorientierten Modell . . . 106
6.2.4.3 Bündelsteuerung und Missionsplan . . . 107
6.2.4.4 Simulation und Auswertung der Ergebnisse 108
6.3 Demonstrationsbeispiel B: Fertigungstechnik . . . 111
6.3.1 Szenario „Verteilte Aufgabe“ . . . 111
6.3.1.1 Aufgabenstellung . . . 111
6.3.1.2 Aktionsprimitive im funktionsorientierten Modell . . . 112
6.3.1.3 Bündelsteuerung und Missionsplan . . . 113
6.3.1.4 Simulation und Auswertung der Ergebnisse 116
6.4 Evaluation der Ergebnisse . . . 119

7 Zusammenfassung und Ausblick 121
7.1 Diskussion der erreichten Ergebnisse . . . 121
7.2 Weiterführende Fragestellungen . . . 122

Anhang A Spezifikation des IfAbot 127

Anhang B Szenario „Transportaufgabe“ 133
B.1 Szenario „Transportaufgabe“ - Aktionsprimitiv-Bibliothek . 133
B.2 Szenario „Transportaufgabe“ - AP-Bündel-Bibliothek . . . 135
B.3 Szenario „Transportaufgabe“ - Experimente . . . 137

Anhang C Szenario „Search and Rescue“ 139
C.1 Szenario „Search and Rescue“ - Aktionsprimitiv-Bibliothek . 139
C.2 Szenario „Search and Rescue“ - AP-Bündel-Bibliothek . . . 141
C.3 Szenario „Search and Rescue“ - Experimente . . . 143

Anhang D Szenario „MultiRobot“ 145
D.1 Szenario „MultiRobot“ - Aktionsprimitiv-Bibliothek . . . 145
D.2 Szenario „MultiRobot“ - AP-Bündel-Bibliothek . . . 147
D.3 Szenario „MultiRobot“ - Simulationsergebnis . . . 149

Anhang E Szenario „Verteilte Aufgabe“ 151
E.1 Szenario „Verteilte Aufgabe“ - Aktionsprimitiv-Bibliothek . . . 151
E.2 Szenario „Verteilte Aufgabe“ - AP-Bündel-Bibliothek . . . 153
E.3 Szenario „Verteilte Aufgabe“ - Simulationsergebnis . . . 155

Literaturverzeichnis 157

Abbildungsverzeichnis

2.1 Funktionsorientierter Architekturansatz 8
2.2 Verhaltensbasierter Architekturansatz 12
2.3 Funktionsorientiertes Modell eines Verhaltens 20
2.4 Hybrider Architekturansatz 22

3.1 Modellsystem zum Architekturentwurf 34
3.2 Funktionsorientiertes Modell eines abstrakten Aktionsprimitivs 40
3.3 Funktionsorientiertes Modell der Aktionsprimitivsteuerung . . 41
3.4 Fusion einzelner Aktionsprimitivausgänge 44
3.5 Objektorienterter Entwurf der Aktionsprimitiv-Ablaufsteuerung 47
3.6 Funktionsorientiertes Modell eines Aktionsprimitivbündels . 48
3.7 Erweitertes funktionsorientiertes Modell der Architektur . . 50
3.8 Ablaufmodell der Architektur 51
3.9 Ablaufsteuerung der Missionssteuerung 52
3.10 Zeitlicher Verlauf auf Missionssteuerungsebene 53
3.11 Gesamtansicht des APECA-*Frameworks* 55

4.1 Systemkontextdiagramm 61
4.2 Facharchitekturdiagramm 62
4.3 Anwendungsfalldiagramm 63
4.4 Aktivitätsdiagramme zum Engineeringprozess 64
4.5 Aktivitätsdiagramme zur Laufzeitumgebung 65
4.6 Klassendiagramm 68
4.7 Komponentendiagramm 69

5.1 Beispielszenario zur Anwendungsarchitektur 75
5.2 Funktionsorientiertes Modell des Beispielszenarios 77
5.3 Missionsplan für das Beispielszenario 78
5.4 Beispielszenario zur verteilten Aufgaben 80
5.5 Abgrenzung von Teilsystemen 83
5.6 Markierungsgraph zur Abgrenzung von Teilsystemen 84
5.7 Missionsplan mit Synchronisation 87
5.8 Funktionsorientierte Modelle bei verteilten Aufgaben 88

6.1 Abbildung des mobilen Roboters *IfAbot* 92
6.2 Funktionsorientiertes Modell zum Szenario „Transportaufgabe“ 95
6.3 Missionsplan des Szenarios „Transportaufgabe“ 96
6.4 Gefahrener Pfad im Szenario „Transportaufgabe“ 98
6.5 Aktionsprimitivwichtungen im Szenario „Transportaufgabe“ 98
6.6 Funktionsorientiertes Modell zum Szenario „Search and Rescue“ 100
6.7 Missionsplan des Szenarios „Search and Rescue“ 103
6.8 Gefahrener Pfad im Szenario „Search and Rescue“ 104
6.9 Aktionsprimitivwichtungen im Szenario „Search and Rescue“ 105
6.10 Funktionsorientiertes Modell zum Szenario „MultiRobot“ . . 108
6.11 Missionsplan des Szenarios „MultiRobot“ 109
6.12 Simulationsumgebung zum Szenario „MultiRobot“ 110
6.13 Simulationsverlauf zum Szenario „MultiRobot“ 110
6.14 Funktionsorientierte Modelle zum Szenario „Verteilte Aufgabe“ 114
6.15 Missionsplan des Szenarios „Verteilte Aufgabe“ 115
6.16 Synchroner Werkstücktransport im Szenario „Verteilte Aufgabe“ 117
6.17 Zeitverlauf und Synchronisation im Szenario „Verteilte Aufgabe“ 118

A.1 Prinzipieller Aufbau des *IfAbot* 127
A.2 Koordinatensysteme des *IfAbot* 129
A.3 Umgebungskarte des *IfAbot* 130

Tabellenverzeichnis

3.1 Anforderungen zur Steuerungsarchitektur 35

4.1 Klassen und Zuordnung zu Subsystemen 67

5.1 Hierarchieebenen innerhalb der APECA-Architektur 72
5.2 Workflow zum APECA-Engineering 74
5.3 Komponenten bei verschiedenen Realisierungsoptionen . . . 85

6.1 Eingesetzte Sensorik des mobilen Roboters *IfAbot* 91
6.2 Beschreibung der Daten im Szenario „Transportaufgabe“ . . . 94
6.3 Zeiterfordernisse im Szenario „Transportaufgabe“ 97
6.4 Beschreibung der Daten im Szenario „Search and Rescue“ . . 101
6.5 Zeiterfordernisse im Szenario „Search and Rescue“ 104
6.6 Beschreibung der Daten im Szenario „MultiRobot“ 107
6.7 Beschreibung der Daten im Szenario „Verteilte Aufgabe“ . . 113

A.1 Technische Daten des auf dem *IfAbot* verbauten *Onboard-PC* 128

B.1 Aktionsprimitive des Szenarios „Transportaufgabe“ 134
B.2 Aktionsprimitivbündel des Szenarios „Transportaufgabe“ . . 136
B.3 Experimente des Szenarios „Transportaufgabe“ 138

C.1 Aktionsprimitive des Szenarios „Search and Rescue“ 140
C.2 Aktionsprimitivbündel des Szenarios „Search and Rescue“ . 142
C.3 Experimente des Szenarios „Search and Rescue“ 143

D.1 Aktionsprimitive des Szenarios „MultiRobot“ 146
D.2 Aktionsprimitivbündel des Szenarios „MultiRobot“ 148
D.3 Simulation des Szenarios „MultiRobot“ 149

E.1 Aktionsprimitive des Szenarios „Verteilte Aufgabe“ 152
E.2 Aktionsprimitivbündel des Szenarios „Verteilte Aufgabe“ . . . 154
E.3 Simulation des Szenarios „Verteilte Aufgabe“ 155

Symbol- und Abkürzungsverzeichnis

Abkürzungen

APECA	Name der in der vorliegenden Arbeit entwickelten Steuerungsarchitektur (A*ction* P*rimitive-based* E*ngineering-oriented* C*ontrol* A*rchitecture)*
Abb.	Abbildung
AP	Aktionsprimitiv
APB	Aktionsprimitivbündel
Gl.	Gleichung
MP	Missionsplan

Symbole

$\dot{\boldsymbol{p}}_{\mathrm{TCP}}$	Endeffektorgeschwindigkeit
$\boldsymbol{\lambda}_i$	Parametrierbarer Satz von Abbruchbedingungen zum Beenden von Aktionsprimitiven
$\boldsymbol{\Phi}_{\mathrm{v}_i}$	Spezifische Eingangsdaten eines Aktionsprimitivs
$\boldsymbol{\Phi}_{\mathrm{w}_i}$	Spezifische Ausgangsdaten eines Aktionsprimitivs
$\boldsymbol{\tau}_i$	Werkzeugkommandos für externe Aktorik zur Ausführung eines Aktionsprimitivs
$\boldsymbol{p}_{\mathrm{TCP}}$	Endeffektorpose
$\boldsymbol{z}_{0_i}$	Satz von Anfangswerten respektive Parametern eines Aktionsprimitivs
$\hat{X}$, $\hat{Y}$	Koordinatensystem Dreigelenk
$\hat{X}_{\mathrm{G}}$, $\hat{Y}_{\mathrm{G}}$	Globales Roboterkoordinatensystem
$\hat{X}_{\mathrm{R}}$, $\hat{Y}_{\mathrm{R}}$	Lokales Roboterkoordinatensystem
$\mathcal{APB}_j$	Aktionsprimitivbündel als Kombination gleichzeitig aktivierter Aktionsprimitive
$\mathcal{AP}_i$	Aktionsprimitiv der AP-Bibliothek; formalisierte atomare Roboteraktion
$\mathcal{A}_i$	Atomare konkrete Aktion innerhalb eines Aktionsprimitivs
$\mathcal{E}_j$	Effekt aller Aktionsprimitive innerhalb eines Aktionsprimitivbündels
$\mathcal{MS}_g$	Missionsschritt im Missionsplan als individuell parametriertes Aktionsprimitivbündel

ω	Rotatorische Geschwindigkeit
θ_{G}	Globale Roboterorientierung
θ_{R}	Lokale Roboterorientierung
a_i	Aktivierungsvariable eines Aktionsprimitivs
G	Anzahl der im Missionsplan gespeicherten Missionsschritte
g	Index des aktuellen Missionsschritts
h_j	Anzahl der in einem APB enthaltenen Aktionsprimitive
i	Index des aktuellen Aktionsprimitivs
j	Index des aktuellen Aktionsprimitivbündels
M	Anzahl der in der AP-Bibliothek gespeicherten Aktionsprimitive
N	Anzahl der in der APB-Bibliothek gespeicherten Aktionsprimitivbündel
v	Translatorische Geschwindigkeit
v_l	Linke Radgeschwindigkeit
v_r	Rechte Radgeschwindigkeit
$W_{\mathcal{AP},i}$	Wichtungsvariable innerhalb eines Aktionsprimitivs

1 Problemstellung und Motivation

1.1 Einleitung

Automatisierte Systeme erleichtern dem Anwender heute in mannigfaltiger Weise das Leben. Robotiksysteme sind dabei vielleicht die faszinierendsten Systeme, da sie vielfach dem Anwender, also dem Menschen, nachempfunden sind und/oder mit ihm in direktem Kontakt stehen. Damit solche Systeme konkrete Aufgabenstellungen bearbeiten und lösen können, verfügen sie über eine Steuerung. Diese hat die Aufgabe, verschiedene Sensorsignale und Eingangsinformationen zu verarbeiten, um daraus Stellsignale für das Robotersystem zu erzeugen. Der Aufbau der wesentlichen Funktionsblöcke dieser Steuerung sowie die softwaretechnische Organisationsstruktur werden dabei durch die Steuerungsarchitektur beschrieben. Neben der bloßen Möglichkeit, mit einer Architektur anforderungskonforme und gegenüber Unsicherheiten aus der Umwelt(-modellierung) robuste Steuerungen erstellen zu können, kommt jedoch auch dem Aspekt des Engineering innerhalb solcher Steuerungen große Bedeutung zu. Unter Engineering wird in diesem Zusammenhang der benutzerorientierte Gestaltungsprozess zur Realisierung einer konkreten aufgabenorientierten Steuerung des Robotiksystems verstanden. Darunter fallen sowohl die Steuerungserstellung als auch die flexible Modifikation derselben bei sich ändernden Aufgaben. Die Erstellung und Wartung von Steuerungen sollte demnach möglichst komfortabel, übersichtlich und transparent erfolgen. Gerade diesen Aspekten wird in der Literatur nur wenig Aufmerksamkeit geschenkt. Daher wird im Rahmen dieser Arbeit versucht, wichtige Forderungen des allgemeinen robusten Aufbaus von Steuerungen für Robotersysteme mit Anforderungen aus dem Bereich des Engineerings von Steuerungen zu verbinden, um daraus eine neue Architektur zur Bewältigung komplexer Aufgaben zu entwerfen.

1.2 Zielstellung und Aufbau der Arbeit

Für die erfolgreiche Kooperation von technischem System und menschlichem Anwender ist es notwendig, Aufgaben intuitiv und übersichtlich realisieren zu können. Dabei müssen auch solche Anwender im Fokus behalten werden, die

über kein umfangreiches technisches (Detail-)Wissen über die eingesetzten Robotersysteme verfügen. Zielstellung der Arbeit ist daher die Entwicklung einer speziellen Engineering-orientierten Steuerungsarchitektur für Robotersysteme, welche zugleich eine flexible und adaptive Lösung komplexer Aufgabenstellungen erlaubt.

Zur Bearbeitung der geschilderten Problemstellung wird die Arbeit wie folgt gegliedert:

- Das **Kapitel 1** bildet die vorliegende Einleitung.
- **Kapitel 2** stellt grundlegende Steuerungsarchitekturen auf Basis einer detaillierten Literaturrecherche vor. Hier werden Anforderungen identifiziert und offene Probleme zusammengestellt. Außerdem wird der Ansatz der Aktionsprimitive erläutert. Dabei soll die Frage beantwortet werden, welche Anforderungen problemorientierte Steuerungsarchitekturen erfüllen sollten und warum derzeit existierende Steuerungen diesbezüglich Schwächen aufweisen.
- In **Kapitel 3** wird anschließend die Engineering-orientierte Steuerungsarchitektur APECA auf Basis der zuvor erstellten Anforderungen vorgestellt. Dazu werden innerhalb eines *Bottom-Up-Ansatzes* einzelne Komponenten theoretisch entwickelt und anschaulich erklärt. Darauf folgend werden diese zu einem *Framework* integriert.
- Das folgende **Kapitel 4** befasst sich mit der Fragestellung, wie die zuvor entworfene Steuerungsarchitektur innerhalb eines Softwareentwurfs benutzerfreundlich realisiert werden kann. Dazu wird eine objektorientierte Analyse des entstandenen APECA-*Frameworks* durchgeführt. Auf dieser Basis werden schließlich Softwarekomponenten erstellt und implementiert.
- Innerhalb von **Kapitel 5** wird der APECA-Engineeringprozess zur Anwendungsrealisierung ausführlich beschrieben. Dabei werden alle notwendigen Steuerungsmodule mithilfe von anschaulichen Beispielmodellen erstellt.
- **Kapitel 6** beschreibt die durchgeführten Experimente anhand von Demonstrationsbeispielen. Dabei werden verschiedene Aufgabenstellungen definiert und anschließend mittels des vorgestellten APECA-Engineeringansatzes jeweils in eine konkrete Steuerung überführt. Die

so entstehenden Systeme werden im Folgenden simulativ und experimentell untersucht und bewertet. Zielstellung dieses Abschnitts ist die Evaluation der entstandenen Steuerungsarchitektur hinsichtlich der Leistungsfähigkeit sowie der Anwendbarkeit.

- In **Kapitel 7** werden die erreichten Ergebnisse zusammengefasst. Dabei wird ferner auf offene Probleme eingegangen.
- **Anhang A** stellt die in den Experimenten verwendete Roboterplattform *IfAbot* detailliert vor.
- Innerhalb der **Anhänge B bis E** werden vertiefende Informationen über die durchgeführten Demonstrationsbeispiele und experimentellen Ergebnisse gegeben.

2 Stand der Technik

Das folgende Kapitel untersucht bereits existierende Steuerungsarchitekturen. Dazu wird anfangs allgemein auf Begriffe und Definitionen eingegangen. Anschließend werden verschiedene Ansätze und Paradigmen zu Steuerungsarchitekturen vorgestellt und anhand von Beispielen aus der Literatur veranschaulicht. Dabei werden jeweils Vor- und Nachteile sowie ableitbare Anforderungen identifiziert. Auf Basis dieser Recherchen werden im Folgenden offene Probleme dargestellt, um daraus einen neuen Ansatz zu generieren. Dieser wird zum Abschluss des Kapitels kurz skizziert und anschließend innerhalb der nachfolgenden Kapitel ausführlich vorgestellt.

2.1 Steuerungsarchitekturen für Systeme der Robotik

Mechatronische **Systeme der Robotik**, wie beispielsweise mobile Roboter oder die robotergestützte Fertigungstechnik, bestehen von außen betrachtet in erster Linie aus *Hardware*. Um mit solchen Systemen nun aufgabenorientiert und effizient zu arbeiten, ist eine spezielle Software zur Steuerung derselben notwendig. Diese Software besteht dabei typischerweise aus verschiedenen Programmbausteinen, welche die unterschiedlichen Teilgebiete der Steuerungsstruktur widerspiegeln. Teilgebiete, wie beispielsweise Sensorik und Wahrnehmung, Aktorik und Bewegungssteuerung, Wissensverarbeitung sowie Aktionsplanung und Durchführung, werden so zu einem Gesamtsystem integriert. Dazu werden die einzelnen Softwarekomponenten innerhalb einer Architektur vereint, der **Steuerungsarchitektur** (engl. *Control Architecture*). Unter dem Begriff der **Architektur** wird dabei die softwaretechnische Organisation der Steuerung zusammengefasst. Steuerungsarchitekturen für mechatronische Systeme der Robotik werden demnach in erster Linie als Softwarearchitekturen realisiert und legen die Organisationsform fest, in der aus den sensorischen Wahrnehmungen des Systems Handlungsanweisungen für die Aktorik erzeugt werden (vgl. [Russell und Norvig, 2003], [Bekey, 2005]).

Der Steuerungsarchitektur fallen neben der einfachen Integration der einzelnen Module noch weitere Aufgaben zu. So müssen Funktionen unterer Ebenen, wie beispielsweise die Einbindung heterogener Sensoren oder die Ansteuerung der Aktorik, ebenso unterstützt werden wie Programmbausteine

höherer Ebenen, beispielsweise die Aktionskontrolle oder das Management von (teilweise widersprüchlichen) Aufgaben. Aktivitäten wie die Reaktion auf verschiedene Ereignisse, die Planung oder die Durchführungsüberwachung können demnach auf unterschiedlichen Ebenen erfolgen. Solche Ebenen unterscheiden sich oft durch den Grad der „Intelligenz“ - messbar durch den Grad der Komplexität einer dort zu bearbeitenden Aufgabe. Die eigentliche Aufgabe der Steuerungsarchitektur ist demnach die Verwaltung heterogener Softwarekomponenten sowie die Bereitstellung von geeigneten Schnittstellen im zu steuernden Gesamtsystem. Dabei steht heutzutage die Bewältigung von komplexen Aufgaben innerhalb dynamischer Umgebungen unter Echtzeitbedingungen im Vordergrund.

Heute existiert bereits eine Vielzahl von speziellen Steuerungsarchitekturen für Systeme der Robotik sowie der robotergestützten Fertigungstechnik. Dabei wird in allen Quellen die Unterscheidung in drei klassische *Paradigmen*, also Leitlinien der Ausgestaltung von Steuerungsarchitekturen, beschrieben (vgl. [Siciliano und Khatib, 2008], [Murphy, 2000], [Arkin, 1998]). Diese unterscheiden sich vorrangig in der Art und Weise wie eingehende Daten verarbeitet werden. Unterschieden werden dabei:

***Sense-Plan-Act-Paradigma*:** Hierbei findet eine *sequentielle Verarbeitung* der Sensordaten und Generierung von Handlungsanweisungen unter Zuhilfenahme eines zentralen Weltmodells zur Planung von Aktionen statt. In Form von Aktionssequenzen werden Stellbefehle an die Aktorik weitergereicht. Aus diesem *Paradigma* leiten sich die *funktionsorientierten Architekturen* ab, welche in Abschnitt 2.1.1 näher erläutert werden.

***Sense-Act-Paradigma*:** Eine Sammlung von softwaretechnisch einfachen Einzelverhalten bewirkt eine unmittelbare und direkte Reaktion auf Ereignisse. Durch Sensordaten als Eingangsinformationen dieser Verhalten wird ein *reaktives Systemverhalten* ermöglicht. Darunter wird nach Siciliano und Khatib [2008] die direkte Kopplung von Sensordaten an die Aktorik über algorithmisch einfache Verhalten ohne zusätzliche (übergeordnete) Module verstanden. Damit ist das Robotersystem in der Lage, in dynamischen und unstrukturierten Umgebungen schnell auf Sensorinformationen zu reagieren. Andere Systeme, speziell die funktionsorientierten Architekturen, reagieren auf Sensordaten natürlich auch mit einer Reaktion und besitzen demnach ein „reaktives“ Verhalten. Jedoch findet der Begriff speziell bei Systemen nach dem *Sense-Act-Paradigma* Anwendung, da hier auf einen Sensorwert (Stimulus)

unmittelbar (also in der Regel sehr schnell) eine Antwort (Response) folgt. Die Daten aller beteiligten Einzelverhalten werden dabei *parallel* verarbeitet. Auf diesem Ansatz basieren die in Abschnitt 2.1.2 vorgestellten *verhaltensbasierten Steuerungen*.

***Hybrides Paradigma*:** Bei diesem Ansatze werden die oben genannten reaktiven Fähigkeiten einer verhaltensbasierten Steuerung mit einer aufgabenorientierten Planung und Aktivierung vereint. Dabei findet eine Nutzung von Einzelverhalten mit übergeordneten Modulen zur situationsabhängigen Auswahl von lösungsorientierten Verhalten statt. Die darauf basierenden *hybriden Architekturen* werden in Kapitel 2.1.3 behandelt.

Konkrete Anwendungen können selbstredend mit jeder der auf den genannten Paradigmen basierenden Architekturen realisiert werden. Dabei entstehen jedoch spezifische Vor- und Nachteile bei der Umsetzung der jeweiligen Aufgabe. Auf diese Eigenschaften sowie die zugehörigen Vorzüge bzw. Probleme der verschiedenen Steuerungsarchitekturen wird darum innerhalb der folgenden Abschnitte näher eingegangen.

2.1.1 Funktionsorientierte Architekturen

Der funktionsorientierte (engl. *functional* oder auch *deliberative*) Ansatz zur Gestaltung von Steuerungsarchitekturen ist gleichzeitig die älteste und somit konventionellste in der Literatur vertretene Betrachtungsweise. Wie in Abbildung 2.1 gezeigt, werden dabei im Wesentlichen drei funktionsorientierte Grundmodule unterschieden (vgl. [Nehmzow, 2003]). Im Modul **Sensorik** werden die an der Steuerung angeschlossenen Sensoren ausgelesen und vorverarbeitet. So wird beispielsweise ein Kamerabild vom Sensorsystem aufgenommen. In einem **Planungsmodul** werden die Daten weiterverarbeitet und anschließend mittels vorhandenem *a-priori*-Wissen und der aktuellen Systemaufgabe neue Handlungsanweisungen berechnet. In erkannten Merkmalen des Kamerabildes wird hier beispielsweise ein gesuchtes Objekt erkannt und anhand von gespeicherten Karteninformationen ein Pfad zu diesem Zielobjekt berechnet. Letztlich werden im Modul **Aktorik** die berechneten Stellbefehle an die angeschlossene Aktorik weitergegeben. Hierbei können zum Beispiel Entfernung und Richtung zum Zielobjekt in Form von Stellbefehlen an die Motoren gesendet werden. Aufgrund dieser Dreiteilung werden funktionsorientierte Architekturen dem *Sense-Plan-Act-Paradigma* zugeordnet.

Wie in Abbildung 2.1 weiterhin zu erkennen ist, kann das Planungsmodul in weitere funktionsorientierte Einheiten unterteilt werden. Dabei übernimmt

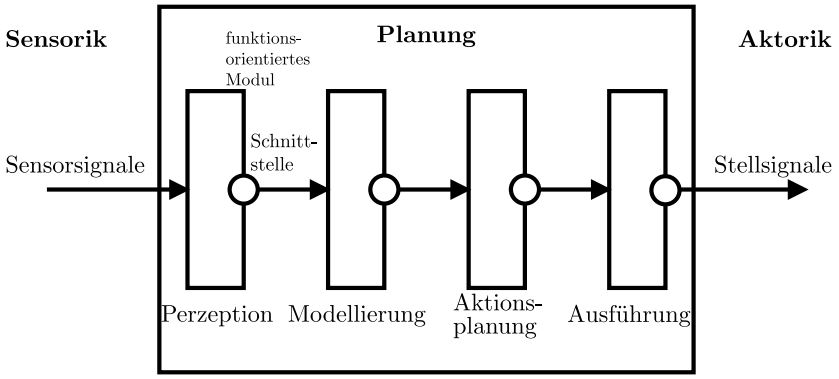


Abbildung 2.1: Beim funktionsorientierten Architekturansatz werden die Sensorsignale der Perzeption in Form von Datenflüssen innerhalb eines Planungsmoduls sequentiell in den einzelnen Modulen verarbeitet und über Schnittstellen ausgetauscht. Generierte Handlungsanweisungen werden anschließend in Form von Stellsignalen an die Aktorik weitergeleitet (nach [Siciliano und Khatib, 2008]).

die **Perzeption** die Auswertung der Sensordaten. Anschließend werden diese Daten mithilfe eines Weltmodells, also aller über die Umwelt bekannten *a-priori*-Informationen, **modelliert**. Darauf folgt in der **Aktionsplanung** eine methodische Erzeugung von Handlungsanweisungen zur Erfüllung der konkret gestellten Aufgabe. Im **Ausführungsmodul** werden dann die berechneten Handlungsanweisungen in Stellbefehle für die Aktorik umgerechnet und an die jeweiligen Aktorikmodule weitergegeben.

Einzelne Module werden über Schnittstellen verbunden. Der Vorteil der Kombination unabhängiger Module über Schnittstellen ist mit der Eigenschaft begründet, dass jedes Modul als eigenständige (sogenannte *stand-alone*) Anwendung entwickelt werden kann und nur über definierte Schnittstellen mit den anderen Modulen kommuniziert. Funktionsorientierte Architekturen sind deshalb für heterogene Module verschiedener Softwarebausteine geeignet und in der Entwicklungsphase unabhängig editierbar. Einzelne Subsysteme lassen sich daher unterschiedlich zu einem Gesamtsystem kombinieren. Neue Module können einfach ausgetauscht oder zusätzlich integriert werden, solange die vorher definierten Schnittstellen die *Kompatibilität* bewahren. Änderungen in einem Subsystem werden somit vielfach nicht in anderen Modulen wirksam und betreffen somit nicht automatisch das Gesamtsystem. Hieraus kann eine wichtige Forderung für den Aufbau von Steuerungsarchitekturen abgeleitet werden.

Anforderung A1:

Ein modularer Aufbau der Softwarearchitektur unterstützt die transparente Entwicklung, Realisierung und Änderung der Steuerung.

Wie gezeigt, verfügen funktionsorientierte Architekturen über eine Repräsentationskomponente. Dies bedeutet, dass die Realität in Form von Modellen und Regeln in die Entscheidungsprozesse des Systems eingeht. Dadurch wird es möglich das Systemverhalten so zu planen und zu koordinieren, dass auch langfristige Ziele erreicht werden. Damit werden in Abhängigkeit von der Vollständigkeit des Modells zielführende Entscheidungen getroffen. Hieraus ergeben sich weitere Anforderungen an Steuerungsarchitekturen.

Anforderung A2:

Die Steuerungsarchitektur soll in der Lage sein, komplexere Aufgaben mithilfe von gegebenem *a-priori*-Wissen in Form von Modellen zu lösen.

Anforderung A3:

Die Bearbeitung und Lösung von an das System gestellten Aufgaben mit kurz- und langfristigen Zielen soll durch die Steuerung realisiert werden.

Konkrete Realisierungen der funktionsorientierten Architektur auf einem mobilen Roboter existieren seit Nilsson [1984]. Der Roboter verfügt dort über eine Kamera und wertet die Bildinformationen anhand eines internen Modells über die Umwelt aus. Daraufhin werden Aktionen geplant und ausgeführt. Damit ist der Roboter in der Lage Zielobjekte anzusteuern. Wahrnehmung und Verarbeitung finden innerhalb der funktionsorientierten Architektur demnach getrennt statt.

Durch Albus et al. [1989] wird mit NASREM eine standardisierte funktionsorientierte Architektur beschrieben. Diese basiert auf den Modulen Sensorverarbeitung, Weltmodellierung sowie einem Modul zur Aufgabenplanung und -ausführung. Dabei besitzt jedes Modul zusätzlich unterschiedliche Hierarchieebenen. So können einfache Aufgaben komplett auf unterer Ebene verarbeitet werden, ohne höhere Ebenen zu sehr in Anspruch zu nehmen.

Durch Albus und Rippey [1994] wurde dieser Architektur noch eine zusätzliche Ebene hinzugefügt. Ein Evaluierungsmodul bewertet Unsicherheiten und Nutzen von Sensorinformationen und Aktionen. Dazu werden Kostenfunktionen zur Abschätzung dieser Kenngrößen realisiert. Voraussetzung dafür ist ein umfangreiches Wissen über die dazu benötigten Informationen innerhalb des Weltmodells.

Durch Hans [2005] wird eine funktionsorientierte Architektur für Anwendungen in der Servicerobotik vorgestellt. Um einer damit verbundenen gesteigerten Komplexität der Aufgaben begegnen zu können, steht eine Vielzahl verschiedener Aktionen im System in Form von Softwarebausteinen, den Makros, bereit. Pläne werden in Form von Aktionslisten vom System bereitgestellt. Die sequentielle Bearbeitung solcher Listen löst eine gestellte Systemaufgabe. Dazu wird eine erweiterte mehrschichtige Steuerungsstruktur vorgestellt, für welche der Anwender Systemwissen innerhalb mehrerer Ebenen benötigt.

In Hennig [2005] und Nüchter et al. [2005] wird ebenfalls eine funktionsorientierte Architektur zur Steuerung eines (teil-)autonomen mobilen Roboters für Aufgaben im Bereich *Search and Rescue* vorgestellt. Dabei werden verschiedene Systemaufgaben auf unterschiedliche Module verteilt. Jedes Modul läuft hierfür in einem eigenen *Thread* und kommuniziert mit anderen Komponenten über Schnittstellen. Der Ausfall einer Komponente führt damit nicht zum Versagen des Gesamtsystems, da bestimmte verbleibende Aufgaben weiterhin erfüllt werden können. Problematisch gestaltet sich hier die Verwaltung der Funktionalität innerhalb einer Komponente aufgrund der dadurch gestiegenen Komplexität. So betreffen auch kleine Änderungen der Funktionalität jeweils das gesamte Modul. Dadurch ist bei jeder Modifikation bereits ein tiefes Verständnis des Programmablaufs erforderlich. So entsteht ein robustes System zur Lösung einer spezifischen Aufgabe mit dem Nachteil spätere Modifikationen nur unter Umständen vornehmen zu können.

Steuerungsarchitekturen für Systeme der robotergestützten Fertigungstechnik basieren im Prinzip auch auf dem funktionsorientierten Ansatz. In Sciavicco und Siciliano [2005] wird dabei ebenfalls vorgeschlagen solche Steuerungsarchitekturen in drei hierarchische Module zu zerlegen. Ein

Sensormodul verarbeitet dazu Positionsdaten der Kinematik. Diese Informationen werden innerhalb eines Planungsmoduls mit Hilfe zusätzlicher Umweltinformationen verarbeitet, um Bewegungskommandos zu erzeugen. Innerhalb eines Entscheidungsmoduls werden letztlich Aktionen ausgelöst und die Aktorik angesteuert. Zusätzlich wird ein Schnittstellenmodul zum Operator, also dem direkten Anwender des Systems, integriert. Innerhalb des hierarchischen Modells der funktionsorientierten Robotersteuerung werden demnach Systemaufgaben auf Motorkommandos abgebildet.

Typische Aufgaben von Manipulatoren sind beispielsweise der Transport von Objekten mittels *Pick-and-Place*-Anwendungen, Montageprozesse (*Assembling*) sowie Bearbeitung von Objekten mit Werkzeugen. Da Industrieroboter für die Erfüllung solcher Aufgaben vornehmlich Bewegungen ausführen, ist die Bewegungsplanung daher die zentrale Aufgabe des Planers. Nach Hesse und Seitz [1996] und Bartenschlager et al. [1998] sind Robotersteuerungen für Systeme der robotergestützten Fertigungstechnik daher üblicherweise als (Multi-)Punktsteuerung bzw. Bahnsteuerung realisiert, wobei innerhalb des Planungsmoduls zumeist vorher definierte Punkte einer Trajektorie hintereinander an das Ausführungsmodul und schließlich an die Aktorik gesendet werden. An bestimmten Punkten kann zusätzlich ein Werkzeug aktiviert werden, um die eigentliche (industrielle) Bearbeitungsaufgabe zu erfüllen.

Je nach Abstraktionsgrad des realisierten Planungsmoduls kann hierbei nach Siegert und Bocionek [1996] zwischen zwei Programmierebenen unterschieden werden. Bei der roboterorientierten Programmierung erfolgt die Bewegungssteuerung des Roboters über die explizite Vorgabe von Trajektorien und Werkzeugkommandos zur Ansteuerung der Aktorik. Solche Vorgaben werden in Form von Listen oder im *Teach-In*-Verfahren vom Anwender vollständig in das System übertragen. Bei der aufgabenorientierten Programmierung spezifiziert ein Operator lediglich die Roboteraufgabe, welche von der Steuerung in einzelne Schritte einer tieferen Abstraktionsebene zerlegt und ausgeführt wird. So wird beispielsweise bloß der Start- sowie Zielpunkt und die Bearbeitungsaufgabe im System hinterlegt. Der aufgabenorientierte Ansatz erscheint daher eher geeignet komplexere Systeme der intelligenten Fertigungstechnik realisieren zu können.

Funktionsorientierte Architekturen bieten durch die Lösung einer gestellten Aufgabe innerhalb eines Planers den Vorteil, in der Regel eine vollständige bzw. global optimale Lösung zu erreichen. Natürlich geht damit eine algorithmische Schwerfälligkeit einher. Diese ist einerseits durch die Autonomie der einzelnen Module begründet, die nur über Schnittstellen kommunizieren.

Andererseits neigen funktionsorientierte Architekturen zu langsamen Reaktionen und langen Rechenzeiten, da für jede aktuelle Aufgabe stets alle Module sequentiell durchlaufen und anhand des Weltmodells abgeglichen werden. Weiterhin entstehen durch die begrenzte Vollständigkeit bei der Modellierung des Weltmodells aufgrund von Unsicherheiten Probleme mit der Fehlertoleranz und Adaptierbarkeit an aktuelle Gegebenheiten in der Umwelt. In dynamischen Umgebungen sind funktionsorientierte Steuerungen demnach unter bestimmten Voraussetzungen nicht mehr ausreichend robust gegenüber Unsicherheiten und zudem unflexibel. Lösungsmöglichkeiten für die geschilderte Problematik zeigt der folgende Abschnitt auf.

2.1.2 Verhaltensbasierte Architekturen

Den zweiten klassischen Ansatz für Steuerungsarchitekturen bildet die Klasse der verhaltensbasierten (engl. *behavior-based*) Architekturen. Dabei findet nach Arkin [1998] für jede Wahrnehmung durch die Sensorik eine direkte Kopplung an die Aktorik über ein Verarbeitungsmodul statt. Jedes dieser Module bildet hierbei von den Eingangssignalen der Sensoren über die Berechnung der benötigten Handlungsanweisung bis zur Ausgabe über die Aktorik ein sogenanntes Verhalten (engl. *behavior*). Wie in Abbildung 2.2 gezeigt, arbeiten mehrere solcher Verhalten innerhalb der Steuerung verteilt und parallel und können untereinander über verschiedene Mechanismen interagieren und dynamisch auf Ereignisse aus der Umgebung reagieren.

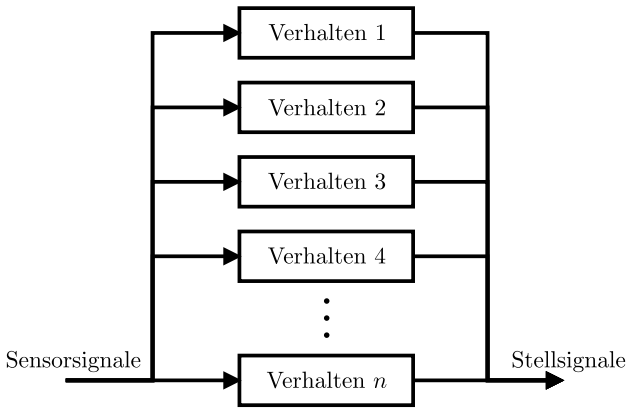


Abbildung 2.2: Innerhalb verhaltensbasierter Architekturen arbeiten verschiedene Verhalten parallel. Dabei werden Eingangsinformationen aus der Sensorik verarbeitet und konkurrierende Ausgangsdaten als Stellsignale für die Aktorik erzeugt (nach [Nehmzow, 2003]).

Komplexere Aufgaben werden in der Regel von Verhalten auf höheren Hierarchieebenen bearbeitet, da rechentechnisch aufwändige Planungsmodule nicht existieren und auch kein übergeordnetes zentrales Weltmodell vorliegt. Aus diesem Grund werden verhaltensbasierte Architekturen dem *Sense-Act-Paradigma* bzw. *Perception-Action-Ansatz* zugeordnet. Durch diese Form der sensorgetriebenen Steuerung erhalten verhaltenbasiert realisierte Systeme ein reaktives Verhalten (vgl. Seite 6). Auf dynamische Ereignisse in der Umwelt kann daher schnell und flexibel reagiert werden (vgl. [Bekey, 2005]).

Einzelne Verhalten realisieren hierbei jeweils nur eine Basisfähigkeit. Erst durch die Anzahl gemeinsam parallel aktivierter Verhalten gewinnt das Gesamtsystem an Komplexität. Durch diese gleichzeitige Bearbeitung mehrerer verschiedener Verhalten können auch anspruchsvollere Aufgaben gelöst werden. Der in diesem Zusammenhang stehende Begriff der *Emergenz* beschreibt dieses Phänomen: Das Gesamtsystem aller Fähigkeiten ist demnach von außen betrachtet „intelligenter“ als die reine Summe der einzelnen Verhalten (vgl. [Nehmzow, 2003]). Als Vorbild für den Aufbau verhaltenbasierter Architekturen dient dabei häufig die „innere Architektur“ einfacher Lebewesen, wie beispielsweise Insekten. Solche biologisch motivierten verhaltensbasierten Steuerungen sind somit auch ein aktueller Forschungsgegenstand für das Verständnis biologischer sowie kognitiver Prozesse. Um einen Gesamtüberblick über verschiedene verhaltensbasiert realisierte Steuerungen zu erhalten, gibt der nächste Abschnitt eine Übersicht.

2.1.2.1 Eigenschaften verhaltensbasierter Architekturen

Es gibt bis heute keine allgemein genutzte verhaltensbasierte Architektur. Daher existieren derzeit viele Ausprägungen unterschiedlicher Steuerungen. Unterschiede sind dabei vor allem im Aufbau und der Verschaltung einzelner Verhalten, der Existenz und Realisierung von Hierarchien und Parallelitäten sowie der Fusion der Einzelverhalten zu suchen. Aktuelle Veröffentlichungen zum Thema beschäftigen sich sowohl mit allgemeinen Fragestellungen, wie dem generellen Aufbau der Architektur, als auch mit ganz speziellen Details, wie beispielsweise Strategien der Verhaltensfusion bei Multi-Robot-Anwendungen. Daher wird an dieser Stelle versucht, relevante verhaltensbasierte Grundkonzepte und Architekturen vorzustellen und wesentliche gemeinsame Anforderungen sowie Probleme zu identifizieren.

Von Walter [1950] wurde der allgemein anerkannte Grundstein für verhaltensbasierte Systeme gelegt, ohne diese bereits so zu benennen. Es handelt sich dabei um eine einfache elektromechanische Apparatur, in welcher Lichtsen-

soren über eine Verstärkerschaltung direkt mit zwei Motoren verbunden sind. Dieser „Roboter“ war damit in der Lage, ein Basisverhalten zum Anfahren von Lichtquellen zu realisieren.

Verallgemeinert wurde dieser Ansatz in Braitenberg [1984]. Bei diesen sogenannten *Braitenberg-Vehikeln* handelt es sich um einfache Modellfahrzeuge mit Lichtsensoren und zugehörig gekoppelten Motoren. Die Sensorik empfängt dabei ein Lichtsignal (den sog. *Stimulus*) und gibt diesen mit wahlweise positivem oder negativem Vorzeichen direkt an die Aktorik weiter. Das Vehikel wird sich, abhängig von der gewählten Kopplung, von der Lichtquelle wegbewegen oder angezogen werden. Dieser Ansatz wurde in Braitenberg [1984] ursprünglich genutzt, um Muster der Psychologie zu veranschaulichen. Es wurden dabei jedoch außerdem die theoretischen Grundlagen für konfigurierbares reaktives Verhalten mechatronischer Robotiksysteme geschaffen.

Die verhaltensbasierte Architektur von Brooks [1986] (sog. *Subsumption*) realisiert erstmals softwaretechnische Verhalten in Form von Modulen, welche die gesamte Bandbreite vom Sensoreingang bis zur Steuerung der Aktorik abdecken. Dazu werden diese Module in eine funktionsorientierte Hierarchie eingebunden. Dabei werden auf höheren Modulebenen komplexere Funktionen, wie beispielsweise Pfadplanung, und auf unteren Ebenen grundlegendere Reaktionen auf Umgebungseinflüsse, z.B. Hindernisvermeidung, parallel ausgeführt. Die generierten Aktionen der jeweils unterlagerten Hierarchieebene haben hierbei stets die höhere Priorität, um die sichere Steuerung des Roboters zu gewährleisten. Solche Softwaremodule werden in Brooks [1986] erstmals Verhalten (engl. *behavior*) genannt. Da ein einzelnes Verhalten immer nur relativ wenig Funktionalität bietet, lassen sich solche Softwarekomponenten vergleichsweise schnell und einfach implementieren. Außerdem ist durch die Unabhängigkeit der einzelnen Module von den anderen Ebenen jeweils eine sehr schnelle Lösung und Generierung einer Aktion, demnach ein reaktives Verhalten, zu erwarten (vgl. Seite 6). Auf Basis der vorangegangenen Erkenntnisse lassen sich somit weitere Anforderungen an Steuerungsarchitekturen definieren.

Anforderung A4:

Auf dynamische Ereignisse aus der Umwelt des Systems soll unmittelbar rechtzeitig und robust reagiert werden.

Anforderung A5:

Steuerungsarchitekturen sollen aus Gründen der Zielorientiertheit sowie der Robustheit gegenüber möglichen Unsicherheiten aus der Umwelt(-modellierung) hierarchisches und paralleles Verhalten ermöglichen.

Anforderung A6:

Einzelne Verhalten sollen möglichst einfach und überschaubar gehalten werden, um eine transparente und schnelle Realisierung zu ermöglichen.

Bei den sogenannten (engl.) *Motor-Control-Schemas* aus Arkin [1987] bildet jedes Verhalten zugehörige Sensoreingänge direkt auf Matrizen von Ausgangsvektoren für die Motorik mobiler Systeme ab. So werden Perzeption und Motorik dynamisch verbunden. Die jeweils resultierenden Vektorfelder der einzelnen Verhalten für die Bewegung eines mobilen Roboters werden schließlich überlagert und ähnlich einer Potentialfeldmethode fusioniert. Das mobile System bewegt sich anschließend in die resultierende Richtung des geringsten Potentials. Aus diesem speziellen Ansatz ergibt sich eine zusätzliche Anforderung an verhaltensbasierte Steuerungen. Dieser Forderung wird in Abschnitt 2.1.2.3 ausführlich nachgegangen.

Anforderung A7:

Für parallele Verhalten einer reaktiven Steuerung wird eine geeignete Möglichkeit zur Koordinierung, also zur Fusionierung der Einzelverhalten, benötigt.

In Anderson und Donath [1990] werden grundlegende Verhaltensmuster für mobile Roboter, die sogenannten primitiven Verhalten (engl. *primitive*

behaviors), vorgestellt. Es wird dabei davon ausgegangen, dass ein Robotersystem vielfältige Aufgaben autonom und in unstrukturierten dynamischen Umgebungen ausführen soll. Dafür ist generell eine Vielzahl von verschiedenen Verhalten erforderlich. Daher wird in Anderson und Donath [1990] der bereits kurz besprochene hierarchische Ansatz gewählt und weiterentwickelt: Im Gegensatz zur Fusion vieler einfacher Verhalten zu einem resultierenden Stellbefehl (vgl. [Arkin, 1987]) wird eine vertikale Unterscheidung von Einzelverhalten mit verschiedenen Komplexitätsebenen vorgeschlagen. Dies führt zu einem konkurrierenden verhaltensbasierten System und schließlich zu einem resultierenden *emergenten* Verhalten des Roboters. Daher wird eine weitere Anforderung gestellt.

Anforderung A8:

Um ein breites Spektrum verschiedener und komplexer Aufgaben zu erfüllen, werden innerhalb einer verhaltenbasierten Steuerung Hierarchiestufen bzw. -schichten unterschiedlicher Komplexität benötigt.

In Anderson und Donath [1990] werden bereits verschiedene primitive jedoch unterschiedlich komplexe Verhalten für mobile Roboter vorgeschlagen und definiert (z.B. Hindernisvermeidung, Freiraum-Befahrung, Objektverfolgung). Der Begriff des **Primitivs** wird an dieser Stelle demnach erstmals definiert und ist *nicht* im Sinne von „einfach“ bzw. „simpel“ sondern als Orientierung eines Verhaltens in Richtung der Lösung *einer* konkreten Aufgabe zu interpretieren.

Der Ansatz der Hierarchieebenen wird auch in Behnke et al. [2000] bzw. Behnke und Rojas [2001] aufgegriffen. Dabei stellen die Autoren eine verhaltensbasierte Architektur vor, in welcher die Verhaltenshierarchie streng nach der Zykluszeit aufgebaut wird. Dafür werden die schnellen Verhalten in die unteren Ebenen eingeordnet. Dort befinden sich demnach vergleichsweise einfache reflexive Verhalten, welche kurzfristig auf dynamische Umweltbedingungen reagieren. Auf höheren Ebenen nehmen die Bearbeitungszeiten der beteiligten Verhalten zu. Dabei steigt in der Regel auch die Anzahl der involvierten Sensoren und Aktoren. In den hierarchisch überlagerten Ebenen werden demnach die Verhalten eingeordnet, welche eine höhere Komplexität besitzen und damit auch zunehmende Rechenzeiten aufweisen.

Von Lenser et al. [2002] wird eine auf Behnke et al. [2000] basierende hierarchische verhaltensbasierte Steuerungsarchitektur zur Ansteuerung von

Schreitrobotern vorgestellt, welche den Aspekt der Modularität von Verhalten in den Vordergrund rückt. Unter dem Begriff der Modularität wird die Fähigkeit des Systems zum flexiblen Hinzufügen, Entfernen und Austauschen von einzelnen Verhaltensmodulen zusammengefasst. Aufgrund ihrer Eigenschaft verschiedene Verhaltensmodule auf Ebenen verschiedener Komplexität bzw. Zeitbasis zusammenzufassen, wird die Architektur als hierarchisch bezeichnet. Anhand dieses flexiblen Ansatzes zur Realisierung des vorgestellten Systems können weitere allgemeingültige Anforderung für nutzerfreundliche Steuerungsarchitekturen identifiziert werden.

Anforderung A9:

Modularität soll innerhalb von Steuerungsarchitekturen eine nutzerfreundliche Austauschbarkeit und Wiederverwendung von Verhalten auch auf verschiedenen Hierarchieebenen ermöglichen, ohne dabei Einfluss auf andere in der Steuerung vorhandene Verhalten auszuüben. Eine Systemkopplung der einzelnen Module soll dabei über definierte Schnittstellen stattfinden.

Dem Problem eines fehlenden zentralen Weltmodells innerhalb der verhaltensbasierten Architekturen zur vorausschauenden Planung von Operationen wird in Nicolescu und Matarić [2002] versucht mit sogenannten abstrakten Verhalten zu begegnen. Dabei werden Roboteraktionen innerhalb einer klassischen hierarchischen verhaltensbasierten Steuerung auf Basis primitiver Verhalten ausgeführt. Zusätzlich eingeführte abstrakte Verhalten realisieren keine eigentliche Funktionalität, sondern speichern lediglich Zustände. Dabei können ganze Sequenzen von Bedingungen im System hinterlegt werden, um verschiedene primitive Verhalten zielorientiert aktivieren oder deaktivieren zu können. So können einmal im Voraus geplante Aktionssequenzen durch Strukturveränderungen der Architektur realisiert werden. Die Forderung nach geplanten Lösungsstrategien komplexer Aufgaben wird somit als zusätzliche Anforderungen aufgenommen.

Anforderung A10:

Die Steuerungsarchitektur soll komplexe Aufgaben bzw. Pläne in Form von Aktionssequenzen unterstützen.

In Proetzsch et al. [2005] bzw. Proetzsch et al. [2007] wird festgestellt, dass innerhalb der Literatur zu verhaltensbasierten Architekturen vor allem Aspekte zum Design der Anwendung und der Analyse der zu entwerfenden Steuerung nur unzureichend thematisiert werden. Die Realisierung von konkreten Anwendungen nach einer gegebenen Architektur wird oft nicht transparent und meist nicht allgemeingültig dargestellt. Auf Basis dieser Problemstellung wird von den Autoren die *iB2C*-Architektur (*integrated Behavior-Based Control*) vorgestellt. Dabei handelt es sich um einen rein verhaltensbasierten Designansatz mit vorgefertigten Verhaltens- und Fusionsmodulen. Mit diesem Ansatz kann eine Steuerung für einfache Aufgaben schnell anhand vorgefertigter Module konfiguriert werden. Für komplexere Anwendungen werden die resultierenden Modelle jedoch schnell unübersichtlich und schwer modifizierbar. Dennoch können auf Basis dieser Erkenntnisse weitere Anforderungen formuliert werden. Weiterführende Aspekte dazu werden im Abschnitt 2.3 behandelt.

Anforderung A11:

Steuerungsarchitekturen sollen möglichst allgemein aufgebaut sein, um eine Vielzahl von Applikationen zu ermöglichen.

Anforderung A12:

Die Nutzung von vorgegebenen *Templates* in Form von definierten Software-Bausteinen erleichtert dem Nutzer die Realisierung konkreter Verhalten.

Verhaltensbasierte Steuerungen reagieren in erster Linie direkt und unmittelbar auf Sensorinformationen. Daher ist es nicht oder nur eingeschränkt möglich, langfristiges Verhalten zu realisieren. Aufgaben und Pläne mit mehreren Schritten bis zur Zielerfüllung sind ohne Erweiterungen generell nicht umsetzbar. Daher sind solche Systeme nur eingeschränkt nutzbar und für komplexere Aufgaben ungeeignet. Da zusätzlich ohne Weltmodelle und Repräsentationen gearbeitet wird, sind optimale Lösungen nicht zu erwarten.

Vielmehr können verhaltenbasierte Steuerungen zur Laufzeit eine Anzahl widersprüchlicher Lösungen erzeugen. Dies kann die Robustheit des Systems gegenüber Unsicherheiten aus der Umwelt sogar erhöhen, verhindert aber planvolles Agieren innerhalb der vorgegebenen Aufgabenstruktur. Demnach bieten verhaltensbasierte Architekturen den Vorteil, Steuerungen reaktiv und robust zu gestalten, gekoppelt mit dem Nachteil der unzuverlässigen oder eingeschränkten Erfüllung komplexerer Aufgaben. Dies führt zur Entstehung des hybriden Ansatzes aus planungsfähiger funktionsorientierter und reaktiver verhaltensbasierter Architektur, welcher in Abschnitt 2.1.3 vorgestellt wird.

2.1.2.2 Realisierung von Verhalten

Verhalten sind letztendlich Softwareblöcke, die anhand einer gegebenen Eingangsinformation (*Stimulus*) einen Ausgangswert (*Response*) berechnen (vgl. [Arkin, 1998], [Siciliano und Khatib, 2008]). Innerhalb eines Verhaltens sind demnach Regeln zur Verarbeitung dieser Daten realisiert. Ein allgemeines funktionsorientiertes Modell eines Verhaltens $\mathcal{V}_i$ wird in Abbildung 2.3 dargestellt. Als Eingangsdaten kommen sowohl Sensorwerte als auch Ausgangsdaten vorgeschalteter Verhalten in Betracht. Ausgangsdaten können entsprechend direkt an die Aktorik gesendet werden oder für nachfolgende Verhalten zur Verfügung stehen. Hierbei können verschiedene Verhalten gleiche Eingangsinformationen nutzen oder auch gleiche Ausgangsinformationen erzeugen. Außerdem sollten neue Verhalten jederzeit in eine bestehende Steuerung eingefügt werden können, ohne dabei die gesamte Steuerungsstruktur ändern zu müssen. Dies gilt auch für die Entfernung nicht benötigter Verhalten. Alle in einer hierarchischen Ebene aktivierten Verhalten werden stets funktionell parallel ausgeführt, um konkurrierendes und somit gegenüber Unsicherheiten aus der Umwelt(-modellierung) robustes Verhalten zu ermöglichen.

Die eigentliche Aufgabe und Implementierung eines Verhaltens ist natürlich system- und anwendungsabhängig. Daher existiert ein breites Spektrum unterschiedlichster Verhaltensansätze. So gibt es beispielsweise reine Bewegungsverhalten für mobile Systeme (z.B. [Larson und Voyles, 2001]) oder Schreitroboter (z.B. [Denk und Schmidt, 2003]), lernende Verhalten zur Robotersteuerung über genetische Ansätze (z.B. [Lee et al., 1997]), *Fuzzy*-basierte Ansätze (z.B. [Vadakkepat et al., 2004]) und auch Verhalten zur Beeinflussung der Motivation mobiler Roboter (z.B. [Manzotti und Tagliasco, 2005]). Viele Ansätze sind durch ihren hohen Spezialisierungsgrad gekennzeichnet und arbeiten innerhalb einer eng eingeschränkten Domäne. Sie verletzen dabei

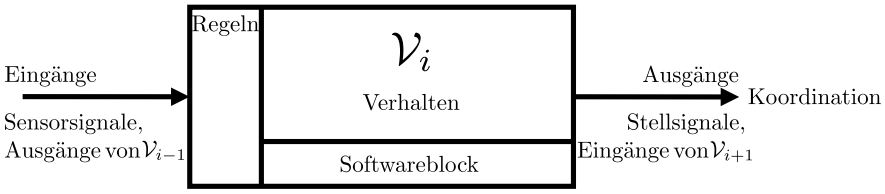


Abbildung 2.3: Im funktionsorientierten Modell eines allgemeinen Verhaltens wird eine spezifische Funktionalität durch Verarbeitung von Eingangsdaten anhand vorgegebener Regeln realisiert. Eingangsinformationen sind dabei Sensorinformationen oder Ausgangsdaten vorgeschalteter Verhalten. Für Systeme der Robotik bzw. der robotergestützten Fertigungstechnik wird ein Verhalten durch einen Softwareblock beschrieben.

jedoch die oben aufgestellte Anforderung A11. Generell sind der Komplexität innerhalb eines Verhaltens keine Grenzen gesetzt. Entwurfsziel ist jedoch in der Regel die Realisierung vergleichsweise einfacher und softwaretechnisch übersichtlicher Verhalten. Diese Idee wird mittels eines in dieser Arbeit genutzten neuen Ansatzes konsequent verfolgt. Dabei wird Funktionalität innerhalb von Verhalten über standardisierte und überschaubare Softwarebausteine realisiert. Der zugrunde liegende Ansatz dazu wird im Abschnitt 2.2 ausführlich beschrieben.

2.1.2.3 Koordination paralleler Verhalten

Bereits durch Pirjanian [1999] werden verschiedene Mechanismen der Koordination paralleler Verhalten beschrieben. Unter dem Begriff **Koordination** wird dabei die Erzeugung einer resultierenden Systemantwort aus der Fülle der Antworten der jeweiligen Verhalten innerhalb der Steuerung verstanden. Dabei wird versucht, Widersprüche zwischen Verhalten aufzulösen, um eine Lösung der Systemaufgabe zu ermöglichen. Hierbei wird zwischen konkurrierenden und kooperierenden Ansätzen unterschieden (vgl. [Arkin, 1998]).

Innerhalb der **konkurrierenden Ansätze** werden Konflikte, die zwischen Verhalten mit gleichen Ausgangsinformationen entstehen, gelöst, indem letztendlich nur ein Verhalten zur Ausführung kommt. Dies kann, wie beispielsweise bei Brooks [1986], prioritätsbasiert geschehen. Dabei erhält jedes Verhalten einen statischen Prioritätswert, der oft von der Hierarchieebene abhängt. Des Weiteren existieren konkurrierende Ansätze mit dynamischen Aktivitätsvariablen. Diese werden, wie beispielsweise bei Maes [1990], zur Laufzeit auf Basis aktueller Daten und Aufgaben berechnet. Anschließend

kommt das Verhalten mit der höchsten Aktivität zur Ausführung.

Die **kooperierenden Ansätze** sind dadurch gekennzeichnet, dass die Ausgangsdaten der parallel aktiven Verhalten fusioniert werden. Somit werden die Antworten aller benötigten Verhalten an der resultierenden Systemantwort beteiligt. Dieses Vorgehen kann Vorteile hinsichtlich der Robustheit gegenüber Unsicherheiten bewirken, wenn die einzelnen Verhalten so realisiert sind, dass nicht nur ausschließlich konträre Ausgangsinformationen berechnet werden. Dies kann beispielsweise, wie bei Badreddin [1994] oder Mai und Janschek [2009], innerhalb eines *Fuzzy-Logic*-Ansatzes über eine *Fuzzifizierung* der Ausgangsdaten erreicht werden. Ein weiterer möglicher Ansatz ist die Fusion der Ausgangsdaten über eine verhaltensspezifische Wichtung. Dieses auch in der vorliegenden Arbeit gewählte Vorgehen wird in Abschnitt 3.2.4 näher erläutert.

Zahlreiche weitere Ansätze zur Koordination paralleler Verhalten existieren bereits in der Literatur. Daher ist eine architektonische Offenheit der genutzten Steuerungsarchitektur gegenüber dem gewählten Prinzip wünschenswert. So lassen sich verschiedene Verfahren anwendungsabhängig austauschen.

2.1.3 Hybride Architekturen

Aus den beiden oben vorgestellten Ansätzen für Steuerungsarchitekturen von Robotiksystemen lässt sich eine dritte Klasse ableiten: Die hybriden Architekturen. Nach Gat [1997] wird dabei der reaktive Charakter der verhaltensbasierten Steuerungen mit einer funktionsorientierten Planungskomponente in einer Architektur vereint. Dafür werden hierarchisch aufgebaute Subsysteme, sogenannte Schichten, mit nach oben zunehmender Komplexität bzw. sich erhöhendem Abstraktionsgrad erstellt. Die einzelnen Schichten interagieren hierbei untereinander, wobei die oben gelegenen Schichten die komplexere Anwendungsplanung übernehmen, während die unteren Schichten die reaktiven Verhaltensweisen (vgl. Seite 6) sicherstellen. Wie in Abbildung 2.4 dargestellt werden somit nur bestimmte aufgabenabhängige Verhalten innerhalb der verhaltensbasierten Schicht durch Module höherer Abstraktionsgrade aktiviert. Die eingeführten Hierarchien bleiben erhalten, jedoch werden relevante Informationen ebenenübergreifend genutzt. Dies resultiert in einem integrierenden Modul zur Planung, Überwachung und Aktionsgenerierung auf oberster Ebene. Durch die Kombination der Vorteile beider Ansätze werden nun Aufgaben mit langfristigen Zielen bei erhaltener Reaktivität realisierbar.

Das Ansatz der hybriden Architekturen wurde durch Firby [1987] bzw. Firby und Slack [1995] eingeführt. Dabei wird zwischen zwei unterschiedlichen

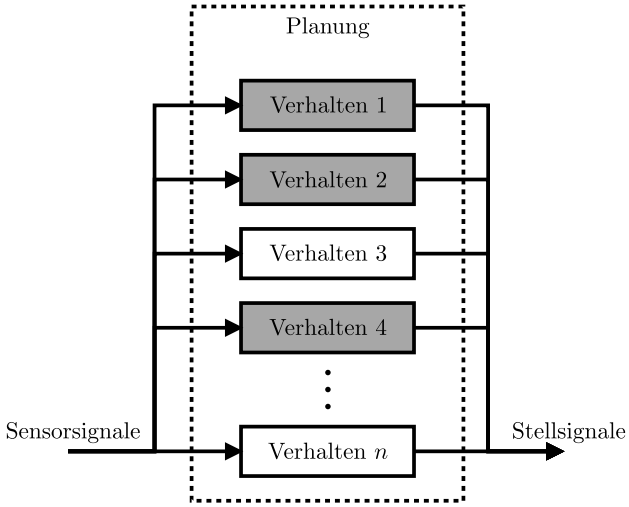


Abbildung 2.4: In hybriden Architekturen werden parallel arbeitende Verhalten durch hierarchisch überlagerte Module aufgabenabhängig aktiviert. Dabei werden Sensorinformationen durch die jeweils aktiven Verhalten in Handlungsanweisungen überführt (nach [Siciliano und Khatib, 2008], [Nehmzow, 2003]).

Schichten differenziert. Eine verhaltensbasierte Schicht realisiert Aktionen eines Robotersystems, während eine überlagerte Schicht die Verhalten in Form einer Zustandsmaschine aktiviert bzw. deaktiviert. In Volpe et al. [2002] bzw. Nesnas et al. [2003] wird eine ähnliche zweischichtige Architektur vorgestellt, wobei hierbei der Fokus auf der Wiederverwendbarkeit der Verhalten liegt. Dazu werden diese möglichst abstrakt und unabhängig von der später zu nutzenden Hardware angelegt.

Aufbauend darauf wurde durch Bonasso [1991] eine hybride Architektur basierend auf drei Schichten vorgestellt. Eine reaktive Schicht realisiert dabei die Systemantworten auf Sensorinformationen auf unterster Ebene. Sie ermöglicht eine schnelle Systemreaktion auf Umweltereignisse. In der Planungsschicht wird die globale Strategie zur Aufgabenerfüllung abgearbeitet und überwacht. Auf Basis eines Weltmodells werden Verhaltenssequenzen initialisiert. Eine Ausführungs- bzw. Sequenzschicht dient als Verbindungsschicht. Anweisungen aus der hierarchisch überlagerten Schicht werden hier in Aktionen für die reaktive Schicht sequenziert und entsprechend aktiviert.

Mehrere Architekturen nutzen diesen dreischichtigen Ansatz (engl. *3T, 3-Tier*). Dabei übernimmt die mittlere Schicht jeweils die Funktion der Vermittlung zwischen aufgabenabhängiger Planung und Aktivierung der dafür benötigten Verhalten. So wird beispielsweise in Alami et al. [1998] eine

dreischichtige Architektur mit heterogen implementierten Ebenen eingeführt, wobei für jede Ebene eigene Softwareentwicklungswerkzeuge bereitgestellt werden. In Li et al. [2004] wird schließlich eine Multi-Schichten-Architektur zur Steuerung mobiler Roboter in *Indoor*-Umgebungen eingeführt. Dabei wird die *Middleware* CORBA als Vermittlungstechnologie über Schnittstellen zwischen den einzelnen Schichten eingesetzt. So können viele Schichten unterschiedlicher Komplexität und Hierarchieebenen eingebunden werden. Da das System beliebig skaliert, können auch beliebig komplexe Aufgaben realisiert werden. Für Änderungen von Aufgaben wird dabei jedoch ein besonders tiefes Systemverständnis aller beteiligten Module vorrausgesetzt, da einzelne Schichten untereinander in Abhängigkeit stehen.

In Martens et al. [2007] bzw. Prenzel et al. [2012] wird die hybride Steuerungsarchitektur *MASSiVE* vorgestellt. Dabei handelt es sich um einen klassischen hybriden Ansatz mit verschiedenen hierarchischen Ebenen innerhalb der Steuerungsarchitektur. Ziel ist es, eine Gesamtaufgabe aus dem Gebiet der Rehabilitationsrobotik in eine Sequenz von Aufgaben zu zerlegen. Für jede Aufgabe stehen nun, abhängig von der aktuellen Konstellation von Objekten in der Roboterumgebung, sogenannte *Skills* zur Verfügung. Jeder *Skill* aktiviert entsprechende reaktive Softwareblöcke auf einer unterlagerten verhaltensbasierten Ebene und löst damit die gestellte Systemaufgabe. Dieser aufgabenorientierte Ansatz dekomponiert Roboteraufgaben jedoch vorrangig sequentiell und unterstützt keine nebenläufigen Teilprozesse und Synchronisationen auf höheren Ebenen. Auf unterster Ebene sind redundante (parallel arbeitende) Verhalten vorgesehen, es wird derzeit aber noch nicht klar gezeigt, wie diese geeignet koordiniert werden können.

Innerhalb aller hybriden Architekturen ist es möglich, die vorhandenen Schichten bei Bedarf mit zusätzlichen Schichten zu erweitern. So wird es beispielsweise möglich, Anwender über eine Benutzerschnittstellenschicht oder mehrere Roboter über eine Multi-Roboter-Schicht einzubinden.

Hybride Architekturen sind praktisch Stand der Technik und werden, aufgrund der Robustheit gegenüber Unsicherheiten aus der Umwelt(-modellierung) sowie der Möglichkeit langfristige Ziele verfolgen zu können, aktuell vielfach eingesetzt. Die genannten Vorteile relativieren sich jedoch aus Entwicklersicht schnell. Die gesteigerte Komplexität der Architektur an sich führt zu einem verkomplizierten Engineering der konkreten Steuerung. Dabei werden dem Entwickler in den wenigsten Fällen praktische Leitlinien und Vorgehensmodelle mitgegeben. Außerdem stellen viele der vorhandenen hybriden Steuerungen Insellösungen für jeweils eine spezifische Problemkategorie dar. Allgemeine Ansätze, die weitgehend universell eingesetzt werden können,

sowie konkrete Methoden zur Anwendungsrealisierung fehlen bislang.

2.2 Funktionsrealisierung über Aktionsprimitive

Im Abschnitt 2.1.2.2 wurde beschrieben, dass es aus Entwicklersicht günstig ist, wenn Verhalten möglichst standardisiert aufgebaut werden und dabei einfach editier- und austauschbar sind. Eine konsequente Umsetzung dieser Anforderungen wird mit dem Ansatz der Aktionsprimitive möglich. Die Grundidee dabei ist die Nutzung von modularen vereinheitlichten Programmbausteinen zur Realisierung genau **einer** und damit **primitiven** Funktionalität innerhalb eines Verhaltens. Diese Funktionalität löst eine gestellte Aufgabe durch die Ausführung einer **Aktion**. Unter einem Aktionsprimitiv (engl. *action primitive* oder *skill primitive*) wird daher ein klar definiertes Softwaremodul zur Lösung dieser Aufgabe verstanden. Ein Aktionsprimitiv ist damit ein eigenständiges und unabhängiges Modul, welches autonom sensorgeführt und sensorüberwacht eine Minimalaktion ausführt. Dabei werden solche Primitive in erster Linie als strukturelle Paradigmen betrachtet und sind damit unabhängig von der konkret gewählten Steuerungsarchitektur. Systemaufgaben werden dazu in eine Menge von Minimalaktionen zerlegt, um die Gesamtaufgabe mittels aller im System hinterlegten Primitive zu lösen. Dieses Vorgehen sowie die Verwendung von Aktionsprimitiven allgemein werden innerhalb des folgenden Abschnitts ausführlich erläutert.

2.2.1 Übersicht über die Nutzung von Aktionsprimitiven

Das Konzept der Aktionsprimitive wurde ursprünglich in der Industrierobotik eingeführt. In Hasegawa et al. [1990] und Hasegawa et al. [1992] werden erstmals sogenannte Manipulationsprimitive vorgestellt, welche als sensorgeführte Bewegungsmakros definiert sind. Eine Folge von Bewegungen eines Industrieroboters wird dabei durch eine Sequenz von Primitiven abgebildet. Montageaufgaben können daher in eine Anzahl grundlegender Aktionen zerlegt werden. Dazu wird ein Primitiv-Ablaufplan aus einem Montageplan manuell erzeugt und als Standardsequenz im System hinterlegt. Eine Steuerung aktiviert nun zur Laufzeit sensorabhängig das jeweils passende Primitiv. Typische Manipulationsprimitive sind hier beispielsweise einfache Bewegungen von Objekten im Arbeitsraum durch einen Roboterarm.

Die Verwendung des Ansatzes über Aktionsprimitive wird auch von Finkemeyer [2004] bzw. Finkemeyer et al. [2005] im Bereich der Industrierobotik zur Ansteuerung von Manipulatoren für Montagearbeiten vorgeschlagen. Mithilfe

der Steuerung werden dabei Führungsgrößen für Bewegungsabläufe berechnet und in Form von Parametern innerhalb der Aktionsprimitive hinterlegt. Anschließend wird die Bewegung sensorgestützt ausgeführt. Eine boolesche Abbruchbedingung legt dafür das Ende einer Aktion respektive Bewegung formal fest. Dazu wird eine Menge von verfügbaren Sensorsignalen auf boolesche Werte abgebildet. Mit der Abbruchbedingung wird somit der zu erreichende Zielzustand ebenso festgelegt wie mögliche Fehler- oder Alternativzustände. Aktionen und Abbruchbedingungen sind innerhalb der Aktionsprimitive implementiert. Die Steuerung kann dabei auf eine vorhandene Bibliothek zugreifen, in welcher alle vorhandenen Aktionsprimitive gespeichert vorliegen. Aus Montageplänen wird nun ein Netz von Aktionsprimitiven erstellt, durch dessen kontinuierliche Abarbeitung die aktuelle Aufgabe erfüllt werden kann. Dieser resultierende Plan entspricht demnach einem Zustandsautomaten. Jedes Aktionsprimitiv wird somit einem diskreten Zustand des Systems zugeordnet. Diese Architektur wurde durch Reisinger [2008] erweitert, um Parallelroboter zu betreiben. Um auch Kontaktregelungskonzepte nutzen zu können, wurden dafür spezielle Aktionsprimitive realisiert.

Bei Nagatani und Tanaka [2000] werden Aktionsprimitive erstmals als grundlegende Bewegungsabläufe für mobile Roboter verwendet. Die Gesamtbewegung, die zur Bewältigung einer Aufgabe nötig ist, wird daher ebenfalls durch die Aneinanderreihung von mehreren Aktionsprimitiven zu einer Sequenz realisiert. Zielpunkte in der Umgebung des Roboters werden dabei zu einem Netz verbunden. Aus der Menge der möglichen Aktionsprimitive werden anschließend jene ausgewählt, welche den Roboter durch das Netz navigieren. Dabei wird jedem Aktionsprimitiv eine Menge an Parametern, beispielsweise Zielkoordinaten, sowie ein Folgeprimitiv zugewiesen. Diese Folgeprimitive werden abhängig vom Status des vorangegangenen Aktionsprimitivs aktiviert.

In Milighetti et al. [2005] bzw. Milighetti und Kuntze [2006] wird das mobile Anwendungsszenario noch erweitert. Es wird eine Aktionsprimitiv-basierte Steuerungsarchitektur für mobile Serviceroboter vorgestellt. Dafür werden Aktionsprimitive in Form sogenannter Grundgeschicklichkeiten verwendet. Hierbei werden Pläne in Form vorprogrammierter Listen von Aktionsprimitiven im System hinterlegt. Die Ausführung solcher Listen führt von einer aktuellen Situation zur Zielsituation. Aktuelle Aufgaben werden daher durch einen Zustandsautomaten sequentiell gelöst. Demnach wird die gesamte Roboteraufgabe vom Anwender in eine dynamisch konfigurierbare Folge von Aktionen zerlegt. Ein zusätzlicher Entscheidungsfindungsalgorithmus ermittelt dabei jedoch situationsabhängig das jeweils nächste zu aktivierende Primitiv. Dazu werden die Aktionsprimitive um *fuzzy*-basierte Bewertungs-

funktionen zur Feststellung von Eignung und Verfügbarkeit des jeweils auszuwählenden Primitivs erweitert (vgl. [Milighetti et al., 2006]). Die Entwicklung und Optimierung solcher Systeme wird anhand von PETRI-Netzen vorgenommen. Jede Roboteraufgabe resultiert demnach in einem PETRI-Netz von Aktionsprimitiven, welches die Zustände des Roboters repräsentiert. Hierfür werden jedoch solche Netze lediglich als Zustandsmaschinen ohne Nebenläufigkeiten verwendet. Es kann demnach zu einem Zeitpunkt auch immer nur eine Roboteraufgabe bearbeitet werden.

Ein ähnlicher Ansatz wird durch Kim et al. [2005] vorgestellt. Verschiedene Navigationsprimitive werden für die Bewegung eines mobilen Roboters im Raum genutzt. Der Fokus dieser Arbeit liegt auf der Auswahl des Primitivs, welches zur Laufzeit die aktuelle Aufgabe innerhalb einer dynamischen Umgebung bestmöglich erfüllt. Dazu wird eine Abschätzung der Güte auf der Basis einer Evaluierung vorangegangener Testläufe des Roboters anhand zeitbewerteter PETRI-Netze in Form einfacher Zustandsmaschinen vorgenommen.

Wie gezeigt, lassen sich Roboteraufgaben intuitiv in eine Anzahl von Minimalaktionen zerlegen. Die Anzahl und Art der involvierten Sensoren sowie der berechneten Ausgangsinformationen, welche einem Aktionsprimitiv zugeordnet werden, ist dabei variabel. Dadurch ergibt sich eine flexible Steuerungsstruktur. Die Ausführung einer Aktion mittels Aktionsprimitiven ist dabei stets **sensorgeführt** und deren Ende über eine Abbruchbedingung **sensorüberwacht**. Bei der Verwendung von Aktionsprimitiven ergeben sich demnach einige Vorteile für die Steuerung:

- Die flexible Implementierung von verschiedenen Aufgaben in Form von minimalen Aktionen ist vergleichsweise aufwandsarm, komfortabel erweiterbar und aus Entwicklersicht transparent. Neue Funktionalität lässt sich über Aktionsprimitive schnell realisieren und leicht in die bestehende Steuerungsstruktur einbinden.

- Eine Wiederverwendbarkeit von einzelnen Primitiven oder Teilen ganzer Aktionsprimitiv-Steuerungen für verschiedene Aufgaben ist möglich. Aktionsprimitive können dabei durch gezielte Kombination zu neuen aufgabenspezifischen Systemen verknüpft werden.

- Durch die Verwendung von standardisierten Aktionsprimitiven steht ein wohldefiniertes Interface für verschiedene Robotersteuerungen zur Verfügung.

Der viel versprechende Ansatz, Verhalten in Form von Aktionsprimitiven zu realisieren, wird darum als Forderung aufgenommen.

Anforderung A13:

Um Funktionalität über formalisierte Verhalten in Form von Minimalaktionen zu realisieren und eine übersichtlich strukturierte und komfortabel modifizierbare Steuerungsarchitektur zu ermöglichen, soll der Aktionsprimitiv-Ansatz genutzt werden.

2.2.2 Formalisierung des Konzepts

Um eine verbal formulierte Aktion in eine für das Softwaresystem verständliche Syntax zu überführen, wird ein Formalismus zur Beschreibung von Aktionsprimitiven benötigt. In der Literatur, speziell in Finkemeyer [2004], wird ein Aktionsprimitiv dabei durch

$$\mathcal{AP} := \{\mathcal{HM}, \tau, \lambda\} \tag{2.1}$$

definiert. Ein Aktionsprimitiv bildet somit ein 3-Tupel (bzw. Tripel) bestehend aus den folgenden Komponenten:

- Mit der hybriden Bewegung $\mathcal{HM}$ (*Hybrid Move* bzw. *Hybrid Motion*) wird die konkrete Bewegung eines Robotersystems beschrieben. Der hybride Charakter beschreibt dabei die Anwendung unterschiedlicher Regelungskonzepte, welche durch ein Aktionsprimitiv aktiviert werden können. Der Begriff stammt ursprünglich aus der Industrierobotik und wird im Rahmen dieser Arbeit verallgemeinert. Dieses Vorgehen wird in Kapitel 3.2.2 beschrieben.

- Bestimmte Systemaufgaben können mit spezieller Hardware verknüpft sein. Durch τ wird daher eine Menge von sogenannten Werkzeugkommandos definiert. Hier wird die zur Laufzeit benötigte zusätzliche Hardware zur Ausführung eines Aktionsprimitivs festgelegt. So werden beispielsweise Greifer als Endeffektor von Roboterarmen angesteuert.

- Aktionsprimitive werden immer mit dem Erreichen eines parametrierten Zielzustandes beendet. Dieser wird durch die Abbruchbedingungen λ definiert. Dabei wird zwischen Standard-Abbruchbedingungen, wie beispielsweise einem *Timeout* oder einem Sensorfehler, und den aufgabenspezifischen Abbruchbedingungen, beispielsweise dem Erreichen einer Zielposition, unterschieden.

Nach dem Auslösen der Abbruchbedingung gibt ein Aktionsprimitiv stets einen Rückgabewert zurück. Anhand dieses Wertes kann der Endzustand eines durchlaufenen Aktionsprimitivs ermittelt und darüber auf das zu aktivierende Folgeprimitiv geschlossen werden.

2.3 Engineering von Robotersteuerungen

Wie oben gezeigt, existiert für mechatronische Systeme der Robotik sowie der robotergestützten Fertigungstechnik eine fast undurchschaubare Anzahl verschiedener Steuerungsarchitekturen. Dabei liegt der Fokus der vorgestellten Architekturen jedoch meist nur auf der Art, wie Funktionalität innerhalb der Steuerung bereitgestellt und verarbeitet wird. Beim Architekturentwurf ist aber noch eine zweite Entwurfsaufgabe zu lösen. Neben dem reinen strukturellen Steuerungsaufbau kommt dem Aspekt des *Engineering zur Anwendungsrealisierung* eine vielfach vernachlässigte jedoch entscheidende Bedeutung zu. Damit wird das Vorgehen zur Realisierung einer konkreten Aufgabe innerhalb der vorliegenden Steuerungsarchitektur beschrieben. Dabei sind nach Schönfeld [1998] Aspekte zur Übersichtlichkeit, Wartbarkeit und Wiederverwendbarkeit zu beachten. Eine bereits vorliegende Steuerungsarchitektur soll demnach den Entwickler von konkreten Anwendungen unterstützen. Eine geeignete transparente Entwurfsrichtlinie bzw. -beschreibung des gesamten Prozesses zur Anwendungserstellung ist somit Grundvoraussetzung für ein schnelles und komfortables Engineering.

Solche Vorgehensmodelle zum Engineering werden bei den vorhandenen Steuerungsarchitekturen in der Regel vernachlässigt. Meist erschließt sich die Anwendungsrealisierung nur dem Anwender, welcher über Spezialwissen über bestimmte Programmiersprachen oder spezielle Softwaresysteme verfügt. Das liegt vorrangig auf dem Fokus der oben vorgestellten Steuerungsarchitekturen auf eine spezifische Problemklasse. Resultat sind vielfach Insellösungen, die sich nur schwer oder überhaupt nicht in eine andere Anwendungsdomäne übertragen lassen. Dabei werden nach Bruyninckx [2002] vor allem potentielle Kleinseriennutzer solcher Systeme (z.B. kleine mittelständische Unternehmen)

eingeschränkt. Sie verfügen einerseits nur über eine beschränkte Anzahl umfassend ausgebildeter Entwickler und andererseits nicht über ausreichend Mittel zur externen Auftragsentwicklung von Anwendungslösungen. Als Antwort auf diese Probleme wird daher eine weitere Anforderung identifiziert.

Anforderung A14:

Der Engineeringprozess der Steuerungsarchitektur zur Realisierung von Anwendungen ist übersichtlich, transparent und in Form eines eindeutigen Vorgehensmodells zum Entwurf zu formulieren.

In Siciliano und Khatib [2008] werden hierbei die zu durchlaufenden Vorgehensschritte im Rahmen des Engineering genannt. Innerhalb einer Aufgabenanalyse werden dazu in einem ersten Schritt die Anforderungen an die Funktionalität der Steuerung aufgestellt. Anschließend werden zugehörige Aktionen zum Lösen der identifizierten Aufgaben definiert. Dabei entsteht ein Datenmodell unter Einbezug vorhandener Sensorik sowie Aktorik. Eine zeitliche Bezugsetzung legt die minimale Zykluszeiten sowie notwendige Verarbeitungskapazitäten der Steuerung fest. Darauf folgend werden mögliche Nutzer benannt und Anwendungsfälle erstellt. Auf Basis dieser Informationen erfolgt der Steuerungsentwurf mittels eines anschließenden Analyseverfahrens. Idealerweise ist der resultierende Entwurf von der zu wählenden Programmiersprache unabhängig.

Anforderung A15:

Die im Entwurfsprozess entstandene Struktur der Steuerungsarchitektur ist idealerweise unabhängig von einer konkreten Programmiersprache.

Ein eindeutiges Vorgehen zur Anwendungsrealisierung wird somit neben der Realisierung einer flexiblen und robusten Steuerung als zweites Entwurfsziel gestellt.

2.4 Offene Probleme und eigener Beitrag

In den vorangegangenen Abschnitten wurden wichtige Anforderungen an Steuerungsarchitekturen für Systeme der Robotik sowie der robotergestützten Fertigungstechnik zusammengestellt. Dabei wurden vordergründig zwei Hauptaspekte identifiziert: Die genutzte Steuerung soll sowohl komplexere Aufgaben mit kurz- und langfristigen Zielen bearbeiten und lösen können als auch adaptiv und reaktiv auf Umgebungseinflüsse reagieren können. Zusätzlich ist ein umfassender Engineeringansatz notwendig, der den Anwender von der Erstellung der Steuerung zur Realisierung einer Anwendung bis zur Modifikation derselben unterstützt.

Auf Basis der Literaturanalyse liefern erweiterte verhaltensbasierte bzw. hybride Architekturen einen geeigneten Rahmen, um flexible und gegenüber Unsicherheiten robuste Steuerungen zu erstellen. Dabei wurde jedoch auch deutlich, dass derzeit vorrangig Insellösungen für spezielle Systeme bzw. Systemklassen existieren. Zusätzlich wird der Engineeringaspekt vielfach vernachlässigt, so dass mögliche Anwender Expertenwissen über das gesteuerte System benötigen oder es sich aneignen müssen. Allgemeingültige Vorgehensmodelle und Methoden zur Steuerungserstellung fehlen oder sind nicht transparent. Die Abwesenheit von Wissensträgern einer speziellen Steuerung führt daher schnell zur problembehafteten Modifizier- und Erweiterbarkeit und somit zu eingeschränkter Nutzbarkeit.

Aktionsprimitive stellen eine transparente und gut überschaubare intuitive Möglichkeit dar, um Funktionalität innerhalb eines Systems bereitzustellen. Dabei sind, wie oben angeführt, Steuerungsarchitekturen mit Aktionsprimitiven bisher nur als Zustandsautomaten realisiert worden. Zu jedem Zeitpunkt ist jeweils nur ein Aktionsprimitiv aktiv, wobei Problemstellungen durch die sequentielle Bearbeitung verschiedener Primitive gelöst werden.

Innerhalb der folgenden Abschnitte wird gezeigt, wie die beschriebenen Konzepte innerhalb eines *integrierten Ansatzes* in Form einer neuen Steuerungsarchitektur fusioniert werden können. Dabei werden *parallel arbeitende Aktionsprimitive* innerhalb einer *hybriden Architektur* genutzt und mittels eines *speziellen Engineeringansatzes* bereitgestellt. Daher wird zunächst die Fragestellung geklärt, wie eine solche Architektur strukturell konzipiert und anschließend realisiert werden kann. Als geeignete Struktur wird dazu eine Trennung zwischen einem funktionsorientierten verhaltensbasierten Modell zur Ausführung von sowohl funktionell parallel als auch hierarchisch arbeitenden Aktionsprimitiven und einem ablauforientierten Modell zur aufgabenabhängigen Konfiguration derselben vorgestellt. Mit einer solchen Struktur entsteht

ein regelungstechnisch-orientiertes kaskadiertes Modell der Steuerung, in welcher Modifikationen transparent, übersichtlich und gegenüber herkömmlichen Steuerungen vergleichsweise einfach vorgenommen werden können.

Diese entworfene Steuerungsstruktur wird zusammen mit allen oben identifizierten Anforderungen innerhalb einer Architektur umgesetzt. Dafür wird ein transparenter Ansatz zur Realisierung von komplexen und parallelen Aufgaben bei gleichzeitiger übersichtlicher Strukturierung und Handhabbarkeit innerhalb der Steuerung entworfen. Hierfür wird ein ganzheitlicher Engineeringansatz vorgestellt, welcher mögliche Nutzer von der System- und Aufgabenanalyse über die Steuerungserstellung bis zum Systemstart und der Modifikation der erstellten Steuerung begleitet.

Der vorgestellte Entwurf wird dabei so ausgelegt, dass auch mehrere voneinander unabhängige Teilsysteme innerhalb einer Steuerung erstellt werden können. Über spezielle Mechanismen können diese Teilsysteme anschließend auf Aktionsprimitivebene synchronisiert werden. Damit wird eine Steuerung von verteilten Aufgaben möglich. Aufgrund des universellen Entwurfs können mit der Steuerungsarchitektur anschließend sowohl Systeme der mobilen Robotik als auch der robotergestützten Fertigungstechnik realisiert werden.

Obwohl die in den nächsten Kapiteln beschriebene Steuerungsarchitektur den Entwurf von funktionsfähigen Steuerungen durch klare Ablaufmodelle innerhalb einer hierarchischen Systemsstruktur unterstützt, ist es nicht Grundanliegen strukturelle sowie funktionelle Korrektheit der Steuerung (Stabilität, maximale Reaktionszeiten, semantische Widerspruchsfreiheit) in jedem Fall sicherzustellen, sondern vielmehr einen Rahmen für transparentes Enginneering solcher Steuerungen innerhalb eines überschaubaren *Frameworks* zur Realisierung von Anwendungslösungen für verschiedene Szenarien zu schaffen.

Der detaillierte Entwurf der Steuerungsarchitektur auf der Basis der oben identifizierten Anforderungen wird im nächsten Kapitel beschrieben. Anschließend wird die entstandene Architektur durch einen objektorientierten Softwareentwurf in eine konkrete Implementierung überführt. Darauf aufbauend wird der Architektur-eigene Engineeringansatz ausführlich anhand von Beispielen erläutert. Auf dieser Basis werden schließlich verschiedene Demonstrationsszenarien umgesetzt und evaluiert, um Vorteile der entstandenen Architektur bezüglich Aufgabenerfüllung und Anwenderfreundlichkeit hervorzuheben.