Lehrstuhl für
Werkzeugmaschinen und Fertigungstechnik
der Technischen Universität München

Methodische Entwicklung störungstoleranter Steuerungen

Khalid-Alexander Sabbah

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. rer. nat. H. Bubb

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. G. Reinhart
2. Univ.-Prof. Dr. rer. nat. Dr. rer. nat. habil. M. Broy

Die Dissertation wurde am 28.10.1999 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 23.02.2000 angenommen.

Forschungsberichte iwb

Band 139

Khalid-Alexander Sabbah

Methodische Entwicklung störungstoleranter Steuerungen

herausgegeben von
Prof. Dr.-Ing. G. Reinhart

Herbert Utz Verlag UTZ

Forschungsberichte iwb

Berichte aus dem Institut für Werkzeugmaschinen
und Betriebswissenschaften
der Technischen Universität München

herausgegeben von

Univ.-Prof. Dr.-Ing. Gunther Reinhart
Technische Universität München
Institut für Werkzeugmaschinen und Betriebswissenschaften (iwb)

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist
bei Der Deutschen Bibliothek erhältlich

Zugleich: Dissertation, München, Techn. Univ., 2000

ISBN 3-89675-739-3

Printed in Germany

Herbert Utz Verlag GmbH, München
Tel.: 089/277791-00 · Fax: 089/277791-01

Geleitwort des Herausgebers

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 und 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 Produktentwicklung über die Planung von Produktionssystemen hin zu 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

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Werkzeugmaschinen und Betriebswissenschaften (*iwb*) der Technischen Universität München.

Herrn Prof. Dr.-Ing. Gunther Reinhart, den Leiter dieses Instituts, gilt mein besonderer Dank für die wohlwollende Förderung und großzügige Unterstützung meiner Arbeit.

Bei Herrn Prof. Dr. rer. nat. habil. Manfred Broy, dem Leiter des Instituts für Informatik der Technischen Universität München, möchte ich mich für die Übernahme des Korreferates und die aufmerksame Durchsicht der Arbeit sehr herzlich bedanken, bei Prof. Dr. rer. nat. Heiner Bubb für die Übernahmen des Vorsitzes.

Darüberhinaus bedanke ich mich bei allen Mitarbeiterinnen und Mitarbeitern des Instituts sowie allen Studenten, die mich bei der Erstellung meiner Arbeit unterstützt haben, recht herzlich.

München, im Februar 2000 *Khalid-Alexander Sabbah*

Inhaltsverzeichnis Seite

1 Einleitung 1
1.1 Ausgangssituation 1
1.2 Ziel der Arbeit 3
1.3 Vorgehensweise und Ergebnisse 4
2 Grundlagen störungstoleranter Steuerungen 6
2.1 Übersicht 6
2.2 Steuerung von Produktionssystemen 6
2.3 Steuerungsarten 7
2.3.1 NC-Steuerungen 9
2.3.2 RC-Steuerungen 10
2.3.3 SPS 10
2.4 Steuerungsentwicklung 11
2.4.1 Genormte Programmiersprachen 12
2.4.1.1 Kontaktplan (KOP) 12
2.4.1.2 Funktionbausteinsprache (FBS) 13
2.4.1.3 Ablaufsprache (AS) 13
2.4.2 Beschreibungstechniken 13
2.4.2.1 Zustandsautomaten bzw. Zustandsgraphen 15
2.4.2.2 Petri-Netze 16
2.4.2.3 Kontrollstrukturen 18
2.4.2.4 Unified Modeling Language (UML) 18
2.5 Störungen in Produktionssystemen 20
2.5.1 Definition des Begriffes „Störung“ 20
2.5.2 Störungsauswirkung 21
2.5.3 Störungsursachen 22
2.5.4 Störungsursachenbereiche 22
2.6 Störungsbehandlung 23
2.6.1 Phasen der Störungsbehandlung 23

2.6.2 Verfahren zur Störungserkennung und Störungslokalisierung 24
2.7 Zusammenfassung 26
3 Anforderungsanalyse 27
3.1 Übersicht 27
3.2 Anforderungen an das Referenzkonzept 27
3.2.1 Anforderungen aus der Betriebsphase 28
3.2.2 Anforderungen aus der Entwicklungsphase 30
3.3 Anforderungen an das methodische Vorgehen 31
3.4 Zusammenfassung 32
4 Bestehende Systeme und Ansätze zur rechnergeführten Störungsbehandlung 33
4.1 Übersicht 33
4.2 Ansätze zur reinen Störungserkennung 33
4.3 Ansätze zur Störungserkennung und Störungslokalisierung 35
4.3.1 Ansätze für einzelne Ursachenbereiche 35
4.3.2 Ansätze für komplette Produktionssysteme 36
4.4 Ansätze mit anschließender Störungsbehebung 37
4.5 Bewertung der Ansätze 39
4.6 Zusammenfassung 40
5 Referenzkonzept der störungstoleranten Steuerung 41
5.1 Übersicht 41
5.2 Die Steuerungsfunktionalität 42
5.2.1 Gliederung in autarke Bausteine 42
5.2.2 Steuerung mittels diskreter Zustände 43
5.2.3 Idle- und Running-Zustände 45
5.3 Die Störungsbehandlungsfunktionalität 46
5.3.1 Voraussetzungen im Rahmen der Störungsbehandlung 46
5.3.2 Störungszustand 47
5.3.3 Störungserkennung 48
5.3.3.1 Zeitüberwachungsmethode 48
5.3.3.2 Signalüberwachungsmethode 50
5.3.3.3 Komponentenüberwachungsmethode 51

5.3.3.4 Weitere Ansätze 52
5.3.4 Störungslokalisierung 52
5.3.4.1 Lokalisierung durch Symptom-Ursachen-Beziehung 53
5.3.4.2 Lokalisierung durch Kombinatorik 54
5.3.4.3 Lokalisierung durch schrittweise Eingrenzung 55
5.3.4.4 Auswahl des geeigneten Verfahrens 56
5.3.4.5 Detaillierungsgrad der Ursachenfindung 56
5.3.5 Störungsbehebung 57
5.3.5.1 Methoden zur Störungsbehebung 57
5.3.5.2 Reversibilität von Abläufen 60
5.3.5.3 Die Reaktionsstrategie 61
5.3.5.4 Einbettung der Reaktionsstrategie in eine zustandsorientierte Steuerungsanweisung 65
5.3.5.5 Auswahl der Reaktionsstrategie 66
5.4 Der störungstolerante Steuerungsbaustein 67
5.4.1 Aufbau 67
5.4.2 Bausteininterner Störungsbehandlungsablauf 70
5.4.3 Bausteinübergreifender Störungsbehandlungsablauf 71
5.5 Zusammenfassung 72
6 Methodisches Vorgehen zur Entwicklung störungstoleranter Steuerungen 74
6.1 Übersicht 74
6.2 Beschreibungstechnik 75
6.2.1 Übersicht 75
6.2.2 Störungsbehandlungstabellen 75
6.2.2.1 Ursachenfindungstabellen 76
6.2.2.2 Behebungstabellen 77
6.2.3 Modellelemente und -diagramme 78
6.2.3.1 Die störungstolerante Steuerungsklasse 78
6.2.3.2 Der STB-Zustandsgraph 80
6.2.3.3 Das STB-Aktivitätsdiagramm 84
6.3 Vorgehensweise 86

6.3.1 Übersicht 86
6.3.2 Modellierung einer störungstoleranten Anwendung 87
6.3.2.1 Identifikation der Steuerungsklassen 88
6.3.2.2 Grobspezifikation 90
6.3.2.3 Feinspezifikation 97
6.3.2.4 Instantiierung und Verknüpfung einer kompletten Anwendung 101
6.3.3 Implementierung einer störungstoleranten Anwendung 101
6.3.4 Zusammenfassung 103
7 Prototypische Realisierung und beispielhafter Einsatz 104
7.1 Übersicht 104
7.2 Realisiertes Entwicklungswerkzeug 104
7.2.1 Entwicklungsumgebung 105
7.2.2 Laufzeitmodule 107
7.3 Einsatzbeispiel 108
7.3.1 Aufbau der Versuchsmaschine 108
7.3.2 Modellierung der Steuerungsaufgabe 109
7.3.3 Modellierung der Störungsbehandlungsfunktionalität 112
7.3.3.1 Störung beim Ausfahren des Greifarms 112
7.3.3.2 Störung beim Greifen des Werkzeuges 114
8 Bewertung 117
9 Zusammenfassung und Ausblick 124
10 Literatur 127

Abkürzungsverzeichnis

A	Aktion im Aktivitätsdiagramm
AC	Axis Control
AS	Ablaufsprache
B/E-Netze	Bedingungs-/Ereignisnetze
BMBF	Bundes Ministerium für Bildung und Forschung
CASE	Computer Aided Software Engineering
EU	Europäische Union
FB	Fehlerbaum
FBS	Funktionsbausteinsprache
H	Definierter Halte-Zustand
I1	Anfänglicher Idle-Zustand in einem Dienst
I2	Endgültiger Idle-Zustand in einem Dienst
IEC	International Electrotechnical Commission
IPC	Industrie-PC
KOP	Kontaktplan
LC	Logic Control
MC	Motion Control
MMI	Man Machine Interface
NC	Numerical Control
OMT	Object Modeling Technique
OO	Objektorientiert
OOAD	Object Orientied Analysis and Design
PC	Personal Computer
RC	Robot Control
RS	Reaktionsstrategie
RZ	Running-Zustand

S	Gestörter Zustand
S/T-Netze	Stellen-/Transitionsnetze
SPS	Speicherprogrammierbare Steuerung
ST	Strukturierter Text
STB	Störungsbehandelnd
UML	Unified Modeling Language
WAP	Wiederaufsetzpunkt
WEP	Wiedereinstiegspunkt

1 Einleitung

1.1 Ausgangssituation

Sowohl wirtschaftliche als auch technologische Faktoren haben dazu beigetragen, daß die Automatisierung in der heutigen Produktion stark zugenommen hat. Infolge der Sättigung lokaler Märkte einerseits und der zunehmenden Globalisierung andererseits sind jedoch weitere Herausforderungen an die Produktion hinzugekommen. So führen die gesättigten Märkte zur verstärkten Senkung von Stückzahlen bei gleichzeitiger Steigerung der Variantenvielfalt und Komplexität von Produkten. Andererseits fordert der verschärfte Wettbewerb kürzere Entwicklungs- und Produktionszeiten sowie eine höhere Produktivität und Qualität. Komplexität und Individualität einerseits und die Forderung nach Produktivität und Qualität andererseits machen eine flexible Automatisierung in der Produktion erforderlich *(Milberg 1997)*. Vor dem Hintergrund hoher Flexibilität in der Produktion hat zunächst die Funktionalität, aber auch die Komplexität der hierfür eingesetzten Produktionssysteme stark zugenommen.

Untersuchungen zur Verfügbarkeit von Produktionssystemen zeigen jedoch auf, daß mit zunehmender Komplexität die Verfügbarkeit der Systeme sinkt. Dies bestätigt beispielsweise die Studie von *Milberg und Ebner (1994),* die im Auftrag des Vereins Deutscher Werkzeugmaschinenfabriken (VDW) zum Thema „Verfügbarkeit von Werkzeugmaschinen“ durchgeführt wurde. Hier betrug der durchschnittliche Zeitanteil technisch begründeter Stillstände bei einfachen Drehmaschinen im Durchschnitt 3,5 %, während er bei komplexeren Bearbeitungszentren im Durchschnitt auf ca. 7 % stieg. Bei 15 % der Bearbeitungszellen lag er sogar über 10 %. Angaben über die Abnahme der Verfügbarkeit bei zunehmender Komplexität gehen auch aus *Kallfass (1999)* und *Pfob (1998)* hervor. Besonders bei Transferstraßen und verketteten Anlagen, wo durch den Ausfall einer einzelnen Maschine die gesamte Anlage zum Stillstand kommen kann, können in ungünstigen Fällen sehr hohe Produktionsausfälle entstehen.

Wagner (1997, S. 3) begründet des Phänomen der sinkenden Verfügbarkeit bei erhöhter Komplexität zum einen mit der steigenden Ausfallhäufigkeit eines Gesamtsystems bei zunehmender Anzahl der verketteten Komponenten, wie dies das Boolsche Modell der *VDI-Richtlinie 4008 Blatt 2 (1986)* beschreibt. Zum anderen führt der erhöhte Zeitaufwand zur Lokalisierung und Behebung von Störungen aufgrund mangelnder Übersichtlichkeit und der verstärkten Forderung nach hochqualifiziertem Instandsetzungspersonal zu langen Ausfallzeiten.

Um die Verfügbarkeit der komplexen Produktionssysteme zu erhöhen, können unterschiedliche Maßnahmen ergriffen werden. So liegt es nahe, in erster Linie die Komplexität der Produktionssysteme zu reduzieren. Derartige Maßnahmen stoßen

jedoch besonders dann an Grenzen, wenn die geforderte Funktionalität und Flexibilität beeinträchtigt wird. Falls die Komplexität nicht reduziert werden kann, müssen Werkzeuge zur Beherrschung der Komplexität entwickelt werden. In technischen Systemen kann dies mit Hilfe spezieller, qualitätsregelnder Steuerungswerkzeuge erfolgen *(Reinhart 1994, S. 192)*. Hierzu gehören Werkzeuge, die mittels umfassender Störungsbehandlungsfunktionalität die technisch bedingten Stillstandszeiten reduzieren.

Derartige störungsbehandelnde Werkzeuge können in der automatisierten Produktion auf Leit-, Zellen- oder Maschinenebene eingesetzt werden. Nach *Milberg und Ebner (1994, S. 97)* sind die meisten technisch begründeten Stillstandszeiten auf Störungen in der Maschinen- und Anlagenperipherie, wie Werkstück- und Werkzeugwechseleinrichtungen oder Handhabungssysteme, zurückzuführen. Die Ursachen hierfür sind meist defekte oder dejustierte Schalter und Geber sowie Defekte an Leitungen und Kabeln. Folglich liegt es nahe, die störungsbehandelnden Ansätze für technische Störungen auf Maschinenebene zu legen, denn besonders dort kann unmittelbar auf die für die Störungsbehandlung notwendigen Informationen zurückgegriffen und für schnelle Reaktionszeiten garantiert werden *(Schönecker 1992, S. 19)*.

Zur Behandlung von Störungen auf Maschinenebene wurden bereits zahlreiche Ansätze und Systeme entwickelt. Viele Ansätze sind Systeme, die in einem Online-Betrieb die Störungen erfassen und im Offline-Betrieb den Bediener bei der Störungslokalisierung bzw. Ursachenfindung unterstützen, wie dies beispielsweise bei Expertensystemen der Fall ist. Die meisten Ansätze sind jedoch Systeme, die Störungen erfassen, dies dem Bediener melden und in Abhängigkeit von der Schwere der Störung den laufenden Betrieb abbrechen *(Koch & Köhne 1995).* Nur selten erfolgt hier eine umfassende Störungsbehandlung, die eine detaillierte Störungsbehebung einschließlich des Wiederanlaufs des normalen Betriebs beinhaltet.

Gerade hier aber liegt ein wesentliches Potential zur Reduzierung der technisch bedingten Stillstandszeiten vor. So geht aus dem Bericht von *Milberg & Ebner (1994, S. 97)* hervor, daß 70 % der anfallenden Störungen Kleinstörungen sind, die eine Reparaturdauer von weniger als 30 Minuten benötigen. 40 Prozent können sogar in weniger als 15 Minuten behoben werden. Meistens sind nur einfache Einstellarbeiten nötig, die bei entsprechender Anleitung auch vom Maschinenbediener durchgeführt werden können. Anschließend kann mit dem Betrieb fortgefahren werden. Laut *Fähnrich (1990, S. 24–26)* würde auch die verstärkte Einbeziehung des Maschinenbedieners in den Instandsetzungsprozeß die benötigte Stillstandszeit erheblich reduzieren, da ein Großteil dieser Zeit durch das Warten auf den Instandsetzer entsteht.

Aus rein technischer Sicht gesehen sind die heutigen Steuerungen ausreichend leistungsfähig, um zusätzliche Funktionen und Algorithmen zu implementieren, die die Störungsbehebung einschließlich des Wiederanlaufs des Betriebes unterstützen. Dies wird durch vereinzelte Steuerungsansätze aus der Forschung bestätigt, die die komplette Wirkungskette der Störungsbehandlung berücksichtigen. Dennoch werden derartige Ansätze nur selten in der Industrie eingesetzt, da sie in der Regel viel zu komplex sind. Erst wenn die Komplexität dieser Systeme abnimmt, rentiert es sich, störungstolerante Produktionssysteme zu entwickeln.

Gleichzeitig hat aber die Komplexität der Software von Produktionssystemen insgesamt zugenommen, was in erster Linie auf die zunehmenden Anforderungen an die Softwarefunktionalität zurückzuführen ist *(Reinhart u. a. 1998).* Zur Bewältigung dieser Komplexität und des damit verbundenen Aufwandes wurden unterschiedliche werkzeugunterstützte Methoden entwickelt. Für die Entwicklung von Steuerungssoftware ohne Störungsbehandlung gibt es beispielsweise werkzeugunterstützte Methoden auf Basis von Petri-Netzen oder Zustandsgraphen, die den Entwicklungsprozeß vereinfachen und die Produktivität erhöhen *(Weck 1995, S. 108–127).* Anders ist dies aber im Zusammenhang mit Software zur rechnergeführten Störungsbehandlung. Hier stehen primär nur rudimentäre Zusatzwerkzeuge zur Verfügung, mit denen auf Implementierungsebene Diagnosefunktionalität in bestehende Steuerungsmodule hinzugefügt werden kann, ohne ein integraler Bestandteil des Gesamtsystems zu sein *(Koch 1996, S. 25–27).* Zudem fehlen Methoden, die zu einem systematischen Vorgehen bei der Entwicklung störungstoleranter Anwendungen verhelfen und entsprechend den Entwicklungsprozeß effektiver gestalten.

1.2 Ziel der Arbeit

Aus den bestehenden Defiziten im Zusammenhang mit der Entwicklung störungstoleranter Systeme leitet sich der Handlungsbedarf und somit das Ziel dieser Arbeit ab. Ziel ist es, eine Methode zur produktiven Entwicklung störungstoleranter Steuerungen, die alle Phasen der Störungsbehandlung unterstützen, zu erarbeiten. Dies soll zum einen durch die Reduktion der Komplexität von störungstoleranten Steuerungen und zum anderen durch die Erhöhung der Effektivität des Entwicklungsvorgangs erfolgen.

Vor diesem Hintergrund soll in der vorliegenden Arbeit ein vereinfachtes Steuerungskonzept entwickelt werden, das als Referenzkonzept für störungstolerante Steuerungen dienen kann. Dieses Konzept unterstützt die Störungserkennung und -lokalisierung sowie die Störungsbehebung einschließlich des Wiederanlaufs des normalen Betriebs. Basierend auf diesem Steuerungskonzept soll ein methodisches Vorgehen beschrieben werden, das mit Hilfe moderner

Beschreibungstechnik und einer effizienten Werkzeugunterstützung die Entwicklung störungstoleranter Steuerungsanwendungen vereinfacht und effektiver gestaltet.

1.3 Vorgehensweise und Ergebnisse

Um die oben beschriebenen Ziele zu erreichen, wird in dieser Arbeit wie folgt vorgegangen (Abbildung 1-1):

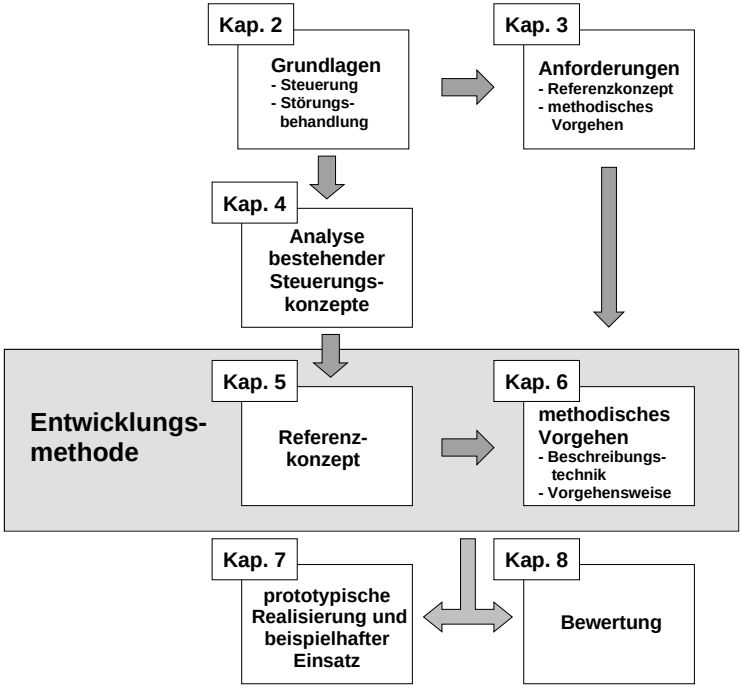


Abbildung 1-1: Vorgehen im Rahmen der Arbeit

Zunächst werden im Kapitel 2 die wesentlichen Grundlagen zur Steuerung und Störungsbehandlung in Produktionssystemen betrachtet. Dabei werden die Grundlagen zur reinen Steuerung, aber auch wichtige Aspekte im Zusammenhang mit Störungen und Störungsbehandlung ermittelt.

Anschließend erfolgt in Kapitel 3 eine umfassende Anforderungsanalyse. Hierbei werden sowohl die Anforderungen an ein Referenzkonzept einer störungstoleranten Steuerung als auch die Anforderungen an ein methodisches Vorgehen spezifiziert.

Basierend auf den Anforderungen an das Referenzkonzept werden in Kapitel 4 bestehende Konzepte zur rechnergeführten Störungsbehandlung hinsichtlich ihrer Eignung als Referenzkonzept analysiert und bewertet.

Danach erläutert Kapitel 5 das hier zu entwickelnde Referenzkonzept einer störungstoleranten Steuerung. Dabei wird zunächst aufgezeigt, wie Steuerungsfunktionalität ohne Störungsbehandlung in einzelne Module gegliedert werden kann. Anschließend wird beschrieben, wie die Störungserkennung, Störungslokalisierung und Störungsbehebung in den einzelnen Modulen erfolgt.

Zur Entwicklung einer konkreten störungstoleranten Anwendung, die entsprechend dem Referenzkonzept aufgebaut ist, wird in Kapitel 6 ein methodisches Vorgehen vorgestellt. Dazu wird zunächst eine graphische Beschreibungstechnik präsentiert, die als Hilfsmittel während des methodischen Vorgehens dient. Anschließend erfolgt die Beschreibung einer Vorgehensweise zur werkzeugunterstützten Entwicklung einer störungstoleranten Anwendung.

In Kapitel 7 wird ein beispielhafter Einsatz einer prototypisch realisierten störungstoleranten Steuerung vorgestellt, die nach dem Referenzkonzept aufgebaut ist. Abschließend erfolgt in Kapitel 8 eine Bewertung der hier erarbeiteten Methode zur produktiven Entwicklung störungstoleranter Steuerungen.

2 Grundlagen störungstoleranter Steuerungen

2.1 Übersicht

In diesem Kapitel werden die wesentlichen Grundlagen beschrieben, die im Zusammenhang mit störungstoleranten Steuerungen in Produktionssystemen relevant sind und zum Grundverständnis dieser Arbeit benötigt werden. So erfolgt zunächst eine Untersuchung der Steuerungsebenen und Steuerungsarten, die in Produktionssystemen wesentlich sind. Anschließend wird der Aspekt der methodischen Entwicklung dieser Steuerungen detailliert untersucht. Außerdem beschreibt dieses Kapitel die wesentlichen Grundkenntnisse zu Störungen in Produktionssystemen. Abschließend werden die wesentlichen Verfahren zur Störungserkennung und -lokalisierung erläutert.

2.2 Steuerung von Produktionssystemen

Nach *Glas (1993, S. 7)* sind Produktionssysteme ein Oberbegriff für Anlagen der Produktionsbereiche „mechanische Teilfertigung“ und „Montage“. Flexible Produktionssysteme können flexible Fertigungssysteme oder flexible Montagesysteme sein. Ein flexibles Fertigungssystem ist ein aus einer oder mehreren Arbeitsmaschinen bestehendes System, das über ein gemeinsames Steuer- und Transportsystem verknüpft ist. Durch diese Verknüpfung kann eine automatisierte Fertigung stattfinden, und zudem können verschiedene Bearbeitungsaufgaben an unterschiedlichen Werkstücken durchgeführt werden *(Kief 1997, S. 405).* Analog zu dieser Definition können für den Bereich der Montage flexible Montagesysteme definiert werden *(Schmidt 1991, S. 10).*

In der rechnergeführten Produktion kann die Steuerung eines Produktionssystems auf fünf Informationsebenen erfolgen *(ISO TC 184, 1986).* Zu unterscheiden sind die Planungs-, Leit-, Zellen-, Steuerungs- und Aktor/Sensorebene. Aufgrund der Tatsache, daß die meisten technisch begründeten Stillstandszeiten auf Störungen in der Maschinen- und Anlagenperipherie zurückgeführt werden können, sind die hier zu entwickelnden störungstoleranten Steuerungen in die Steuerungsebene der ISO TC 184 einzuordnen (Abbildung 2-1).

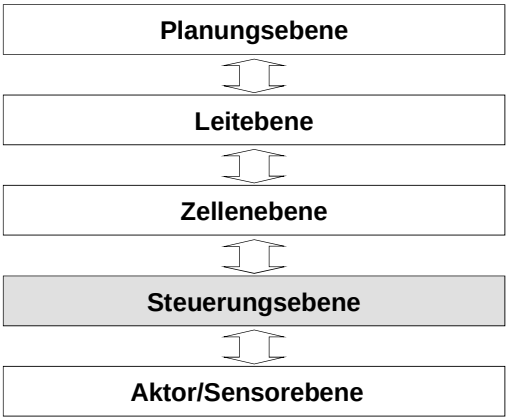


Abbildung 2-1: Steuerungsebenen nach ISO TC 184 (1986)

2.3 Steuerungsarten

Grundsätzlich können auf Steuerungsebene mechanische und elektrische Steuerungen unterschieden werden. Erstere steuern Abläufe mit Hilfe von mechanischen Nocken, Kurven und Hebeln. Letztere funktionieren auf Basis elektrischer bzw. elektronischer Bauelemente *(Weck 1995, S. 24–45, 90–138)*. Aufgrund der rasanten Entwicklung im Bereich der Mikroelektronik, insbesondere bei den Mikroprozessoren, und der Zunahme der Leistungsfähigkeit der elektronischen Bauteile bei gleichzeitiger Abnahme von Herstellungskosten werden heutzutage vorwiegend elektrische Steuerungen eingesetzt.

Nach *DIN 19237 (1980)* und *DIN 19226 Teil 5 (1994)* können elektrische Steuerungen nach unterschiedlichen Gesichtspunkten, wie z. B. dem Funktionsprinzip oder der Programmierbarkeit gegliedert werden. Bei einer Gliederung nach dem Funktionsprinzip werden **Verknüpfungs**- und **Ablaufsteuerungen** unterschieden (Abbildung 2-2).

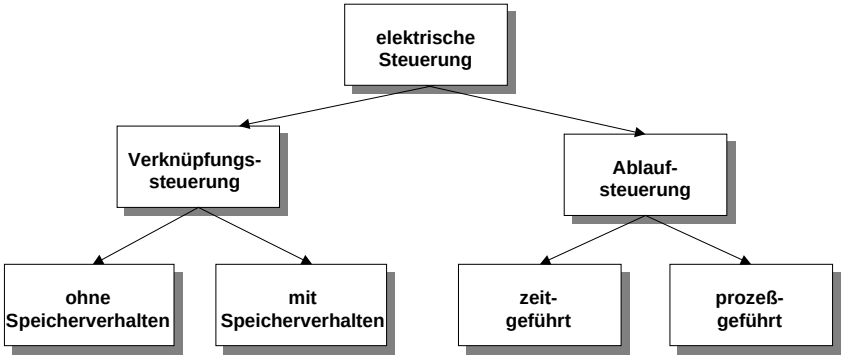


Abbildung 2-2: Einteilung von Steuerungen nach dem Funktionsprinzip

Bei einer Verknüpfungssteuerung handelt es sich um eine Steuerung, die den Signalzuständen der Eingangssignale bestimmte Signalzustände der Ausgangssignale im Sinne Boolscher Verknüpfungen zuordnet. Ein Beispiel hierfür ist eine Steuerung für die Drehrichtung eines Motors. Dabei gibt es Steuerungen mit und ohne Speicherverhalten. Bei ersteren hängt der Ausgangszustand ausschließlich vom Eingangszustand ab, während bei Verknüpfungssteuerungen mit Speicherverhalten auch der aktuelle Zustand des zu steuernden Systems noch eine Rolle spielt.

Ablaufsteuerungen sind hingegen Steuerungen mit zwangsläufig schrittweisem Ablauf. Das Weiterschalten von einem auf den programmgemäß folgenden Schritt erfolgt in Abhängigkeit von Weiterschaltbedingungen, wie dies z. B. bei Transport- oder Handhabungsvorgängen der Fall ist. Zu unterscheiden sind zeitgeführte und prozeßgeführte Ablaufsteuerungen. Bei zeitgeführten Steuerungen erfolgt das Weiterschalten ausschließlich in Abhängigkeit von der Zeit, bei prozeßgeführten Ablaufsteuerungen erfolgt das Weiterschalten aufgrund des Eintretens eines definierten Ereignisses, wie z. B. das Erreichen einer definierten Endlage. Nach *Weck (1995, S.94)* werden in Produktionssystemen vorwiegend Ablaufsteuerungen eingesetzt.

Aus Sicht der Programmierbarkeit von Steuerungen werden **verbindungsprogrammierbare** und **speicherprogrammierbare** Steuerungen unterschieden *(DIN 19226 Teil 5 1994).*

Verbindungsprogrammierbare Steuerungen sind auf der Basis von Relais, Schützen oder elektronischen Funktionsbausteinen aufgebaut. Die Steuerungsaufgabe wird durch die feste Verdrahtung der Hardware realisiert. Derartige Steuerungen werden vorwiegend bei sicherheitskritischen Anwendungen eingesetzt.

Im Vergleich zu verbindungsprogrammierbaren Steuerungen arbeiten speicherprogrammierbare Steuerungen auf der Basis elektronischer Speicherbausteine. Hier wird die Steuerungsaufgabe in Form eines Steuerungsprogrammes, der sogenannten Steuerungssoftware, bestimmt. Dadurch ist die Hardware der Steuerung unabhängig von der eigentlichen Steuerungsaufgabe. Aufgrund dieser hohen Flexibilität und der geringen Kosten werden bei sicherheitsunkritischen Anwendungen vorwiegend speicherprogrammierbare Steuerungen eingesetzt.

Zusätzlich zu den oben beschriebenen Einteilungen lassen sich die Steuerungen in die drei Gruppen NC-Steuerung und RC-Steuerung und SPS gliedern *(Barth u. a. 1988)*. Diese drei Gruppen sollen im folgenden näher erläutert werden.

2.3.1 NC-Steuerungen

NC-Steuerungen gehören, global betrachtet, zu den Ablaufsteuerungen und basieren auf der Mikroprozessortechnik. Sie dienen der numerischen Bewegungskoordination einer oder mehrerer Maschinenachsen und führen die für die Punkt-, Strecken- oder Bahnbewegungen notwendigen Interpolations- und Regelungsaufgaben aus. Die aktuellen Positionen der gesteuerten Achsen werden mit Hilfe von in den Achsen integrierten Winkel- und Positionsgebern erfaßt und an die Regler der NC-Steuerung zurückgeliefert. Je nach Ausbaustufe können NC-Steuerungen auch Zusatzmodule integrieren, um die in der Maschine anfallenden Schaltfunktionen auszuführen. Diese Module verfügen dann über einen eigenen Mikroprozessor *(Kief 1997, S. 55ff.)*.

Zur Beschreibung der Steuerungsaufgabe dient das mittels alphanumerischer Zeichen aufgebaute NC-Programm. Dieses enthält die Bewegungsabläufe der Maschinenachsen und beschreibt Art und Reihenfolge von Fertigungsschritten eines Werkstückes.

Aufgrund der zunehmenden Leistungsfähigkeit moderner IPC-Karten[1] werden derartige Karten vermehrt in NC-Steuerungen eingesetzt. Auf diesen Karten laufen die benötigten Steuerungsmodule, wie z. B. Interpolator- oder Interpretermodule in Form von Softwareprogrammen *(Weck 1995, S. 150)*.

Zusätzlich zu diesen hardwaretechnischen Trends zeichnen sich in NC-Steuerungen Entwicklungen ab, die unter den Begriff **offene Steuerung** fallen *(Weck 1995, S. 169–178)*. Auch hier ist die grundlegende Motivation die Reduktion von Entwicklungskosten und -zeiten. Kennzeichnend für offene Steuerungen ist die Modularisierung und Standardisierung von Steuerungssoftware sowie die Offenlegung von internen und externen Schnittstellen. Vor diesem Hintergrund wurde im Rahmen des von der EU finanzierten ESPRIT-Projektes „OSACA“ *(Pritschow*

[1] IPC: Industrie-PC

1996) eine Steuerungsplattform definiert, die zur Entkopplung von Steuerungshardware und Anwendungssoftware eine herstellerunabhängige Kommunikationsschicht spezifiziert (Abbildung 2-3). Zudem wurde in diesem Projekt eine grundlegende Referenzarchitektur für eine offene Steuerung beschrieben, die die elementaren Applikationsmodule einer Steuerung festlegt. Aufbauend auf diesen Ergebnissen wurde diese Referenzarchitektur im Rahmen des BMBF-Projekts „HÜMNOS“ *(Diesch u. a. 1997)* weiter spezifiziert und zahlreiche Steuerungsmodule prototypisch realisiert.

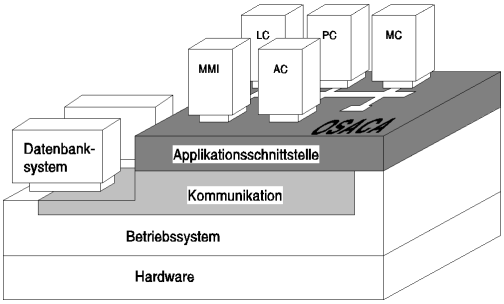


Abbildung 2-3: OSACA-Referenzsteuerung

2.3.2 RC-Steuerungen

RC-Steuerungen (Robot Control) dienen auch der numerischen Bewegungskoordination und ähneln in ihrem Aufbau den NC-Steuerungen. Bei RC-Steuerungen werden jedoch sehr hohe Anforderungen an die Berechnungsalgorithmen für Bewegungsbahnen und Koordinatentransformationen gestellt, da die Roboterachsen oftmals komplizierte kinematische Ketten bilden *(Kief 1997, S. 431–450).* Zur Beschreibung der Steuerungsaufgabe dient hier, ebenso wie bei dem NC-Programm, das sogenannte RC-Programm.

2.3.3 SPS

Der Begriff SPS steht für speicherprogrammierbare Steuerung. SPSen sind Geräte, die vorwiegend für binäre Schaltfunktionen eingesetzt werden. Hardwaretechnisch bestehen sie aus einer Zentraleinheit mit einem oder mehreren Prozessoren, einem Programmspeicher sowie zahlreichen Eingangs- und Ausgangseinheiten *(Kief 1997, 249–273).* An den Eingängen sind meist binäre Signalgeber, wie z. B. induktive

Näherungsschalter oder Bedientaster, angebracht, während die Ausgänge mit Stellgliedern hydraulischer oder pneumatischer Aktoren verbunden sind. Steuerungen werden vorwiegend für binäre Schaltfunktionen eingesetzt. Ein Überblick über bestehende Systeme geben *Klinker (1995)* und *Grötsch & Seubert (1997, S. 115–124).*

Zur Verarbeitung der Steuerungsprogramme werden in SPSen Universalprozessoren oder Spezialprozessoren eingesetzt. Mit der zunehmenden Leitungsfähigkeit der Universalprozessoren finden diese immer mehr Einsatz in der Steuerungstechnik. Eine besondere Form von SPS stellen die sogenannten Soft-PLCs dar. Das sind Steuerungen, die auf Basis von Industrie-PCs arbeiten und mit Hilfe von Echtzeitbetriebssystemen und Zusatzprogrammen die Funktionalität von SPS-Modulen softwaretechnisch ersetzen. Vorteilhaft an solchen PC-basierten Lösungen ist die Verwendung von Standardkomponenten sowie deren Offenheit *(SPS-Magazin 1997).*

Zur Programmierung bzw. Projektierung der SPSen werden sogenannte Programmiergeräte eingesetzt. Auf die eingesetzten Programmiersprachen wird in einem späteren Kapitel eingegangen.

Sowohl die SPSen als auch die Programmiergeräte sind herstellerspezifisch und somit nicht miteinander kompatibel. Aus diesem Grund gibt es neuerdings zahlreiche Normen und Initiativen zur Standardisierung von Programmiersprachen und Schnittstellen. Grundlegende Motivation dieser Entwicklung ist die Reduktion der hohen Entwicklungskosten und -zeiten von Steuerungssystemen, die in der Regel auf die technologischen Speziallösungen und die Inkompatibilität der Systeme zurückzuführen sind. Beispiele für solche Bestrebungen sind die internationale Norm *IEC 1131 Teil 3 (1994)*, die Firmenvereinigung „PLCopen“ *(PLCopen 1994)* sowie die Prozeßschnittstelle OPC *(Start-Magazin 1998).*

Da in erster Linie SPSen zur Steuerung der Schaltfunktionalität der Maschinen- und Anlagenperipherie eingesetzt werden, liegt das Hauptaugenmerk bei der Betrachtung der hier zu entwickelnden störungstoleranten Steuerungen auf der Steuerungsgruppe der SPSen.

2.4 Steuerungsentwicklung

Wie bereits beschrieben, handelt es sich bei den meisten Steuerungen, die in der Produktionstechnik eingesetzt werden, um programmierbare Steuerungen. Diese benötigen zur Durchführung der Steuerungsaufgaben Steuerungsanweisungen in Form von Softwareprogrammen. Diese Programme werden anhand spezieller Programmiersprachen erstellt und anschließend mit Hilfe prozessorspezifischer Compiler in den sogenannten Maschinencode übersetzt. Je nach Art der betrach-

teten Steuerung unterscheiden sich die Programmiersprachen und die dafür eingesetzten Programmiersysteme. Da der Fokus im Rahmen dieser Arbeit auf Ablaufsteuerungen liegt, die in erster Linie für binäre Schaltfunktionalität in Produktionssystemen zuständig sind, wird hier nur die Erstellung von SPS-Programmen näher betrachtet.

2.4.1 Genormte Programmiersprachen

Zur Programmierung speicherprogrammierbarer Steuerungen haben sich zahlreiche Programmiersprachen entwickelt. Dabei haben einige ihren Ursprung in der klassischen Schalttechnik, während andere aus der Mikroprogrammierung stammen. Zudem haben viele dieser Sprachen herstellerspezifische „Dialekte“ *(Grötsch & Seubert 1997).*

Im Rahmen der bereits erwähnten Standardisierungsbemühungen von Steuerungen wurden in der *IEC 1131 Teil 3 (1994)* fünf derartige Programmiersprachen genormt. Hier wurden die zwei textuellen Sprachen „Anweisungsliste“ (AWL) und „Strukturierter Text“ (ST) sowie die drei graphischen Sprachen „Funktionbausteinsprache“ (FBS), „Kontaktplan“ (KOP) und „Ablaufsprache“ (AS) vereinheitlicht. Um SPS-Programme zu erstellen, werden aber darüber hinaus auch höhere Programmiersprachen verwendet, wie z. B. BASIC oder C *(Grötsch & Seubert 1997, S. 126–143*). Da im Rahmen dieser Arbeit die graphische Programmierung im Vordergrund steht, wird hier nur auf die graphischen Sprachen näher eingegangen (Abbildung 2-4).

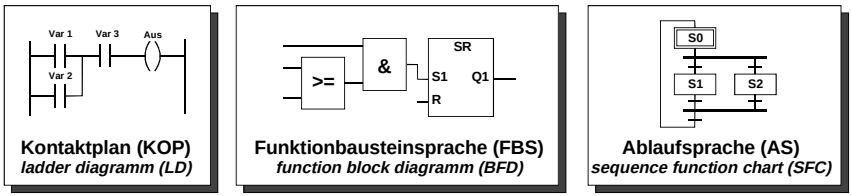


Abbildung 2-4: Graphische Programmiersprachen der IEC 1131-3

2.4.1.1 Kontaktplan (KOP)

Der Kontaktplan kommt aus dem Bereich der elektromechanischen Schützsteuerungen. Er beschreibt das Systemverhalten durch die Darstellung des Stromflusses mittels einzelner Netzwerke (Abbildung 2-4, links). An der linken und rechten Seite des Diagramms sind sogenannte Stromschienen. Die linke Schiene

besitzt den Zustand 1, und durch sie fließt Strom. Von dieser Schiene aus führen Verbindungen den Strom zu Elementen (Variablen), die abhängig von ihrem logischen Zustand die Weiterführung des Stroms zu den Ausgängen ermöglichen oder unterbinden. Diese Elemente können in Serie, d. h. hintereinander, oder parallel verbunden sein. Mittels KOP können aus bestehenden Relaisstromlaufplänen direkt SPS-Programme erstellt werden.

2.4.1.2 Funktionbausteinsprache (FBS)

Die Funktionbausteinsprache (FBS) stammt aus dem Bereich der Signalverarbeitung, wo im wesentlichen ganzzahlige Werte oder Gleitpunktwerte verarbeitet werden *(John & Tiegelkamp 1995, S. 133ff.)*. Wie bei KOP werden die SPS-Anweisungen in Netzwerke gegliedert (Abbildung 2-4, Mitte). Diese Netzwerke sind logische Schaltpläne mit Logikbausteinen (UND-Gatter, ODER-Gatter etc.). Zusätzliche Bausteine, wie z. B. Flip-Flops, Taktgeber, Vergleicher oder Zähler, können auch in den Schaltplan eingebaut werden. Diese Bausteine werden durch Linien verbunden, die den Signalfluß der Logikbausteine auf der linken Seite zu denen auf der rechten symbolisieren.

2.4.1.3 Ablaufsprache (AS)

Die Ablaufsprache (AS) ist eine Weiterentwicklung der Schrittkettenprogrammierung. Hier werden komplexe Aufgaben in überschaubare Einheiten zerlegt, und es wird der Kontrollfluß zwischen diesen Einheiten beschrieben. So besteht die Ablaufsprache aus Schritten und Transitionen (Abbildung 2-4, rechts). Ein Schritt kann entweder aktiv oder inaktiv sein. Er verweist auf eine Anzahl von Vorschriften, die durchzuführen sind, solange der Schritt aktiv ist. Die einzelnen Schritte sind durch Transitionen verbunden, an die Bedingungen gekoppelt sind. Ein aktiver Schritt wird erst dann inaktiv, wenn die Bedingung der Ausgangstransition erfüllt ist. Dadurch werden die nachfolgenden Schritte aktiviert. Mittels AS können sequentiell, alternativ, iterativ und parallel ablaufende Prozesse formuliert werden.

2.4.2 Beschreibungstechniken

Mit der zunehmenden Funktionalität heutiger Produktionssysteme ist auch die Funktionalität der eingesetzten Steuerungen gestiegen. Wurden früher viele Aufgaben noch mechanisch gelöst, werden sie heutzutage von der Steuerung mit Hilfe elektronischer Komponenten durchgeführt. Durch die Zunahme der Funktionalität hat auch die Komplexität der Steuerungsanweisungen zugenommen. Zur Bewältigung dieser Komplexität können sogenannte Beschreibungstechniken

eingesetzt werden, die der Erfassung bzw. Spezifikation der Anforderungen an die Steuerungssoftware dienen und somit die Entwicklung vereinfachen und strukturieren.

Beschreibungstechniken sind ein wesentliches Element des *Systems & Software Engineerung*, jener Wissenschaft, die sich mit der ingenieurorientierten Entwicklung und dem Betrieb großer Systeme der digitalen Inforamtionsverarbeitung und –kommunikation befassen, und hierfür Werkzeuge, Referenzarchitekturen und Vorgehensweisen bereitstellen *(Broy & Schmidt 1999).* Mittels Beschreibungstechniken werden definierte Sachverhalte der zu entwickelnden Software unter einem oder mehreren Gesichtspunkten modelliert. Dabei können für eine Beschreibungstechnik unterschiedliche Notationen, d. h. Symbole, verwendet werden. Eine Zusammenfassung der wesentlichen Beschreibungstechniken zeigt nach *Balzert (1996, S. 98)* Abbildung 2-5.

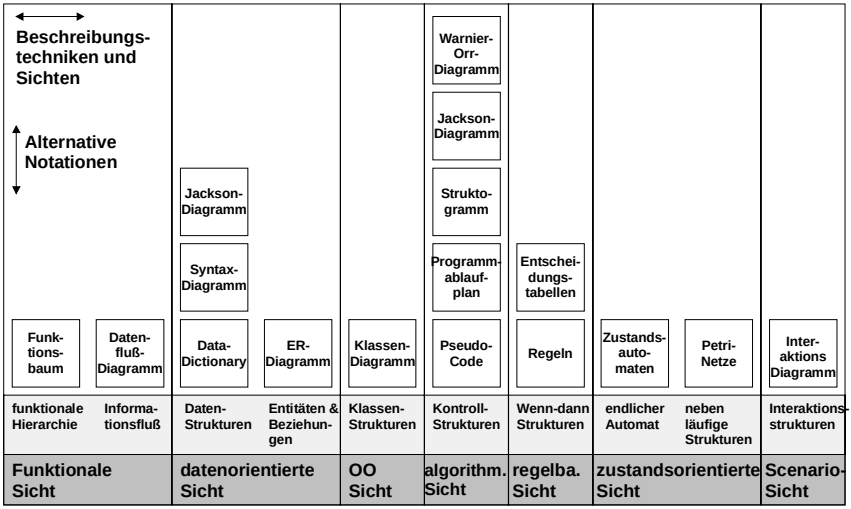


Abbildung 2-5: Gängige Beschreibungstechniken in der Softwaretechnik nach Balzert (1996, S. 98)

Im folgenden sollen jene Beschreibungstechniken näher erläutert werden, die zur Beschreibung von Abläufen und reaktivem Verhalten geeignet sind, wie dies für Steuerungsanweisungen erforderlich ist.

2.4.2.1 Zustandsautomaten bzw. Zustandsgraphen

Zustandsgraphen dienen der graphischen Spezifikation von Zuständen und Zustandsübergängen technischer Systeme. Interessant für den Bereich der Steuerungstechnik sind die sogenannten deterministischen Zustandsautomaten, in denen es bei einem Ereignis von einem Zustand aus immer nur einen Zustandsübergang gibt. Für die Zustandsautomaten gibt es verschiedene graphische Notationen, die in Form von Zustandsdiagrammen dargestellt werden. Alle Graphen haben gemeinsam, daß sie die Zustände durch Kreise bzw. durch abgerundete Rechtecke und die Übergänge zwischen den Zuständen anhand von Pfeilen, die auch Transitionen genannt werden, darstellen. Um einen Zustandsübergang zu ermöglichen, muß ein Ereignis vorliegen, das an die Transition gekoppelt ist. Nach *Balzert (1996, S. 275ff.)* können drei Typen von Automaten unterschieden werden: Mealy-, Moore- und Harel-Automaten (Abbildung 2-6):

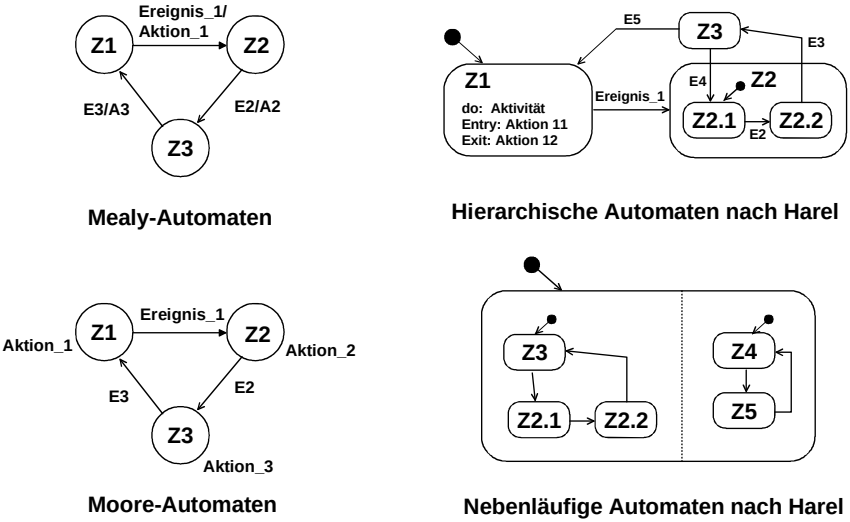


Abbildung 2-6: Varianten von Zustandsgraphen

Mealy-Automaten sind dadurch gekennzeichnet, daß die Ausgaben bzw. Aktionen, die durch den Graphen beschrieben werden, an die Zustandsübergänge gekoppelt sind. Hier repräsentieren die Zustände Zeitperioden bzw. Zeitintervalle. Die Zustandsübergänge sind Zeitpunkte. Dargestellt wird dies durch die zweiteilige Beschriftung der Transitionen. Der erste Teil gibt die Eingabe bzw. das Ereignis an. Der zweite Teil, der durch einen Trennstrich markiert wird, gibt die Ausgänge bzw.

die durchzuführende Aktion an. In Gegensatz dazu sind in **Moore-Automaten** die Ausgaben bzw. die Aktionen an die Zustände gebunden. Hier werden die Angaben über die Ausgänge bzw. Aktionen unterhalb der Zustände eingezeichnet. Wesentlich bei diesen Automaten ist, daß in den jeweiligen Zuständen jeweils nur eine Ausgabe bzw. Aktion erfolgt. Eine weitere Variante von Automaten sind **Harel-Automaten** *(Harel 1987),* die sich besonders zur Beschreibung komplexer Zusammenhänge eignen. Sie sind dadurch gekennzeichnet, daß sie zunächst beide oben beschriebenen Konzepte miteinander kombinieren. Zudem ist in den Harel-Automaten eine Verschachtelung von Zuständen möglich, so daß hierarchische Automaten entstehen. Im übrigen können hier Nebenläufigkeiten beschrieben werden. Dazu wird ein verschachtelter Zustand in mehrere parallel laufende Untergraphen gegliedert.

Zur weiteren Detaillierung der Grundtheorie von Zustandsautomaten wird auf *Hopcraft & Ullman (1990)* verwiesen.

2.4.2.2 Petri-Netze

Eine weitere in den Ingenieursdisziplinen gängige Beschreibungstechnik, deren Theorie auf die Dissertation von Carl Adam Petri *(Petri 1962)* zurückgeht, sind Petri-Netze. Sie eignen sich zur Modellierung, Analyse und Simulation von diskreten dynamischen Systemen mit Nebenläufigkeiten.

Grundsätzlich sind Petri-Netze gerichtete Graphen mit Kanten bzw. Pfeilen und Knoten. Dabei wird bei den Knoten zwischen Stellen und Transitionen unterschieden. Während die Stellen die verschiedenen Zustände repräsentieren, in denen sich das System befinden kann, beschreiben die Transitionen die Zustandsübergänge zwischen den Stellen. Die Verknüpfung der Stellen und Transitionen mit Hilfe der Kanten legt die statische Struktur des Petri-Netzes fest. Dabei müssen sich laut Definition Stellen und Transitionen abwechseln. Die Dynamik, d. h. das Wechseln der diskreten Zustände, wird durch das Wandern sogenannter Marken im Netz repräsentiert. Der Markenfluß erfolgt durch das Schalten der Transitionen. Erst wenn alle Stellen, die vor einer Transition liegen (Eingangsstellen) markiert sind, kann die Transition geschaltet werden, und die Marken können in die nachfolgenden Stellen (Ausgangsstellen) wandern.

In der Literatur werden verschiedene Netz-Klassen unterschieden. In Abhängigkeit von den Marken im Netz bzw. von den Markierungen, die eine Stelle einnehmen kann, werden Einmarken-Netze, Bedingungs-/Ereignis-Netze (B/E-Netze), Stellen-/ Transitionsnetze (S/T-Netze) und kolorierte Netze unterschieden (Abbildung 2-7).

Einmarken-Netze und **B/E-Netze** sind dadurch gekennzeichnet, daß die Stellen jeweils nur eine Marke aufnehmen können. In einem Einmarken-Netz darf zu jedem Zeitpunkt im gesamten Netz nur eine einzige Stelle markiert sein. In B/E-Netzen

können zu einem Zeitpunkt mehrere Stellen gleichzeitig mit jeweils einer Marke markiert sein. In **S/T-Netzen** und **kolorierten Netzen** sind dagegen mehrere Marken pro Stelle zugelassen. In diesen Netzen werden die Kanten gewichtet, d. h., an der Kante ist angegeben, wie viele Marken sie pro Schaltvorgang transportieren. Während in S/T-Netzen alle Marken gleich sind, können die Marken in einem kolorierten Netz unterschiedliche Farben besitzen und somit einen anderen Sachverhalt modellieren.

Eine weitere Klassifizierung von Petri-Netzen richtet sich danach, ob die Netze durch die Verfeinerung von Stellen oder Transitionen hierarchisiert werden können. In der Literatur sind sowohl Verfeinerungen von Stellen als auch von Transitionen zu finden. Eine Verfeinerung von Transitionen ist jedoch vorzuziehen, da dies einer besseren zeitlichen Auflösung entspricht *(Balzert 1996, S. 306ff., Abel 1990, S. 45).* Zur Darstellung von Zeitfolgen in Petri-Netzen können sowohl Stellen als auch Transitionen mit Zeitangaben versehen werden. In beiden Fällen verharren die Marken für eine bestimmte Zeitdauer (t > 0) auf den Stellen, bevor sie von den Transitionen verbraucht werden.

	graphische Darstellung	B/E-Netze	S/T-Netze	Kolorierte-Netze	zeit-behaftete Netze	hierar-chische Netze
Stelle		nur eine Marke	mehrere Marken	mehrere Marken	Zeitangabe möglich	verfeinerbar
Transition		keine Schaltbedin-gung nötig	keine Schaltbedin-gung nötig	Schalt-bedingung nötig	Zeitangabe möglich	verfeinerbar
Kante		ungewichtet	gewichtet	gewichtet	je nach Netztyp	je nach Netztyp
Marke		uniforme Marke	uniforme Marke	individuelle Marken	je nach Netztyp	je nach Netztyp

Abbildung 2-7: Varianten von Petri-Netzen

Zur weiteren Detaillierung dieser Theorie wird auf *Reisig (1985)* verwiesen. Einen Überblick über die verschiedenen Netzarten gibt Abbildung 2-7.

2.4.2.3 Kontrollstrukturen

Kontrollstrukturen gehören zu den ältesten Beschreibungstechniken und dienen der Darstellung des Ablaufes eines Algorithmus. Diese Darstellung erfolgt mit Hilfe der vier semantisch unterschiedlichen Kontrollstrukturen: Sequenz, Auswahl (einseitig, zweiseitig und mehrfach), Wiederholung und Aufruf eines weiteren Algorithmus. Die verbreitetste Notation für Kontrollstrukturen ist der Programmablaufplan (PAP), der in der *DIN 66001 1983* genormt ist. Repräsentativ für die unterschiedlichen Notationen wird die Notation des PAP in Abbildung 2-8 dargestellt.

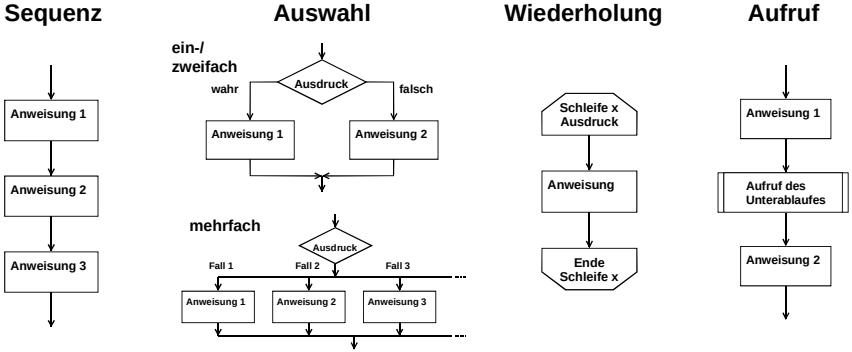


Abbildung 2-8: Die vier semantischen Elemente des PAP nach DIN 66001 (1983)

2.4.2.4 Unified Modeling Language (UML)

Durch die Vielzahl der in den letzten Jahren entwickelten objektorientierten Softwareentwicklungsmethoden, wie z.B. die Methoden von *Rambaugh (1991), Jacobsen (1992), Martin & Odell (1992), Booch (1994) und Selic (1994),* sind viele unterschiedliche Beschreibungstechniken entstanden. In der Unified-Methode von *Booch & Rambaugh (1995)*, die eine Zusammenfassung der Methoden OOAD[2] nach *Booch (1994)* und OMT[3] nach *Rambaugh (1991)* ist, wurde der erste Schritt in Richtung Vereinheitlichung von Beschreibungstechniken getätigt. Später wurden die Use-Case-Diagramme von *Jacobsen (1992)* sowie die State Charts von *Harel (1987)* der Unified-Methode hinzugefügt. Dadurch entstand die UML, die 1997 von der OMG (Objects Management Group) zum Quasistandard für Beschreibungstechniken

[2] OOAD: Object Oriented Analysis and Design

[3] OMT: Object Modeling Technique

spezifiziert wurde. Die UML *(Booch u. a. 1997)* ist in erster Linie eine einheitliche Notation zur Modellierung von objektorientierter Software. Zusätzlich zur graphischen Notation spezifiziert sie ein Metamodell, das die Semantik und Verknüpfung der graphischen Elemente beschreibt.

Die in der UML spezifizierten Diagramme können für die Entwicklung verschiedener Softwareanwendungen verwendet werde. Zur Beschreibung von Abläufen und Systemen mit reaktivem Verhalt, wie dies für Steuerungsanweisungen erforderlich ist, stehen folgende Diagramme zur Verfügung:

- Zustandsdiagramme: Zur Beschreibung des dynamischen Verhaltens einzelner Softwarebausteine werden die UML-Zustandsdiagramme spezifiziert. Bis auf wenige Punkte lehnen sich diese sehr stark an die State Charts von *Harel (1987)* an.
- Aktivitätsdiagramme: Zur Beschreibung von ablaufgesteuerten Sequenzen eines oder mehrerer Softwarebausteine werden sogenannte Aktivitätsdiagramme eingesetzt. In ihrer Semantik ähneln sie einfachen Petri-Netzen, wobei hier keine Marken verwendet werden.

Zusätzlich zu diesen Diagrammen spezifiziert die UML noch weitere Diagramme, wie z. B. Klassen-, Kollaborations- oder Komponentendiagramme. Eine weitere Detaillierung der UML-Diagramme ist bei *Booch u.a. (1997)* zu finden.

Mit der UML liegt eine einheitliche Beschreibungstechnik zur Modellierung und Dokumentation von objektorientierter Software vor, die im Vergleich zu bisherigen Beschreibungstechniken zahlreiche Vorteile bietet:

- Während beispielsweise die *IEC 1131 Teil 3 (1994)* ein implementierungsnaher Standard ist, der sich auf die Beschreibung von SPS-Software beschränkt, sind die Anwendungsbereiche der UML viel weiter gefaßt und nicht nur auf eine Art von Software begrenzt. Hinzu kommt, daß die UML keine Implementierungssprache, sondern eine Modellierungs- und Designsprache ist. Durch die Vielzahl der definierten Diagramme können sowohl große objektorientierte Anwendungen als auch reaktive Anwendungen mit Echtzeitanforderungen spezifiziert werden.
- In der UML können sogenannte „Stereotypes" definiert werden. Diese sind projekt-, unternehmens- oder methodenspezifische Erweiterungen vorhandener Modellelemente des UML-Metamodells *(Oesterreich 1997).* Dadurch kann die UML in einem definierten Rahmen einfach an neue Anwendungen angepaßt werden.
- Die UML ist durch eine starke Verbreitung in der Softwareindustrie gekennzeichnet. Mittlerweile gibt es zahlreiche CASE-Tools (Computer Aided Software Engineering), die die UML unterstützen. Zwar werden nicht immer alle

Diagramme unterstützt *(Versteegen & Versteegen 1998)*. Dennoch haben die CASE-Tool-Hersteller erkannt, daß die UML der Quasistandard der Beschreibungstechniken ist.

Aufgrund dieser Vorteile wird die UML als die einzusetzende Beschreibungstechnik gewählt.

2.5 Störungen in Produktionssystemen

2.5.1 Definition des Begriffes „Störung“

In der Literatur kennzeichnen zahlreiche Begriffe die unerwünschte Beeinträchtigung eines technischen Systems. Diese Begriffe sind durch voneinander abweichende Formulierungen mit unterschiedlicher Sicht auf Ursache und Wirkung geprägt. Nach *DIN 40041 (1990)* und *DIN 55350 Teil 11 (1995)* ist ein **Fehler** die unzulässige Abweichung eines Merkmals von seinem spezifizierten Soll-Zustand, wobei die Funktionserfüllung der Betrachtungseinheit nicht notwendigerweise beeinträchtigt ist. Dagegen ist eine **Störung** ein unbeabsichtigtes Aussetzen bzw. eine unbeabsichtigte Beeinträchtigung der Funktionserfüllung einer Betrachtungseinheit. Wird die Funktionsfähigkeit einer Betrachtungseinheit unbeabsichtigt unterbrochen, so wird hier von einem **Ausfall** gesprochen.

Ist der Grad der Beeinträchtigung einer Betrachtungseinheit so weit fortgeschritten, daß an der Einheit oder an nachgelagertem Material eine Veränderung oder Zerstörung festzustellen ist, handelt es sich um einen **Schaden** (siehe auch *DIN 31051 1985*).

Vossloh (1988, S. 27f.) faßt in seiner Arbeit die obengenannten Begriffe unter der Bezeichnung **Defekt** zusammen und versteht darunter unterschiedlich intensive Ausprägungen der Nichterfüllung einer Funktion in Abhängigkeit von der Zeit (Abbildung 2-9).

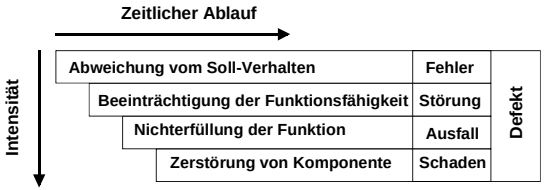


Abbildung 2-9: Intensität der Nichterfüllung einer Funktion nach Vossloh

Während für Vossloh der Fehler nur die leichteste Form der Nichterfüllung einer Funktion ist, wird in den Arbeiten von *Schönecker (1992, S. 12f.)* und *Wagner (1997, S. 13)* der Begriff Fehler als Allgemeinbegriff für die Funktionsbeeinträchtigung verstanden. In der Arbeit von *Koch (1996, S. 12)* wird der Begriff Störung als Allgemeinbegriff für die Funktionsbeeinträchtigung verwendet. Auch in dieser Arbeit soll der Begriff Störung im Vordergrund stehen, da hier die Beeinträchtigung bzw. das unbeabsichtigte Aussetzen einer Funktionseinheit im Vordergrund steht und weniger die unzulässige Abweichung eines Merkmals einer Funktionseinheit an sich.

2.5.2 Störungsauswirkung

In technischen Systemen bestehen zwischen den Systemkomponenten und den technischen Prozessen verschiedenartige Abhängigkeiten, so daß eine primäre Störung zu Folgestörungen führen kann. Diese Folgestörungen können sich auf temporal aufeinanderfolgende Prozesse und Zustände (Prozeß n folgt Prozeß n-1) bzw. auf die physikalisch gekoppelten Komponenten (Komponente n-1 beeinflußt Komponente n) auswirken. Je nach Standpunkt des Beobachters innerhalb einer sogenannten Kausalitätskette kann eine Störung entweder als Störungsursache oder Störungsauswirkung gesehen werden (Abbildung 2-10).

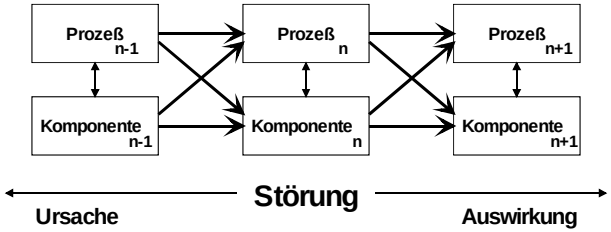


Abbildung 2-10: Kausalitätskette der Störungsausbreitung in einem technischen System

Aufgrund der logischen Wirkungskette von System–Prozeß–Produkt können sich Auswirkungen von Störungen in der Produktion am Produktionssystem, am Produktionsprozeß oder am Produkt selbst bemerkbar machen *(Schönecker 1992, S. 16f.).*

2.5.3 Störungsursachen

Kommt es zu einer Störung während der Inbetriebnahme eines Produktionssystems, sind die Ursachen oftmals auf Entwicklungs-, Fertigungs- oder Montagefehler zurückzuführen. Befindet sich das Produktionssystem dagegen bereits im Betrieb, können in Anlehnung an *Vossloh (1988, S. 35–43)* folgende Gruppen von Ursachen unterschieden werden:

1. **Externe Umwelteinflüsse:** Hierzu gehören Schwankungen oder komplette Ausfälle in der Energieversorgung sowie direkte, unzulässige thermische oder chemische Einwirkungen, wie z. B. erhöhte Betriebstemperatur oder Luftfeuchtigkeit.
2. **Organisatorische Ursachen:** Hierzu gehört das Fehlen von Rohteilen oder Betriebsmitteln sowie das Vorhandensein falscher bzw. ungeeigneter Rohteile oder Betriebsmittel.
3. **Bedienerbedingte Ursachen:** Hierzu gehören falsche Bedienereingaben oder fehlerhafte NC-Programme. Zudem zählen hierzu falsche Magazin- oder Palettenbelegungen, die von seiten des Bedieners verursacht wurden.
4. **Technische Ursachen:** Trotz des sachgemäßen Einsatzes eines Produktionssystems können Störungen aufgrund folgender Ursachen entstehen:
   - Prozeßbedingte Abnutzung: Darunter versteht man verschleißbedingte Ursachen sowie Störungsursachen, die durch den Fertigungs- oder Montageprozeß begünstigt werden. Hierzu gehören Störungsursachen wie beispielsweise der Verschleiß von Werkzeugen oder Führungsbahnen, die Lockerung oder Lösung von Verbindungen oder Kontakten aufgrund von Betriebsvibrationen oder Änderungen von Materialeigenschaften aufgrund von prozeßbedingten Temperatureinflüssen.
   - Alterung: Diese Störungsursachen sind prozeßunabhängig. Hierzu gehört beispielsweise die zeitbedingte Versprödung von Dichtungen und Schläuchen sowie die zeitbedingte Korrosion an Klemmen und Kontakten.

2.5.4 Störungsursachenbereiche

Bei einer Zuordnung von Störungen zu einzelnen Bereichen in einem Produktionssystem können verschiedene Ursachenbereiche unterschieden werden. Eine derartige Unterscheidung ist deshalb notwendig, da für die verschiedenen Bereiche unterschiedliche Verfahren zur Erfassung der Störungen notwendig sind. In Anlehnung an *Schwager (1983, S. 16)* können die Ursachenbereiche wie folgt eingeteilt werden (Abbildung 2-11):

1. **Steuerungsinterner Bereich:** Dieser Bereich umfaßt alle Störungen, die unmittelbar in einer Steuerung, d. h. in der Hardware oder Software, auftreten können, wie z. B. das Ausfallen einer Einschubkarte oder das Abstürzen eines Steuerungsprogrammes.
2. **Steuerungsperipherer Bereich** (NC-Funktionalität)**:** Hierzu gehören Defekte an Antrieben, Meßsystemen sowie deren Energie- und Informationsträger.

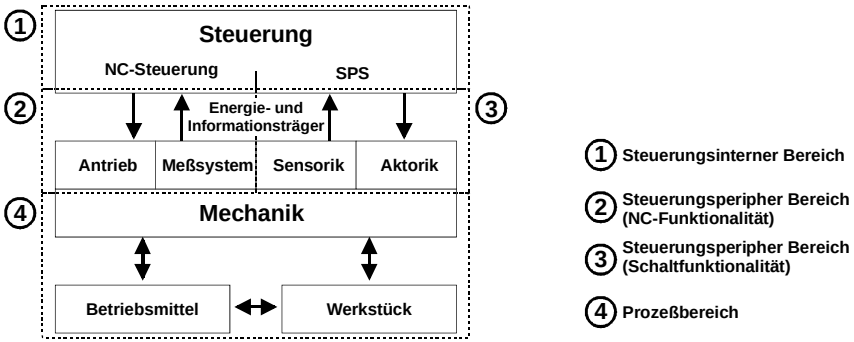


Abbildung 2-11: Störungsbereiche in einem Produktionssystem

3. **Steuerungsperipherer Bereich** (Schaltfunktionalität)**:** Dieser Bereich umfaßt alle Störungen, die an schaltenden Aktoren, den zugehörigen Stellgliedern, den verwendeten Signalgebern sowie den zur Energie- und Informationsübertragung benötigten Leitungen auftreten können.
4. **Prozeßbereich:** Zu diesem Bereich gehören Störungen an Betriebsmitteln aufgrund von Brüchen oder Verschleiß, Qualitätsmängel an Werkstücken und Rohlingen sowie Verformungen am Produktionssystem, die den Prozeß beeinträchtigen können.

2.6 Störungsbehandlung

2.6.1 Phasen der Störungsbehandlung

Die Störungsbehandlung ist ein Ablauf, der sich nach *Schönecker (1992, S. 16), Koch (1996, S. 12)* und *Wagner (1997, S. 54)* im wesentlichen in die drei Phasen **Störungserkennung**, **Störungslokalisierung** und **Störungsbehebung** gliedert.

Die Störungserkennung ist die erste Phase der Störungsbehandlung. Sie hat die Aufgabe, frühzeitig Störungen zu detektieren bzw. zu erfassen. Je nach Anwendung kann hier eine sofortige Eindämmung der Störung notwendig sein, um eine weitere Ausbreitung zu vermeiden. Im Anschluß an die Störungserkennung erfolgt die Störungslokalisierung. Hier erfolgt die Ortung der Störung sowie die Ursachenfindung.

Die letzte Phase der Störungsbehandlung ist die Störungsbehebung. Sie hat zunächst die Aufgabe der Beseitigung von Störungsauswirkungen und Störungsursachen. *Wagner (1997, S. 15)* unterscheidet unterdessen noch zwischen einer vollständigen Behebung, die mit einer Reparatur verbunden ist, und der partiellen Behebung, die nur eine Tolerierung bzw. Umgehung der Störung darstellt. Eine weitere Aufgabe der Störungsbehebung ist der Wiederanlauf, d. h. das Wiederaufnehmen des normalen Betriebs. Hier muß zuerst derjenige Zustand angefahren werden, von dem aus der unterbrochene Betrieb fortgesetzt werden kann. Anschließend erfolgt die Fortführung des Betriebs. Nach *Schönecker (1992, S. 18)* wird dieser Vorgang auch als Recovery-Ablauf bezeichnet. In einigen Fällen, in denen das Wiederaufsetzen des normalen Betriebs mit wesentlich mehr Aufwand verbunden ist als die reine Störungsbehebung, wird dieser Vorgang auch als eigene Phase betrachtet *(Hofmann 1990, S. 68ff.)*.

In zahlreichen Arbeiten werden in diesem Zusammenhang auch die Begriffe Diagnose oder Diagnosevorgang verwendet. Die Bedeutung ist jedoch sehr unterschiedlich. So verstehen *Grimm (1989)* und *Seifert (1992)* unter der Diagnose alle drei Phasen der Störungsbehandlung. Anders ist dies bei *Hofmann (1990)* und *Schwager (1983)*, die die Diagnose nur als Erkennung und Lokalisierung sehen. Nach *Härdtner (1992)* und *Isermann (1994)* stellt die Diagnose sogar nur die Phase der Lokalisierung dar. Zudem weist *Schönecker (1992)* darauf hin, daß auch die Ergebnisse eines Diagnosevorgangs als Diagnose bezeichnet werden können. Um eventuellen Mißverständnissen vorzubeugen, werden im Rahmen dieser Arbeit die Begriffe Diagnose und Diagnosevorgang bewußt gemieden.

2.6.2 Verfahren zur Störungserkennung und Störungslokalisierung

In der Literatur sind zahlreiche Verfahren zur Störungserkennung und -lokalisierung zu finden. Zudem gibt es unterschiedliche Klassifizierungen für diese Verfahren. In dieser Arbeit soll die Gliederung nach *Schönecker (1992, S. 32–37)* erläutert werden. Demzufolge werden die Verfahren basierend auf einer Gliederung nach *Puppe (1987)* zunächst in funktionale und wissensbasierte Verfahren aufgeteilt (Abbildung 2-12). Während die funktionalen Verfahren nochmals in numerische und statistische Verfahren unterteilt werden, gliedern sich die wissensbasierten Verfahren in

assoziative und modellbasierte Verfahren. Eine Sonderrolle besitzen in diesem Zusammenhang hybride Verfahren, da der Begriff „hybrid" unterschiedliche Bedeutungen haben kann. So werden modellbasierte Verfahren mit unterschiedlichen Beschreibungsmodellen als hybrid modellbasiert bezeichnet und Verfahren, die ein assoziatives und zugleich modellbasiertes Verhalten aufweisen, als hybrid wissensbasiert beschrieben. Verfahren, die eine Kombination mehrerer Methoden beinhalten, werden allgemein „hybrid“ genannt.

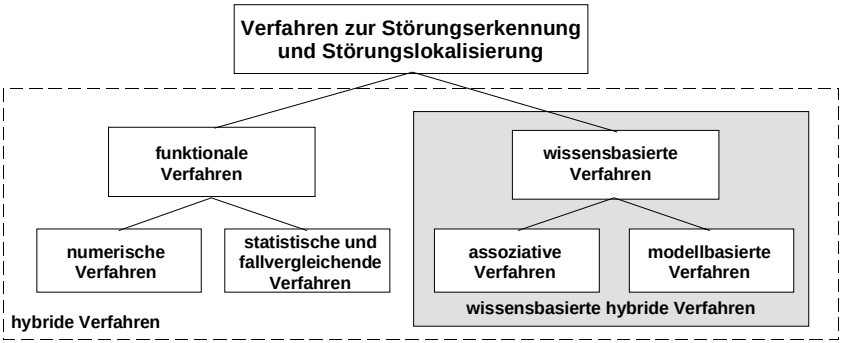


Abbildung 2-12: Verfahren zur Störungsbehandlung

Im folgenden sollen diese Verfahren kurz erläutert werden:

Numerische Verfahren

Als numerische Verfahren werden nach *Wagner (1997, S. 26)* alle signalverarbeitenden mathematischen Verfahren zur Prozeßdatenverarbeitung bezeichnet. Sie dienen der Datenverdichtung, Merkmalsextraktion oder Signalfilterung sowie dem direkten Erkennen von Fehlern, z. B. auf Basis der Boolschen Logik.

Fallvergleichende und statistische Verfahren

Bei den fallvergleichenden Verfahren wird ein aktuelles Symptommuster mit bekannten Symptom-Störungsursachen-Paaren verglichen. Durch einen Vergleich der Symptome mit gespeicherten Daten kann auf die Störungsursache geschlossen werden.

Bei den statistischen Verfahren wird hingegen der Zusammenhang zwischen Ursache und Auswirkung, d. h. dem Symptom, mit Hilfe statistischer Merkmale hergestellt. Hier wird das Problem der Bewertung aus einer unsicheren Menge möglicher Lösungen gelöst.

Assoziative Verfahren

Assoziative Verfahren benutzen Erfahrungswissen, welches in Form von Koppelungen zwischen Symptomen und Ursachen vorliegt und keine tieferen kausalen Zusammenhänge berücksichtigt. Hier liegt das Wissen in Form von Erfahrungswerten vor, die zwar durch ein komplexes Modell beschrieben werden können, aber aufgrund ihrer Auftrittshäufigkeit in expliziten Regeln (z. B. „Wenn-/dann-Regeln“) vorhanden sind.

Einen Spezialfall der assoziativen Verfahren stellen nach *Schönecker (1992)* die klassifizierenden Verfahren dar, die mittels eines Musterabgleiches zwischen Realität und abgebildeten Zustands- bzw. Fehlermustern die Störungslokalisierung und Ursachenfindung durchführen.

Modellbasierte Verfahren

Im Gegensatz zu den assoziativen Verfahren, die ausschließlich angeben, welche Symptome auf welche Ursachen hindeuten, wird bei den modellbasierten Verfahren die Verbindung zwischen Symptomen und Ursachen durch implizite, kausale Zusammenhänge hergestellt. Hier wird das nötige Wissen in Form von Modellen abgelegt, die zum einen die Struktur des Systems und zum anderen dessen Verhalten beschreiben. Zur Erfassung der Störungsursache wird zur Laufzeit das Modell mit der Realität verglichen. Die Modelle können mathematisch in Form von Differentialgleichungen oder analytisch in Form von Zustandsgraphen bzw. Petri-Netzen beschrieben werden.

2.7 Zusammenfassung

In diesem Kapitel wurden die wesentlichen Grundlagen beschrieben, die zum Grundverständnis dieser Arbeit benötigt werden. So wurden zunächst die verschiedenen Steuerungen in der Produktionstechnik sowie deren hardwaretechnische Entwicklungstrends erläutert. Anschließend wurde das Thema Entwicklung von Steuerungen betrachtet, wobei hier explizit die Entwicklung von Ablaufsteuerungen mit Schaltfunktionalität betrachtet wurde. Um ein Basisverständnis für die Thematik Störungen und Störungsbehandlung zu vermitteln, wurden zuletzt die wesentlichen Begriffe, Klassifikationen und Verfahren im Rahmen dieser Thematik erörtert.

3 Anforderungsanalyse

3.1 Übersicht

In diesem Kapitel werden die wesentlichen Anforderungen an die Methode zur produktiven Entwicklung von störungstoleranten Steuerungen analysiert. Wie bereits beschrieben, basiert die Methode zunächst auf einem Referenzkonzept für störungstolerante Steuerungen, das alle Phasen der Störungsbehandlung unterstützt. Zusätzlich beinhaltet die Methode ein systematisches Vorgehen zur Entwicklung konkreter störungstoleranter Steuerungsanwendungen, die auf dem Referenzkonzept aufbauen (Abbildung 3-1). Im folgenden sollen die wesentlichen Anforderungen an diese beiden Elemente der Entwicklungsmethode einzeln betrachtet werden.

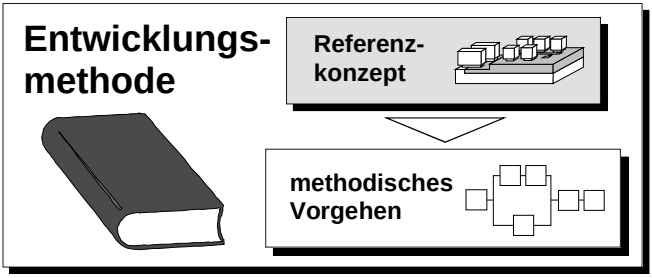


Abbildung 3-1: Elemente der Entwicklungsmethode

3.2 Anforderungen an das Referenzkonzept

Das Referenzkonzept stellt eine funktionale und strukturelle Richtlinie für die Software einer störungstoleranten Steuerung dar. Die Anforderungen an dieses Konzept sind in erster Linie technischer Natur und können nach verschiedenen Gesichtspunkten gegliedert werden. Laut *Ehrlenspiel (1988)* können Anforderungen an ein Produkt unter anderem nach den jeweiligen Lebensphasen gegliedert werden. Bei störungstoleranten Steuerungen sind die zwei wesentlichen Lebensphasen die Betriebsphase und die Entwicklungsphase. Aus beiden Phasen resultieren Anforderungen, die in den folgenden Absätzen einzeln behandelt werden.

3.2.1 Anforderungen aus der Betriebsphase

Die Anforderungen aus der Betriebsphase leiten sich aus dem Wunsch des Endanwenders ab, die technisch bedingten Stillstandszeiten seiner Produktionssysteme maximal zu reduzieren. Hierfür muß eine störungstolerante Steuerung eine vollständige, rechnergeführte Störungsbehandlung ausführen können. Dazu sind umfassende Funktionen zur Störungserkennung, -lokalisierung und -behebung erforderlich. Da das Automatisierungspotential der einzelnen Phasen der Störungsbehandlung unterschiedlich ist, muß die Steuerung zudem Algorithmen enthalten, durch die der Bediener in den Störungsbehandlungsprozeß mit einbezogen wird.

Zusätzlich zur von Ablaufsteuerungen bekannten Steuerungsfunktionalität muß das Referenzkonzept einer störungstoleranten Steuerung folgende Features berücksichtigen (Abbildung 3-2):

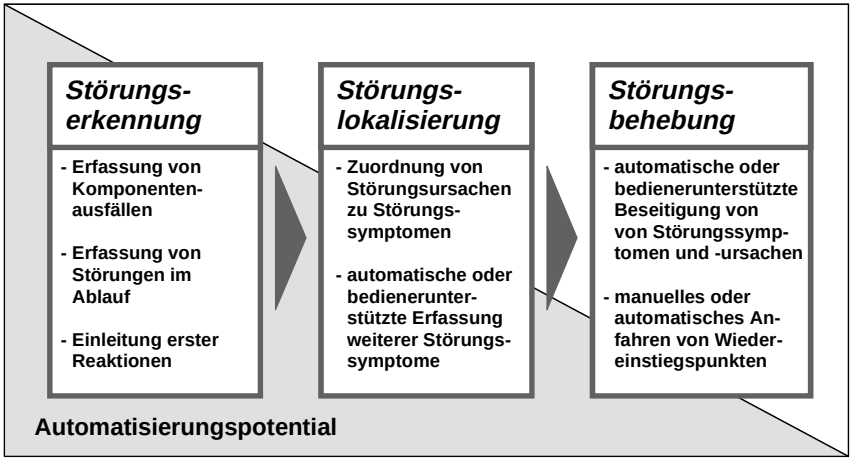


Abbildung 3-2: Anforderungen an das Referenzkonzept aus der Betriebsphase

Erkennung von Störungen im Produktionssystem

Grundsätzlich muß die störungstolerante Steuerung unterschiedliche Mechanismen enthalten, um Ausfälle von Maschinenkomponenten oder Fehler in Abläufen zu registrieren. In vielen Fällen muß hier eine erste Reaktion eingeleitet werden, um eine weitere Ausbreitung der Störungsfolgen zu verhindern. Aufgrund der hohen Anforderungen an die Reaktionszeit sowie der Tatsache, daß die Störungserkennung ein hohes Automatisierungspotential bietet, sollte die Störungserkennung weitgehend vollautomatisch erfolgen.

Lokalisierung von Störungen

Zur Störungslokalisierung und Ursachenfindung muß die störungstolerante Steuerung entsprechende Mechanismen zur Zuordnung von Störungsursachen zu Störungssymptomen enthalten. Zur eindeutigen Identifikation von Störungsursachen müssen eventuell weitere Symptome erfaßt werden. In zahlreichen Fällen kann dieser Vorgang vollautomatisch erfolgen, indem auf die Gebersignale der im Produktionssystem installierten Sensoren oder auf die vorhandenen Prozeßparameter und -variablen zurückgegriffen wird. In vielen Fällen können jedoch nicht alle Symptome automatisch erfaßt werden. Hier ist der Bediener bzw. dessen Wissen erforderlich, um die aufgetretenen Störungssymptome genau zu erfassen. Um möglichst viele Störungen im Betrieb eindeutig identifizieren zu können, ist eine starke Einbindung des Bedieners in den Störungslokalisierungsprozeß erforderlich. Deshalb sollte die störungstolerante Steuerung entsprechende Interaktionsmechanismen zur Verfügung stellen, um eine einfache Interaktion mit dem Bediener zu ermöglichen.

Behebung von Störungen

Nach der eindeutigen Erkennung und Lokalisierung einer Störung muß diese nun behoben werden. In einigen Fällen kann die Behebung der Störungssymptome vollautomatisch erfolgen. Dies ist beispielsweise dann der Fall, wenn nur ein einfaches Wiederholen der fehlgeschlagenen Aktion erforderlich ist. Oftmals müssen jedoch Reparaturarbeiten im Sinne einer Ursachenbeseitigung durchgeführt werden, die einen menschlichen Eingriff erfordern. In zahlreichen Fällen kann dies der Bediener vor Ort erledigen. Hierfür benötigt er aber konkrete Anweisungen, die die störungstolerante Steuerung liefern muß.

Des weiteren kann es zu Situationen kommen, in denen die Entscheidung für die einzuleitende Störungsbehebung nicht automatisierbar ist, sondern unter Einbeziehung des Bedieners erfolgen muß. Ist beispielsweise die Strategie im Rahmen der Störungsbehandlung ein bewußtes Ignorieren einer bestimmten Störungsursache, muß dieses Vorgehen mit dem Bediener abgestimmt werden. Auch das Anfahren definierter Zustände nach einer Störung kann eine Zustimmung des Bedieners erfordern, insbesondere dann, wenn es zu Kollisionen kommen kann. In diesen Fällen muß der Bediener mit in den Behebungsprozeß integriert werden können, damit er sein spezifisches Wissen und seine Fertigkeit einbringen kann. Hierfür muß die Steuerung entsprechende Mechanismen bereitstellen.

Im Zusammenhang mit dem Wiederanlauf des angehaltenen Produktionssystems erfordert dieser Vorgang nach *Schönecker (1996, S. 18)* oftmals mehr Aufwand als die Beseitigung der Störung selbst. Dies ist besonders dann der Fall, wenn es sich um ein verkettetes System handelt, in dem mehrere Systemkomponenten untereinander synchronisiert sind. Die störungstolerante Steuerung muß den

Wiederanlauf nach einer Störung berücksichtigen und hierfür eventuell verschiedene Wiederanlaufvarianten vorschlagen, aus denen der Bediener dann auswählen kann.

3.2.2 Anforderungen aus der Entwicklungsphase

Die Anforderungen an das Referenzkonzept, die aus der Entwicklungsphase resultieren, leiten sich aus der Forderung nach der Reduktion des benötigten Entwicklungsaufwands ab. Da dieser Aufwand jeweils stark mit der Komplexität des zu entwickelnden Systems verbunden ist, beschreibt die Anforderung an das Referenzkonzept Maßnahmen, die die Komplexität der störungstoleranten Steuerung reduzieren bzw. die Komplexität beherrschbar machen. Hierzu zählen folgende Punkte:

- **Modularisierung der Steuerungsaufgabe:** Ein bewährter Ansatz zur Reduktion von Komplexität ist die Modularisierung, d. h. die strukturelle Gliederung eines Systems in autarke Bausteine, die über klare Schnittstellen externe Bezüge definieren. Das Referenzkonzept muß es ermöglichen, die Steuerungsaufgabe einer störungstoleranten Steuerung in derartige autarke Bausteine zu gliedern.
- **Integration von Steuerung und Störungsbehandlung in einem Modul:** Bedeutend für die Komplexität eines modularen Systems ist die Modulkopplung, d. h. die Vernetzung zwischen den einzelnen Modulen im betrachteten System. Mit abnehmender Modulkopplung sinkt die Komplexität des Systems *(Balzert 1998, S. 474)*. Nach *Wagner (1998, S. 39)* ist jedoch für eine durchgängige, rechnergeführte Störungsbehandlung die starke Vernetzung zwischen Steuerungsfunktionalität und Störungsbehandlungsfunktionalität im System erforderlich. Zur Reduktion der Gesamtkomplexität soll deshalb eine modulbezogene Integration von Steuerung und Störungsbehandlung im Referenzkonzept berücksichtigt werden (Abbildung 3-3).

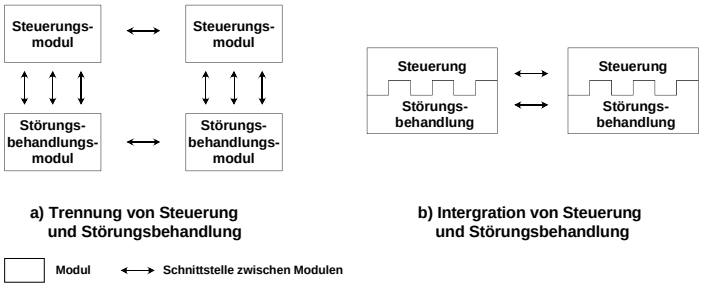


Abbildung 3-3: Kopplung zwischen Modulen

- **Graphische Beschreibungstechniken:** Um die Transparenz der Struktur und des Verhaltens eines Softwaresystems zu erhöhen, dienen graphische Beschreibungstechniken (siehe auch Kapitel 2.3.5). Eine wesentliche Anforderung ist es, daß das Referenzkonzept vollständig anhand derartiger graphischer Beschreibungstechniken beschrieben werden kann.
- **Einfache Verfahren zur Störungsbehandlung:** Wie bereits in Kapitel 2.5.2 beschrieben, gibt es zahlreiche Verfahren zur Störungserkennung und Störungslokalisierung. Viele dieser Verfahren sind jedoch mit einem erheblichen Entwicklungsaufwand verbunden, da hier zusätzlich zur Steuerungsanweisung komplexe Anlagenmodelle oder mathematische Formeln erstellt werden müssen, mit denen die Störungslokalisierung oder Ortung erfolgt. Das hier zu entwickelnde Referenzmodell soll weitgehend auf derartig komplexe Verfahren verzichten. Vielmehr sollen hier einerseits einfache und transparente Verfahren eingesetzt werden und andererseits die Informationen, die in den einfachen Steuerungsspezifikationen enthalten sind, verstärkt zur Störungsbehandlung herangezogen werden.

3.3 Anforderungen an das methodische Vorgehen

Zur Entwicklung störungstoleranter Steuerungen beinhaltet die hier beschriebene Methode zusätzlich zum Referenzkonzept ein methodisches Vorgehen, das zeigt, wie mit Hilfe moderner Beschreibungstechnik und einer effizienten Werkzeugunterstützung Steuerungen auf Basis des Referenzkonzepts systematisch aufgebaut werden können. Die hier gestellten Anforderungen leiten sich aus der Forderung nach einem produktiven Vorgehen ab. Insbesondere müssen folgende Punkte berücksichtigt werden.

- **Systematische Anforderungsanalyse:** Zu Beginn jeder systematischen Entwicklung muß die geforderte Funktionalität des zu entwickelnden Systems genau spezifiziert werden. Zur detaillierten Spezifikation der Funktionalität reiner Steuerungssoftware sind in der Literatur bereits zahlreiche erprobte Verfahren und Methodiken zu finden. Das hier zu entwickelnde methodische Vorgehen muß eine Systematik zur Erfassung und Analyse der geforderten Störungsbehandlungsfunktionalität bereitstellen.
- **Parallele Entwicklung von Steuerung und Störungsbehandlung**: Durch die funktionale Integration von Steuerung und Störungsbehandlung im Referenzkonzept muß das methodische Vorgehen eine parallele Entwicklung von Steuerungs- und Störungsbehandlungssoftware ermöglichen. Diese Parallelisierung muß sich auf alle Phasen der Softwareentwicklung beziehen, von der Anforderungserfassung bis hin zur Implementierung.

- **Inkrementelles Vorgehen:** Da es sich bei dem Referenzkonzept um eine modulare Steuerungsarchitektur handelt, ist es wichtig, daß das dazugehörige Vorgehen ein inkrementelles Vorgehen ermöglicht.
- **Keine Informationsverluste:** Um eine hohe Produktivität während der Entwicklung zu erzielen, darf es zwischen Entwicklungsphasen, wie z. B. der Analyse- und Designphase, keine Informationsverluste geben. Das zu entwickelnde Vorgehen muß hierfür Techniken einsetzen, die sicherstellen, daß die in einer Phase bereits gewonnenen Daten direkt und ohne Verluste in die nächste Phase übernommen werden können.

3.4 Zusammenfassung

Die hier zu erarbeitende Methode zur Entwicklung störungstoleranter Steuerungen basiert auf zwei wesentlichen Bestandteilen: einem Referenzkonzept und dem methodischen Vorgehen für den Aufbau und die Entwicklung störungstoleranter Steuerungen. In diesem Kapitel wurden hierzu die wesentlichen Anforderungen analysiert.

4 Bestehende Systeme und Ansätze zur rechnergeführten Störungsbehandlung

4.1 Übersicht

Im Bereich der Forschung und Technik gibt es eine Vielzahl von Ansätzen zur rechnergeführten Störungsbehandlung in Produktionssystemen. Diese Ansätze sind meist auf gezielte Anwendungen bezüglich System-, Prozeß- oder Produktstörungen zugeschnitten und unterstützen nur bestimmte Phasen der Störungsbehandlung.

In diesem Kapitel sollen die wichtigsten dieser Konzepte untersucht werden. Um diese Untersuchung zu strukturieren, werden zunächst die Ansätze aufgezeigt, die sich ausschließlich mit Störungserkennung befassen. Anschließend werden Ansätze aufgezeigt, die eine rechnergeführte Störungserkennung und Störungslokalisierung unterstützen. Weiterhin werden Ansätze betrachtet, die zusätzlich zur Störungserkennung und -lokalisierung auch die Störungsbehebung ausführen. Abschließend erfolgt eine Bewertung der Ansätze hinsichtlich ihrer Eignung als Referenzkonzept für eine störungstolerante Steuerung.

4.2 Ansätze zur reinen Störungserkennung

Aufgabe der Störungserkennung ist die Erfassung und Messung relevanter Daten und Informationen. Diese dienen dazu, Störungen festzustellen. Solche Ansätze können nach unterschiedlichen Kriterien klassifiziert werden. Mögliche Unterscheidungskriterien sind:

- zeitliches Meßverhalten (kontinuierlich/regelmäßig/sporadisch),
- Bezüge zum Prozeß (Präprozeß/In-Prozeß/Postprozeß),
- Position in der Störungswirkungskette (Ursachenerkennung – Auswirkungserkennung) und
- Störungsart (Produkt, Prozeß, System)

Bei den Ansätzen zur reinen Störungserkennung handelt es sich meistens um Systeme, die mittels der Messung spezifischer Größen auf eine ganz bestimmte Störung schließen lassen. Hier korrelieren die Meßgrößen mit den interessierenden Prozeßparametern oder der Störung, die erfaßt werden soll.

Beispielhaft für Systeme zur Störungserkennung sind die Ansätze zur Erkennung von Werkzeugbrüchen oder von Werkzeugverschleiß. In den Arbeiten von *König und Ketteler (1994)* wird dies mit Hilfe der Körperschallanalyse durchgeführt. Ein

umfassenderes System stellt die Einrichtung von *Nordmann (1994)* dar. Diese führt die Störungserkennung multisensorisch durch. Es werden Sensoren zur Messung von Wirkleistungs-, Kraft-, Hydraulikdruck oder Körperschallsignalen parallel eingesetzt, um eine bessere Absicherung der überwachten Werkzeuge zu gewährleisten. Die Einrichtung kann die benötigten Toleranzbänder der Eingangssignale selbstlernend ermitteln und eine parallele Auswertung der verschiedenen Eingangssignale auf Höhe und dynamischen Anteil durchführen. Über eine serielle Schnittstelle werden dann Meldungen über Werkzeugbrüche an die Steuerung geschickt.

Ein weiterer Ansatz zur Störungserkennung an Werkzeugen in Fertigungseinrichtungen stellt *Kühne (1985)* in seiner Arbeit vor. Das Besondere an diesem auf der Basis der Mustererkennung aufgebauten Konzept ist die universelle Einsetzbarkeit des Verfahrens für unterschiedliche Prozesse und verschiedene Fertigungseinrichtungen. Hier wird eine Programmiersprache mittels Anweisungstabellen zur Verfügung gestellt, mit der die prozeßspezifische Konfiguration des Systems vom Anwender vorgenommen werden kann. Das System ist auf einem separaten Mehrprozessorenrechner realisiert, der über binäre Signale einfache Meldungen an die Maschinensteuerung schickt.

Während sich die obigen Ansätze mit der Erkennung von werkzeugbedingten Störungen befassen, gibt es auch zahlreiche Arbeiten, die sich mit der Erkennung von Störungen am Produktionssystem selbst befassen. Ein Beispiel hierfür ist die Arbeit von *Eißler (1983),* die unterschiedliche Ansätze zur Erfassung von Störungen in der Maschinenperipherie mit Hilfe binärer Steuerungssignale beschreibt. Diese Ansätze basieren auf der Überwachung von Eingangssignalen und der kontinuierlichen Zeitüberwachung von Arbeitseinheiten. Außerdem wird in dieser Arbeit ein Konzept beschrieben, das auf Basis prozeßabhängiger Zeitabläufe Störungen erfaßt. Dazu werden die Ausführungszeiten einzelner Schritte sowie kompletter Maschinenzyklen kontinuierlich erfaßt und in Zeitebenen projiziert. Dadurch entstehen 3-D-Projektionen, die charakteristisch für unterschiedliche Maschinenzustände und -abläufe sind. Beim Auftreten von Störungen am System werden diese durch Verzerrungen in der Projektion erkannt. Für dieses Konzept wurde ein externes System entwickelt, das allerdings keine direkte Rückwirkung auf die Maschinensteuerung hat.

Weitere Ansätze zur Erkennung von Systemstörungen wurden von *Zender (1994)* beschrieben. Hierbei handelt es sich um die Erfassung von Störungen in der Hardware von sicherheitsgerichteten Steuerungen. Die Erfassung dieser Störungen erfolgt zum einen mit Hilfe hochfrequenter Testsignale, die zyklisch die einzelnen elektronischen Bauelemente durchlaufen und ihre Funktionsfähigkeit testen. Zum anderen werden Störungen dadurch erfaßt, daß redundante elektronische Baugruppen mit dem gleichen Programm parallel arbeiten und währenddessen die

Zustände aller Ein- und Ausgänge zyklisch untereinander austauschen und kontrollieren.

4.3 Ansätze zur Störungserkennung und Störungslokalisierung

Im Gegensatz zu den bisher beschriebenen Systemen sollen nun Ansätze vorgestellt werden, die zusätzlich zur Störungserfassung auch eine detaillierte Beschreibung von Ort, Art und Ursache der Störung beinhalten. Dabei soll im folgenden zwischen speziellen Ansätzen, die sich auf besondere Störungsursachen oder einzelne Ursachenbereiche im Produktionssystem beziehen, und universellen Ansätzen, die das komplette Produktionssystem umfassen, unterschieden werden.

4.3.1 Ansätze für einzelne Ursachenbereiche

In der Arbeit von *Vossloh (1988)* wird ein Ansatz zur Erkennung und Lokalisierung von Prozeßstörungen in Drehmaschinen beschrieben. Die Störungserkennung erfolgt durch den Vergleich von aktuellen Meßgrößen mit sogenannten Normalmodellen. Zur Störungslokalisierung dient eine externe Datenbasis, die mit Angaben über die vermutlich gestörte Maschinenfunktion und die Art der Störerscheinung aufgerufen wird. Mit Hilfe zweier Verknüpfungsmatrizen, die ein mit statistischem Wissen aufgebautes Strukturmodell der Maschine bilden, werden dem Benutzer die wahrscheinlichen Ursachen für den erfaßten Störungsfall angegeben.

He (1993) stellt in seiner Arbeit einen Ansatz zur Erkennung und Lokalisierung von Störungen in Vorschubantrieben von Fertigungssystemen vor. Bei diesem Ansatz basiert die Störungserkennung auf der Methode der Parameterschätzung von Prozeßmodellen der Antriebsstränge. Zur Störungslokalisierung dient eine Wissensbasis, die analytische Symptome mit heuristischem Störungswissen ergänzt. Mit Hilfe von Störung-Symptom-Bäumen erfolgt hier die detaillierte Störungsursachenfindung. Ähnliche Lösungsvorschläge stellen hierzu auch *Helml (1992)* und *Freyermuth (1993)* in ihren Arbeiten vor. Während auch *Helml (1992)* Vorschubantriebe in Fertigungssystemen betrachtet, werden in der Arbeit von *Freyermuth (1993)* Störungen an den Antriebssträngen von Industrierobotern behandelt.

Ein Konzept zur Erkennung und Lokalisierung von Störungen in der Maschinenperipherie stellt *Schwager (1983)* in seiner Arbeit vor. Hier wird auf Basis von Zustandsgraphen eine Beschreibungstechnik für komplexe Aufgaben in Produktionssystemen definiert. Zur Störungserkennung werden die Methoden der Zeitüberwachung und der Überwachung unzulässiger Wertekombinationen von

Gebern eingesetzt. Die Störungslokalisierung erfolgt durch das Vergleichen von Steuerungszuständen und Fehlermustern, die in sogenannte „Fehlerauswirkungsmatrizen“ abgelegt sind. Während im System von Schwager die Störungserkennung auf der Steuerung selbst läuft, wird zur Störungslokalisierung ein separater Rechner verwendet, der die Störungsursachen an den Bediener meldet.

Ein weiterer Ansatz, der sich mit Erkennung und Lokalisierung von Störungen in der Maschinenperipherie befaßt, ist die Arbeit von *Grimm (1987)*. Hier wird ein Konzept vorgestellt, das Steuerung und Störungserkennung hardware- und softwaretechnisch integriert. Dies erfolgt mittels einer problemorientierten Eingabesprache auf der Basis von Zustandsgraphen, die von einem mikroprogrammierbaren Prozessor interpretiert werden kann. Auch hier erfolgt die Störungserkennung durch Zeitüberwachung und die Überwachung fehlerhafter Signalkombinationen. Für die Störungslokalisierung dient ein Offline-Diagnosesystem, das dem Bediener Meldungen über die fehlerhaften Signale bzw. Steuerungszustände gibt. Zudem können im Diagnosesystem auch weitere textuelle Angaben über bekannte Fehler, deren Ort und Ursache sowie eventuelle Hinweise zu Behebungsmaßnahmen abgelegt werden.

4.3.2 Ansätze für komplette Produktionssysteme

In der Arbeit von *Kiratli (1989)* wird ein Konzept zur reinen Störungslokalisierung für komplette Produktionssysteme vorgestellt. Es wird ein wissensbasiertes Expertensystem entwickelt, das nach dem assoziativen Verfahren aus vorgegebenen Störungssymptomen an Produktionssystemen die Entstehungsursachen und deren Orte ermitteln kann. Im Expertensystem werden in Form von Baumstrukturen die Struktur des Fertigungssystems sowie Wissen über Störungsursachen und deren Symptome abgelegt. Zur Eingabe dieses Wissens steht dem Experten eine Wissensakquisitionskomponente zur Verfügung. Dem Anwender des Expertensystems, d. h. dem Bediener am Fertigungssystem, werden über ein Eingabeterminal zahlreiche Eingabemenüs bereitgestellt, um für erfaßte Störungssymptome am Produktionssystem Entstehungsursachen und -ort zu erhalten. Die Störungen müssen hier jedoch vom Bediener erfaßt werden.

Weitere Arbeiten, die sich mit dem Einsatz von wissensbasierten Systemen zur Störungslokalisierung in kompletten Produktionssystemen und der Strukturierung des dazu benötigten Wissen befassen, sind z. B. *Reuschenbach (1992), Härdtner (1992), Wiedmann (1993), Birkel (1995)* und *Anders (1998)*.

4.4 Ansätze mit anschließender Störungsbehebung

Im folgenden sollen einige Ansätze zur rechnergeführten Störungsbehandlung vorgestellt werden, die mehr als nur die Erkennung und Lokalisierung von Störungen unterstützen.

Einen Ansatz zur rechnergeführten Störungsbehandlung, der die Störungsbehebung mit einschließt, stellt das steuerungsperiphere System von *Diehl (1992)* dar. Hier wurde ein Störungsbehandlungsrechner entwickelt, der zwischen die Anlagensteuerung (SPS) und die Anlagenkomponenten (Sensoren, Aktoren, Stellglieder) geschaltet wird. Dieser Rechner verfügt über Wissenslisten zur Störungslokalisierung und kann als Reaktion auf eine Störung die Anlage abschalten oder die Störung tolerieren. Von besonderer Bedeutung sind hier die notwendigen überwachungsgerechten Komponenten, die für die Tolerierung von Störungen benötigt werden.

Schneider (1994) stellt in seiner Arbeit einen interessanten Ansatz für Störungen in der Peripherie von Werkzeugmaschinen vor. Dieser Ansatz sieht die Bereitstellung von zustandsgraphenbasierten Steuerungsanweisungen sowohl für den normalen als auch für den störungsbehafteten Betrieb vor. Dabei sind die Graphen für den störungsbehafteten Betrieb Abbildungen der Graphen des normalen Betriebs. Sie beinhalten aber nur noch jene Zustände, die das System nach einer Störung im eingeschränkten Betrieb einnehmen kann bzw. darf. Nach dem Auftreten einer Störung wird mit Hilfe eines weiteren Zustandsgraphen der entsprechende Graph für den störungsbehafteten Betrieb ausgesucht und aktiviert.

In der Arbeit von *Wagner (1997)* wird ein System entwickelt, das bei Störungen in maschinennahen Abläufen alle Phasen der Störungsbehandlung berücksichtigt und eine weitestgehend selbständige und umfassende Behandlung durchführt. Zur Störungserkennung dient hier ein universell konfigurierbares Baukastensystem, das Sensordaten verarbeitet und Informationen über aufgetretene Störungen generiert. Für die anschließende Lokalisierung wird ein iteratives Verfahren eingesetzt, das selbständig Testaktionen auf Basis eines Anlagenmodells generiert, ausführt und auswertet. Dabei kommt der Sicherheit der Testabläufe eine besondere Bedeutung zu, weshalb hier ein Mechanismus entwickelt wurde, der kollisionsfreie Zustände für die Ausführung der Testaktionen garantiert. In der letzten Phase der Störungsbehandlung können nach der Störungslokalisierung neben automatischen Reparaturaktionen, die der vollständigen Störungsbehebung dienen, auch Maßnahmen zur Umgehung der Störungen ergriffen werden.

Koch (1996) stellt in seiner Arbeit ein Konzept zur umfassenden Störungsbehandlung auf Zellenebene vor. Hier wird ein Zellenrechner entwickelt, der aus auf drei hierarchischen Ebenen angesiedelten autarken Steuerungsmodulen besteht. Dabei dienen die Module der obersten Ebene der zelleninternen und zellenübergreifenden Auftragsdisposition und behandeln organisatorische Störungen in der Fertigungszelle. Die Module der mittleren Ebene koordinieren das Zusammenspiel der

Komponenten der Fertigungszelle. Auf unterster Ebene sind Module angesiedelt, die der dezentralen, aufgabenorientierten Steuerung einzelner Zellenkomponenten dienen. Sowohl die Module der mittleren als auch die der unteren Ebene verfügen über zahlreiche Funktionen zur Störungserkennung und -lokalisierung. Im Rahmen der Störungsbehebung wurden hier Algorithmen entwickelt, die nach einer Störung Abläufe wiederholen oder redundante Zellenkomponenten aufrufen können.

Das Konzept von *Schönecker (1992)* ist ebenfalls ein Ansatz zur umfassenden Störungsbehandlung auf Zellenebene. Hier wird ein Zellenrechner, der zusätzlich zum Kernprozeß der Auftragsabwicklung drei Tasks zur umfassenden Störungsbehandlung besitzt, mit einem wissensbasierten System verknüpft. Bei den drei Tasks handelt es sich um eine Störungserkennungs-, eine Störungslokalisierungs- und eine Störungsbehebungstask, die zusammen die gesamte Kette der Störungsbehandlung abdecken. Die Erkennungstask empfängt die unterschiedlichen Störungsmeldungen der einzelnen Zellenkomponenten. In der Lokalisierungstask können Störungsursachen anhand von Störungszuordnungslisten ermittelt werden. Reichen diese Informationen zur Ursachenfindung nicht aus, wird die angeschlossene Wissensbasis aktiviert, die mittels assoziativen Störungswissens und einer Hypothesengenerierung eine weitere Analyse der Störung durchführt. Mit der abschließenden Behebungstask können fehlerhafte Zellenabläufe wiederholt oder alternative Abläufe aktiviert werden, um den normalen Betrieb wieder zu aktivieren.

Einen weiteren Ansatz zur Störungsbehandlung mit Reaktion auf erkannte und lokalisierte Störungen beschreibt die Arbeit von *Seifert (1992)*. Hier wird ein Konzept vorgestellt, das auf dem Vergleich von zeitlichem und funktionalem Soll- und Ist-Verhalten des zu überwachenden Systems basiert. Der Soll-Zustand wird dabei aus Petri-Netz-basierten Modellen entnommen, die den realen Prozeß zeit- und funktionsparallel simulieren. Der Ist-Zustand wird durch aktuelle Signalzustände steuerungsperipherer Sensoren erfaßt. Ein Unterschied zwischen Soll und Ist-Verhalten läßt das Störungsbehandlungssystem auf Störungsart und -ort schließen. Mittels Wenn-/dann-Regeln, die in einer Reaktionstabelle abgelegt sind, werden die anschließenden Reaktionen auf die lokalisierte Störung bestimmt. Als konkrete Reaktionen stehen hier die Unterbrechung der Stromzuführung, das Anhalten aller Antriebe oder das Ignorieren der eingetretenen Störung zur Verfügung.

Zuletzt wird hier auf die Arbeit von *Enderle (1989)* verwiesen. Diese befaßt sich mit der Behandlung von Störungen in Montagesystemen, die auf die Unzulänglichkeit von Bauteilen zurückzuführen sind. Hierfür wurde ein Konzept entwickelt, das mit Hilfe eines bestimmten Algorithmus automatische Reaktionsstrategien auf Störungen erarbeitet. Ähnlich wie bei dem Ansatz von *Seifert (1992)* wird hierzu zur Störungserkennung ein vereinfachtes kinematisches Ersatzmodell des Montagesystems verwendet, das in Form von Zustandsvektoren vorliegt. Durch den Vergleich der modellierten Soll-Zustände und gemessenen Ist-Zustände werden Störungen im

Montagesystem erfaßt und in Form von Fehlervektoren abgelegt. Auf Basis dieser Fehlervektoren können automatisch einfache Kompensationsschritte eingeleitet werden, die das Montagesystem wieder in einen definierten Zustand versetzen, von dem aus der Montagevorgang fortgesetzt wird. Störungen im Montagesystem selbst wurden hier jedoch nicht berücksichtigt.

4.5 Bewertung der Ansätze

Abbildung 4-1 faßt nochmals die wichtigsten Merkmale der untersuchten Ansätze zusammen. Aufgelistet werden folgende Punkte:

- unterstützte Störungsbehandlungsphasen (siehe auch Kapitel 2.6.1)
- betrachteter Ursachenbereich (siehe auch Kapitel 2.5.4)
- informationstechnische Ebene (siehe auch Kapitel 2.2)

		Schwager 83	Kühne 85	Grimm 87	Vossloh 88	Ederle 89	Kiriatli 89	Diehl 92	Freyermuth 92	Helml 92	Reuschenbach 92	Schönecker 92	Seifert 92	Wiedmann 93	Nordmann 94	Schneider 94	Zender 94	Koch 96	Wagner 97	Anders 98
Störungs-behandlungs-phasen	Störungs-erkennung	X	X	X	X	X		X	X	X	X	X	X		X		X	X	X	X
	Störungs-lokalisierung	X		X	X	X	X	X	X	X	X	X	X	X				X	X	X
	Störungs-behebung					X		X				X	X			X		X	X	
betrachtete Verursachungs-bereiche	steuerungs-interner Bereich						X													
	NC-Bereich				X		X		X	X										
	Schaltfunktio-nalitätsbereich	X		X			X	X			X	X	X	X		X		X	X	X
	Prozeßbereich		X		X	X	X							X	X		X			
betrachtete Steuerungs-ebenen	Steuerungs-ebene	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X
	Zellenebene					X	X				X	X	X					X		X
	Leitebene										X		X					X		

Ansätze auf Maschinenebene, die alle Phasen zur Behandlung von Störungen in der Anlagenperipherie berücksichtigen

Abbildung 4-1: Bewertung bestehender Ansätze zur Störungsbehandlung

Aus der Matrix kann zunächst entnommen werden, daß nicht alle aufgelisteten Ansätze für die hier gestellte Aufgabenstellung der umfassenden Störungsbe-

handlung ausgelegt wurden. Des weiteren gibt die Matrix Auskunft darüber, welche Ansätze sich auf Maschinenebene mit Störungen in der Schaltfunktionalität der Maschinen- und Anlagenperipherie befassen. Die markierten Konzepte der Matrix stellen zwar für die einzelnen Phasen der Störungsbehandlung interessante Lösungen dar. Als komplettes Referenzkonzept für eine vereinfachte Entwicklungsmethode sind sie jedoch aus folgenden Gründen nicht geeignet:

- Viele Ansätze betrachten zwar alle Phasen der Störungsbehandlung, unterstützen diese aber nicht ausreichend. Dies ist besonders im Zusammenhang mit der Störungsbehebung der Fall. So beschränken sich viele Lösungen auf das Stillsetzen der Maschine, das Ignorieren einer Störung oder das einfache Vorschlagen von Reparaturanweisungen.
- Ansätze, die eine ausführliche Störungslokalisierung und -behebung unterstützen, verwenden hierfür komplexe Maschinen- und Anlagenmodelle. Mitunter werden auch komplexe Berechnungsalgorithmen aus der künstlichen Intelligenz, wie z.B. Inteferenzverfahren und neuronalen Netzen eingesetzt. Derartige Ansätze verkomplizieren jedoch den Entwicklungsvorgang und sind mit erheblichem Aufwand verbunden.

4.6 Zusammenfassung

In diesem Kapitel wurden zahlreiche Konzepte zur rechnergeführten Störungsbehandlung untersucht. Dabei wurden Ansätze zur Störungserkennung, -lokalisierung und -behebung für verschiedene Anwendungsbereiche betrachtet. Viele dieser Ansätze stellen interessante Lösungen für einzelne Phasen der Störungsbehandlung dar. Als Referenzkonzept der hier zu entwickelnden Methode, das einerseits alle Phasen ausreichend unterstützt und andererseits hierfür keine komplexen Algorithmen oder Modelle einsetzt, sind die bestehenden Ansätze jedoch nicht geeignet.

5 Referenzkonzept der störungstoleranten Steuerung

5.1 Übersicht

In diesem Kapitel wird das Referenzkonzept der zu entwickelnden Methode vorgestellt (Abbildung 5-1). Es beschreibt Funktionalität und Aufbau der Steuerungssoftware einer störungstoleranten Steuerung und soll als Gestaltungsrichtlinie für störungstolerante Steuerungsanwendungen dienen. Zur besseren Veranschaulichung dieses Konzepts werden hier graphische Beschreibungstechniken eingesetzt. Aufgrund der in Kapitel 2.3.5.4 beschriebenen Vorteile der UML, sollen hierfür die UML-Diagramme eingesetzt werden.

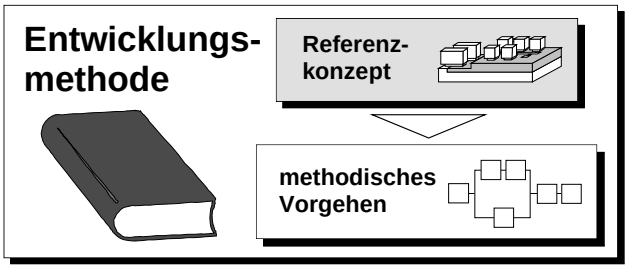


Abbildung 5-1: Das Referenzkonzept als Element der Entwicklungsmethode

Die Erläuterung des Referenzkonzepts erfolgt in drei Abschnitten. Die ersten zwei Abschnitte befassen sich mit der grundlegenden Funktionalität des Referenzkonzepts. Dazu wird zunächst beschrieben, wie Steuerungsfunktionalität auf Basis diskreter Zustände ohne Störungsbehandlung spezifiziert wird. Anschließend werden die im Referenzkonzept verwendeten Methoden zur Störungserkennung, Störungslokalisierung und Störungsbehebung aufgezeigt. Der letzte Abschnitt dieses Kapitels befaßt sich mit dem strukturellen Aufbau des Referenzkonzepts. Es wird ein sogenannter *störungstoleranter Steuerungsbaustein* als Grundelement der störungstoleranten Steuerung vorgestellt, in den die verschiedenen Funktionen zur Steuerung und Störungsbehandlung eingebettet sind.

5.2 Die Steuerungsfunktionalität

5.2.1 Gliederung in autarke Bausteine

Das Prinzip der Modularisierung ist eine in den Ingenieurdisziplinen altbekannte und bewährte Methode zur effizienten und wirtschaftlichen Entwicklung technischer Systeme. Durch die Modularisierung werden Systeme in weitestgehend kontextunabhängige Bausteine gegliedert, die in sich abgeschlossen sind und ihre externen Bezüge über klar definierte Schnittstellen regeln. Diese Strukturierung ermöglicht eine weitgehend unabhängige Entwicklung, Prüfung und Wartung der Bausteine losgelöst von ihrer Umgebung.

Die Modularisierung wird auch auf die hier entwickelte Steuerung und ihre Steuerungssoftware übertragen. Dazu sieht die Steuerung die Gliederung ihrer gesamten Steuerungsaufgabe in einzelne Aufgabenpakete vor, die von separaten, autarken **Steuerungsbausteinen** abgearbeitet werden. Mit den Steuerungsbausteinen können einzelne **Funktionseinheiten** eines technischen Systems, aber auch größere **Baugruppen** bis hin zu **kompletten Maschinenaggregaten** gesteuert werden.

Unter einer Funktionseinheit wird hier das kleinste steuerbare Element in einem technischen System verstanden. Ein Beispiel für eine Funktionseinheit ist eine einfache Achse, die aus einem linearen Zylinder und zwei Endlagegebern besteht (Abbildung 5-2). Unter einer Baugruppe wird hier die Zusammenfassung mehrerer Funktionseinheiten verstanden. Ein Beispiel für eine Baugruppe ist ein komplexer Greiferarm, der aus mehreren einfachen Achsen besteht. Ein Maschinenaggregat besteht wiederum aus mehreren Baugruppen, wie dies beispielsweise bei einem kompletten Werkzeugwechsler eines Bearbeitungszentrums der Fall ist.

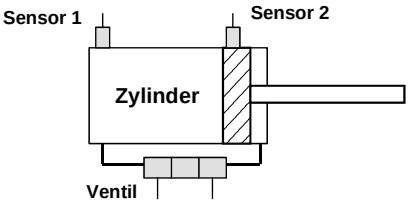


Abbildung 5-2: Beispiel für eine Funktionseinheit

In der störungstoleranten Steuerung stellt jeder Steuerungsbaustein sogenannte **Steuerungsdienste** zur Verfügung, die andere Bausteine von außen aufrufen können. Mittels dieser Dienste kann beispielsweise der Zustand der gesteuerten

Funktionseinheit manipuliert werden. Ein Beispiel hierfür sind die Dienste „Zylinder einfahren“ und „Zylinder ausfahren“, die der Steuerungsbaustein einer einfachen Achse zur Verfügung stellt.

Im Rahmen der Steuerung größerer Baugruppen oder kompletter Aggregate werden von den zugehörigen Steuerungsbausteinen Dienste zur Verfügung gestellt, die wiederum auf die Dienste anderer Steuerungsbausteine zurückgreifen. Ein Beispiel hierfür ist der Steuerungsbaustein eines kompletten Werkzeugwechslers, der zur Bereitstellung seines Dienstes „Werkzeug wechseln“ auf die Dienste seiner Baugruppen und Funktionseinheiten zurückgreift (Abbildung 5-3). Hier dienen die Steuerungsbausteine der Baugruppen und Funktionseinheiten quasi als Server für den Client „Steuerungsbaustein des Werkzeugwechslers“.

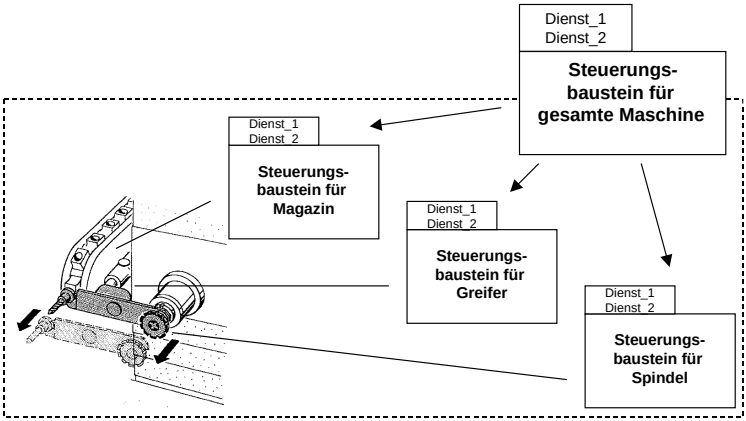


Abbildung 5-3: Interaktion zwischen Steuerungsbausteinen

Durch die Strukturierung der störungstoleranten Steuerung in autarke Steuerungsbausteine, die nach dem hier vorgestellten Client-Server-Prinzip interagieren, können die Vorteile der Modularisierung hinsichtlich unabhängiger Entwicklung, Prüfung und Wartung voll genutzt werden.

5.2.2 Steuerung mittels diskreter Zustände

Ähnlich wie in zahlreichen Arbeiten, die sich mit Methoden zur Entwicklung von Steuerungen bzw. Steuerungsanweisungen für Maschinen befassen, wie z. B. *Fleckenstein (1987), Seifert (1992), Schneider (1994) oder Moßig & Stäble (1995)*, werden zur Beschreibung der Steuerungsanweisungen eines Steuerungsbausteines

die diskreten Zustände des zu steuernden Systems verwendet. Zur Modellierung der Zustände und der zulässigen Zustandsübergänge ist eine Beschreibungstechnik mit zustandsorientierter Sicht erforderlich. Dazu werden in dieser Arbeit die Zustandsgraphen der UML eingesetzt. Sollen diese Graphen gleichzeitig zur Beschreibung von Steuerungsanweisungen dienen, muß klargestellt sein, wie die Steuerungsanweisungen mit den Zuständen bzw. Zustandsübergängen verknüpft sind. In Anlehnung an die Semantik der UML-Zustandsgraphen sieht dies wie folgt aus:

- Zur Laufzeit können die Zustände eines Zustandsgraphen den Status aktiv oder inaktiv einnehmen, dabei kann in einem herkömmlichen Graphen zu einem Zeitpunkt immer nur ein Zustand aktiv sein.
- In Verbindung mit der Aktivierung eines Zustands können zwei verschiedene Aktionen unterschieden werden:
  1. Entry-Aktionen werden einmalig beim Eintreten in einen Zustand, d. h. nur beim Aktivieren des Zustands, ausgeführt.
  2. Exit-Aktionen werden einmalig beim Verlassen des Zustandes, d. h. nur beim Deaktivieren des Zustands, ausgeführt.
- Zur Deaktivierung bzw. zum Verlassen eines Zustands muß eine zugehörige Ausgangstransition feuern, d. h., die Schaltbedingung der Transition muß erfüllt sein.

In Abbildung 5-4 ist beispielhaft aufgezeigt, wie die modellierten Zustände eines Zylinders zur Beschreibung der Steuerungsanweisungen dienen.

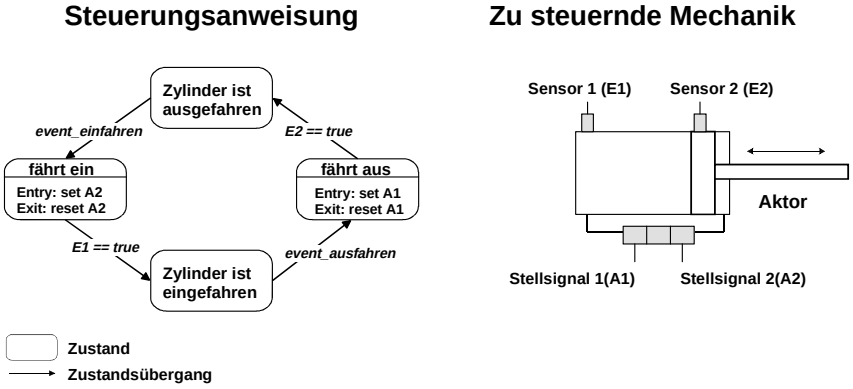


Abbildung 5-4: Diskrete Zustände zur Spezifikation von Steuerungsanweisungen

Werden Zustandsgraphen zur diskreten Steuerung technischer Systeme eingesetzt, entspricht im störungsfreien Fall der reale Zustand des gesteuerten Systems dem augenblicklich aktiven Zustand im zugehörigen Zustandsgraphen. Nach *Hofmann (1990, S. 60)* wird dies auch als ***Integrität*** zwischen einer diskreten Steuerung und einem technischen System bezeichnet. Diese Integrität ist jedoch im störungsbehafteten Fall nicht gegeben. Dieser Aspekt wird in einem späteren Kapitel noch weiter detailliert.

5.2.3 Idle- und Running-Zustände

Im Rahmen der Modellierung technischer Systeme werden im Referenzkonzept zwei Arten von diskreten Zuständen unterschieden:

- ***Idle-Zustände*** spezifizieren Situationen, bei denen das technische System eine Ruhelage einnimmt und diese zeitweise beibehält. Beispielhaft für derartige Zustände sind bei der Modellierung eines Zylinders die beiden Endlagen der Bewegung, d. h. die Zustände „ausgefahren“ und „eingefahren“.
- ***Running-Zustände*** spezifizieren Situationen, bei denen sich das modellierte System bewegt und dabei seine Ruhelage verändert. In bezug auf den Zylinder sind es die Zustände „Zylinder fährt ein“ und „Zylinder fährt aus“.

Bei der Modellierung eines Systems anhand von Idle- und Runningzuständen können einem Idle-Zustand immer nur Running-Zustände folgen. Einem Running-Zustand können dagegen Idle- und Running-Zustände folgen. Somit können zwischen zwei Idle-Zuständen auch mehrere Running-Zustände liegen.

Werden Idle- und Running-Zustände zur Beschreibung von Steuerungsanweisungen herangezogen, beinhalten die Idle-Zustände keine Aktionen, die eine Zustandsänderung im gesteuerten System bewirken. Im Idle-Zustand wird so lange gewartet, bis die Weiterschaltbedingung einer seiner Ausgangstransitionen erfüllt ist. Mit dem Feuern der Ausgangstransition wird der nachfolgende Running-Zustand aktiviert.

Den Running-Zuständen des Zustandsgraphen sind dagegen Aktionen zugeordnet, die eine Bewegung, d. h. Zustandsänderung, im gesteuerten System hervorrufen. So besitzt beispielsweise der Running-Zustand „Zylinder fährt aus“ eine Aktion, die das Ausfahren des realen Zylinders bewirkt. Die Weiterschaltbedingungen der Ausgangstransitionen von Running-Zuständen sind immer mit Signalen verknüpft, die das Erreichen einer bestimmten Soll-Position signalisieren. Diese können beispielsweise Signale von Positionsgebern sein.

Eine Kette von Running-Zuständen zwischen zwei Idle-Zuständen stellt jeweils einen der in Kapitel 5.2.1 beschriebenen Dienste eines Steuerungsbausteines dar. Während der Durchführung eines Dienstes wandert die Funktionseinheit immer von einem Idle-Zustand ***(I1)*** in einen nächsten Idle-Zustand ***(I2)***. Der Aufruf des Dienstes

erfolgt durch das Setzen der Weiterschaltbedingung der Ausgangstransition des Idle-Zustands (I1). Das erfolgreiche Beenden eines Dienstes signalisiert eine Aktion im Idle-Zustand (I2). Diese Aktion schickt eine Rückmeldung an den Auftraggeber, d. h. den aufrufenden „Client“.

In Abbildung 5-5 wird dieser Sachverhalt nochmals verdeutlicht. Dargestellt ist der Zustandsgraph eines modellierten Zylinders mit zwei Idle- und zwei Running-Zuständen. Die beiden Idle-Zustände stellen die Endlagen, in denen sich der Zylinder befinden kann, dar. Dargestellt sind auch die beiden Dienste „einfahren“ und „ausfahren“, die der Zylinder bzw. dessen Steuerungsbaustein zur Verfügung stellt. Die Dienste enthalten jeweils einen einzigen Running-Zustand, in dem die Lage des Zylinders verändert wird. Für den Dienst „Einfahren“ ist der Idle-Zustand I1 gleich „Zylinder ist ausgefahren“ und der Idle-Zustand I2 gleich „Zylinder ist eingefahren“. Für den Dienst „Ausfahren“ ist der Idle-Zustand I1 gleich „Zylinder ist eingefahren“ und der Idle-Zustand I2 gleich „Zylinder ist ausgefahren“.

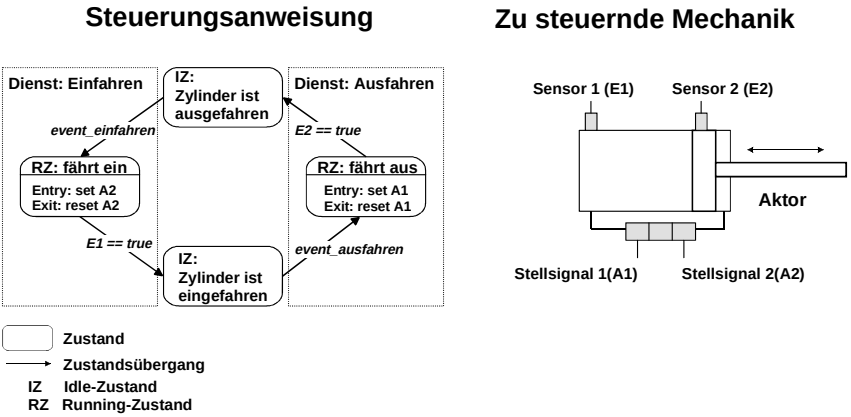


Abbildung 5-5: Dienste im Zustandsgraphen

5.3 Die Störungsbehandlungsfunktionalität

5.3.1 Voraussetzungen im Rahmen der Störungsbehandlung

Das hier zu entwickelnde Steuerungskonzept soll gemäß den Anforderungen aus Kapitel 3.2.1 eine rechnergestützte Behandlung von Störungen in der Maschinen-

und Anlagenperipherie ausführen. Um diese zu ermöglichen, werden folgende zwei Annahmen vorausgesetzt:

Fehlerfreiheit der Steuerungshardware und -software: Es wird davon ausgegangen, daß die störungstolerante Steuerung selbst fehlerfrei ist. Bei der Steuerungshardware handelt es sich um ein getestetes, qualitativ hochwertiges System, das keine internen Störungen aufweist. Weiterhin wird angenommen, daß die Steuerungssoftware aufgrund einer gewissen Softwarereifung keinerlei „Bugs“ enthält.

Nur eine Störungsursache: Theoretisch ist es zwar möglich, daß gleichzeitig mehrere, voneinander unabhängige Störungen in einem technischen System auftreten können. Da dies jedoch eher unwahrscheinlich ist, wird hier immer nur von einer Störungsursache ausgegangen. Alle damit verbundenen Störungen sind Symptome dieser einen Ursache.

5.3.2 Störungszustand

Auf der Basis der bereits erläuterten diskreten Zustände erfolgte zunächst die Beschreibung von Steuerungsanweisungen für den störungsfreien Betrieb. Sollen nun zusätzlich die Störungen, die im gesteuerten System auftreten können, berücksichtigt werden, sind sie ebenfalls in entsprechender Form anhand diskreter Zustände zu modellieren.

Grundsätzlich bestünde hier die Möglichkeit, für alle erfaßbaren Störungen jeweils einen eigenen Störungszustand zu spezifizieren. Problematisch ist jedoch die Vielzahl der möglichen Störungen und die Vielzahl der damit verbundenen Einträge im Zustandsgraphen. Dies ginge auf Kosten der Übersichtlichkeit. Des weiteren könnten im Betrieb auch unbekannte Störungen auftreten, die keinem bestimmten Störungszustand zugeordnet werden können. Deshalb wird im Rahmen dieser Arbeit der **Störungszustand „S“** zunächst als Sammelzustand für verschiedene Störungen definiert, ohne daß auf eine detailliertere Modellierung der tatsächlichen Störung eingegangen würde (Abbildung 5-6).

Im Unterschied zu den Idle- und Running-Zuständen, die reale, räumliche Lagen einer Funktionseinheit symbolisieren, ist der Störungszustand eine abstrakte Beschreibung von Situationen, in denen das gewünschte Soll-Verhalten des gesteuerten Systems nicht mehr vorliegt.

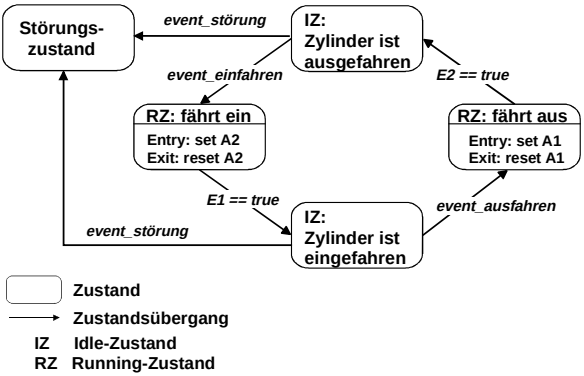


Abbildung 5-6: Der Störungszustand

Zur Laufzeit wird der Störungszustand durch die verschiedenen Störungserkennungsmethoden, die die störungstolerante Steuerung bereitstellt, erreicht. Das Verlassen des Störungszustands ist nur mit Hilfe bestimmter Störungsbehebungsmethoden möglich. Die folgenden Kapitel über die einzelnen Phasen der Störungsbehandlung beschreiben detaillierter, wie dieser Zustand erreicht und wieder verlassen wird.

5.3.3 Störungserkennung

Wie bereits in Kapitel 2.5 beschrieben, hängt das Verfahren der Störungserkennung sehr stark von den betrachteten Störungen ab. Für Störungen, die in der Schaltfunktionalität eines technischen Systems auftreten können, gibt es in der Literatur bereits zahlreiche Erfassungsverfahren, die auch in den Arbeiten von *Schwager (1985), Schönecker (1992), Schneider (1994) oder Wagner (1997)* verwendet werden. Das Referenzkonzept muß die wesentlichen dieser Verfahren enthalten. Im folgenden werden diese Verfahren erläutert. Gleichzeitig wird aufgezeigt, wie diese Verfahren in Steuerungsanweisungen auf Basis von UML-Zustandsgraphen integriert werden können.

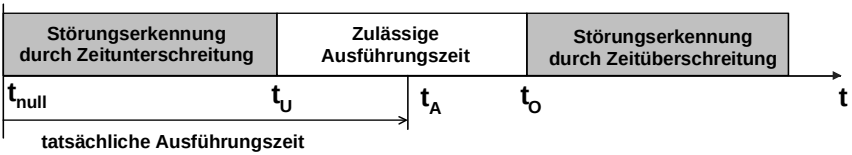
5.3.3.1 Zeitüberwachungsmethode

Die wohl gängigste und einfachste Methode der Störungserkennung bei Abläufen ist die Zeitüberwachung einzelner Aktionen. In der Literatur sind zahlreiche Arbeiten zu finden, die diese Methode zur Störungserkennung verwenden. Auch die meisten am

Markt erhältlichen Ablaufsteuerungen unterstützen bereits diese Methode der Störungserkennung *(Grötsch & Seubert 1997).*

Bei der Zeitüberwachung wird einer Aktion ein bestimmtes Zeitintervall zugeordnet und die benötigte Ausführungszeit überwacht. Dauert die Aktion länger als das vorgegebene Zeitintervall, wird davon ausgegangen, daß eine Störung aufgetreten ist.

In einigen Ansätzen wird sowohl eine obere als auch eine untere Zeitgrenze für eine Aktion gesetzt *(Schwager 1985, S. 47ff., Grimm 1987, S. 48).* Nur wenn die Ausführungsdauer einer Aktion innerhalb dieser Grenzen liegt, wird von einer störungsfreien Aktion ausgegangen. Liegt die Ausführungszeit aber außerhalb dieses Intervalls, egal ob darunter oder darüber, wird vom Auftreten einer Störung ausgegangen (Abbildung 5-7).



t_{null} = Startzeit einer Aktion
$t_U - t_{null}$ = Mindestdauer zur Ausführung einer Aktion
$t_O - t_{null}$ = maximal zulässige Dauer für eine Aktion

Abbildung 5-7: Störungserkennung mittels Zeitüberwachung

Da mit der Zeitunterschreitung nur wenige außergewöhnliche Störungen, wie z. B. der Wegfall einer Belastung aufgrund eines Bruches in der Aktorik, erfaßt werden, soll im Rahmen dieser Arbeit nur die Störungserkennung durch Zeitüberschreitung betrachtet werden. In Zusammenhang mit der Beschreibung von Steuerungsanweisungen auf Basis von Idle- und Running-Zuständen ist diese Methode der Störungserkennung nur für Running-Zustände interessant.

Zur Darstellung von Überwachungszeiten in Zustandsgraphen dienen sogenannte **Störungserkennungstransitionen**, die in den Störungszustand münden. Diese Transitionen besitzen Weiterschaltbedingungen, die nach einem definierten Zeitintervall erfüllt sind. Dieses Zeitintervall beginnt mit dem Aktivieren des entsprechend zu überwachenden Zustands (Abbildung 5-8).

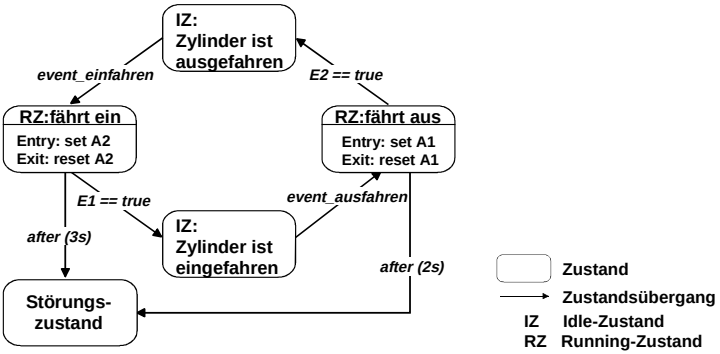


Abbildung 5-8: Zeitüberwachung in Zustandsgraphen

5.3.3.2 Signalüberwachungsmethode

Eine zweite gängige Methode zur Erfassung von Störungen bei Schaltfunktionen ist die Überwachung von Gebersignalen. Diese Erkennungsmethode basiert auf der Annahme, daß zu bestimmten Zeitpunkten binäre Eingangswerte nur bestimmte Werte oder Wertmuster annehmen dürfen. Auch diese Methode wird von der störungstoleranten Steuerung unterstützt. So ist beispielsweise das gleichzeitige Auftreten des Gebersignales für „Zylinder ist ausgefahren" (E1) und „Zylinder ist eingefahren" (E2) (siehe Abbildung 5-5) nicht zulässig und führt zu der Annahme, daß eine Störung vorliegt. Ursache könnte beispielsweise ein Kurzschluß in einem der beiden Geberstränge sein.

Die Überprüfung von Eingangssignalen bzw. deren Kombinationen kann auf zwei Arten erfolgen. Bei der „temporären Überwachung“ werden nur während der Ausführung bestimmter Dienste oder während der Aktivierung einzelner Zustände gezielte Signalkombinationen überwacht. Anders ist dies bei der „permanenten Überwachung“. Hier werden ganz bestimmte Signalkombinationen unabhängig vom aktiven Zustand eines Zustandsgraphen kontinuierlich auf Störungsfreiheit untersucht. Ein Beispiel hierzu ist die oben erläuterte Überwachung der beiden Eingangssignale (E1, E2).

Analog zur Zeitüberwachungsmethode erfolgt die Beschreibung der Störungserkennung mittels Signalüberwachung ebenfalls durch Störungserkennungstransitionen. Die zu überwachenden Signalkombinationen sind die Weiterschaltbedingung dieser Transitionen. In Abbildung 5-9 ist dieser Sachverhalt am Beispiel des modellierten Zylinders dargestellt. Hier werden beide Idle-Zustände

temporär überwacht. Gleichzeitig erfolgt eine permanente Überwachung der Idle- und Running-Zustände.

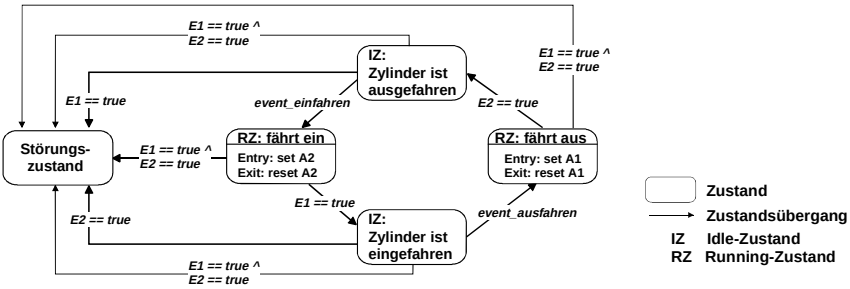


Abbildung 5-9: Erfassung von Störungen mittels Signalüberwachung

5.3.3.3 Komponentenüberwachungsmethode

Diese Art der Störungserkennung ähnelt der Signalüberwachung. Der Unterschied liegt jedoch darin, daß hier die Störung nicht aktiv vom aktuellen Steuerungsbaustein aus erkannt wird, sondern bereits in einer anderen Komponente erfaßt wurde und diese Information nun an den Steuerungsbaustein weitergeleitet wird.

Bei den Komponenten kann es sich zum einen um weitere Steuerungsbausteine handeln, die während einer Dienstausführung eine Störung erfassen und deren Behandlung weiterleiten. Zum anderen kann eine Komponente aber auch ein elektromechanisches Element sein, das eigenständig Diagnoseinformationen generiert und diese weiterleitet. Ein Beispiel hierfür ist ein intelligenter, induktiver Näherungsschalter mit integrierter Überwachungsfunktion *(Schell 1997).* Solche Näherungsschalter verfügen über einen sogenannten Diagnoseausgang (Diagnose-Bit), mit dem Störungen wie z. B. Spulendrahtbruch, Leitungsbruch oder Sensorelektronikausfall erfaßt werden können. Liegt im Betrieb kein Signal auf dem Diagnoseausgang, ist dies ein Zeichen für eine Störung im Signalgeber.

Wie diese Störungserkennungsmethode in die zustandsgraphenbasierte Erstellung von Steuerungsanweisungen integriert werden kann, zeigt Abbildung 5-10.

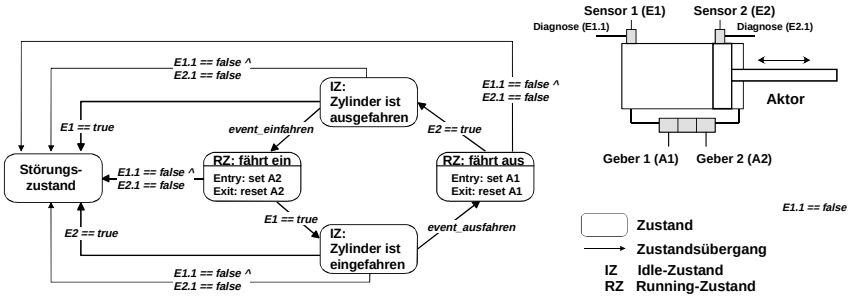


Abbildung 5-10: Erfassung von Störungen mittels Komponentenüberwachung

5.3.3.4 Weitere Ansätze

In der Literatur sind noch weitere Verfahren zur Erfassung von Störungen bei Schaltfunktionen zu finden, wie z. B. die Antivalenzüberwachung oder die Überwachung kurzzeitiger Fehlsignale *(Grimm 1987, S. 44ff.).* Diese speziellen Verfahren sollen hier jedoch nicht weiter betrachtet werden, da sie zum großen Teil mit den obigen Verfahren abgedeckt werden können.

5.3.4 Störungslokalisierung

Mit der Erfassung einer Störung in einem technischen System muß das erkannte Symptom nicht immer eindeutig auf eine entsprechende Störungsursache hindeuten. Oftmals können mehrere Störungsursachen dasselbe Symptom hervorrufen. Ein klassisches Beispiel hierfür ist das Symptom der Zeitüberschreitung an einer Funktionseinheit. Diese Störung kann an einem Defekt in der Aktorik, den Stellgliedern oder den Signalgebern liegen (Abbildung 5-11). Für eine umfassende Störungsbehandlung ist deshalb nach der Störungserkennung eine genaue Lokalisierung der Störungsursache notwendig.

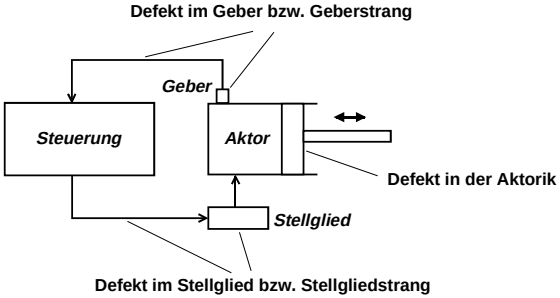


Abbildung 5-11: Störungsursachen an einer Funktionseinheit

Zur Lokalisierung von Störungen in technischen Systemen gibt es in der Literatur unterschiedliche Verfahren (siehe auch Kapitel 2.6.2). Im vorliegenden Referenzkonzept werden in Anlehnung an *Härdtner (1992, S. 16)* folgende drei Verfahren für die Ursachenfindung von Störungen in der Schaltfunktionalität von Maschinen unterstützt:

- Lokalisierung durch Symptom-Ursachen-Beziehung,
- Lokalisierung durch Kombinatorik und
- Lokalisierung durch schrittweise Eingrenzung.

Folgende Abschnitte erläutern diese Verfahren im einzelnen:

5.3.4.1 Lokalisierung durch Symptom-Ursachen-Beziehung

Dieses Verfahren beruht auf dem Grundgedanken, daß eine bestimmte Störungsursache zu einem eindeutigen Störungssymptom führt. Umgekehrt läßt dies darauf schließen, daß ein bestimmtes Symptom mit größter Wahrscheinlichkeit durch eine individuelle Störung entsteht. Ein Beispiel hierfür sind Störungen in einem induktiven Näherungsschalter mit integrierter Überwachungsfunktion (Abbildung 5-12). So läßt das Ausbleiben des Diagnose-Bits darauf schließen, daß im entsprechenden Geber eine Störung vorliegt bzw. daß dieser defekt ist. Dies setzt natürlich voraus, daß die Leitung des Überwachungsbits, d. h. die Strecke vom Geber bis zum Eingang in die Steuerung, keine Störungen aufweist.

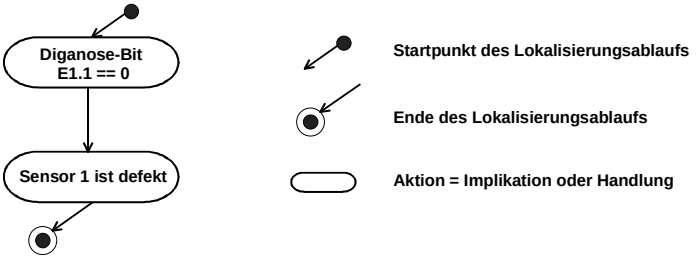


Abbildung 5-12: Einfache Lokalisierung durch Ursachen-Symptom-Beziehung

5.3.4.2 Lokalisierung durch Kombinatorik

Anders als in der ersten Methode werden bei der Lokalisierung durch Kombinatorik mehrere Symptome zur Ermittlung der Störungsursache verwendet. Die kombinatorische Verknüpfung dieser Symptome läßt dann auf die Störungsursache schließen. Wichtig für diese Art der Störungslokalisierung ist die Kenntnis der Beziehungen zwischen den einzelnen Elementen des zu steuernden Systems.

Folgendes Beispiel erläutert eine Anwendung dieser Art der Ursachenfindung. Beim Ausfahren des Zylinders einer einfachen Achse wird mittels Zeitüberwachung eine Störung festgestellt. Zur genauen Ermittlung der Störungsursache werden nun die Signalwerte der beiden Endlagengeber herangezogen. Meldet der Geber der Ausgangslage nach wie vor ein Signal, während der Geber der Endlage keines meldet, kann davon ausgegangen werden, daß die Ursache der Störung in der Aktorik oder dem Stellglied liegt, nicht aber im Geberstrang. Abbildung 5-13 verdeutlicht nochmals diese Schlußfolgerung.

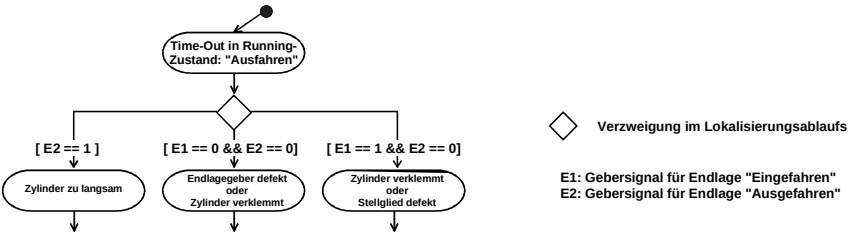


Abbildung 5-13: Lokalisierung durch Kombinatorik

5.3.4.3 Lokalisierung durch schrittweise Eingrenzung

Zur detaillierteren Beschreibung von Störungsursachen ist jedoch in der Regel die Methode der schrittweisen Eingrenzung erforderlich. Diese geht wie die kombinatorische Methode davon aus, daß zur genauen Lokalisierung der Störungsursache mehrere Symptome betrachtet werden müssen. Diese Symptome können aber nicht alle auf einmal verglichen werden. Vielmehr wird hier in einem iterativen Vorgehen nach weiteren Symptomen bzw. Merkmalen gesucht, um schließlich die eigentliche Störungsursache zu finden. Dabei orientiert sich die Suche an der mechanischen Struktur des gesteuerten Systems, d. h. an den Elementen, die eine Funktionseinheit bilden bzw. mit ihr in Verbindung stehen, oder den Baugruppen, die zu einem Aggregat gehören.

Als Beispiel soll hierfür nochmals die Störung beim Ausfahren des Zylinders betrachtet werden. Konnte z. B. die Störungsursache anhand der Endpositionsgeber nicht eindeutig erfaßt werden, können in einem zweiten Schritt für bestimmte Fälle Angaben über den Öldruck herangezogen werden, um zu erfassen, ob die Störung an der Energieversorgung liegt oder ob der Zylinder selbst defekt ist (Abbildung 5-14).

In vielen Fällen reichen jedoch die Angaben aus der Steuerung, d. h. die Angaben, die mittels Gebern erfaßt und an die Steuerung weitergeleitet werden, für eine detailliertere Beschreibung der Störungsursache nicht aus. Informationen darüber, ob ein Aktor tatsächlich ausgefahren wurde oder ob ein bestimmter induktiver Näherungsschalter auch wirklich aktiv ist, können meist nur vom Bediener erfaßt werden. Aber gerade diese Informationen sind für die genaue Feststellung der Störungsursachen von wesentlicher Bedeutung. Deshalb verfügt das Referenzkonzept zur Detaillierung der Störungslokalisierung über Mechanismen, die der Erfassung von Bedienerinformationen dienen und diese in den Lokalisierungsablauf integrieren können. Abbildung 5-14 zeigt, wie die Bedienerangaben über die Position des Aktors der weiteren Ursachenfindung einer Störung dienen können.

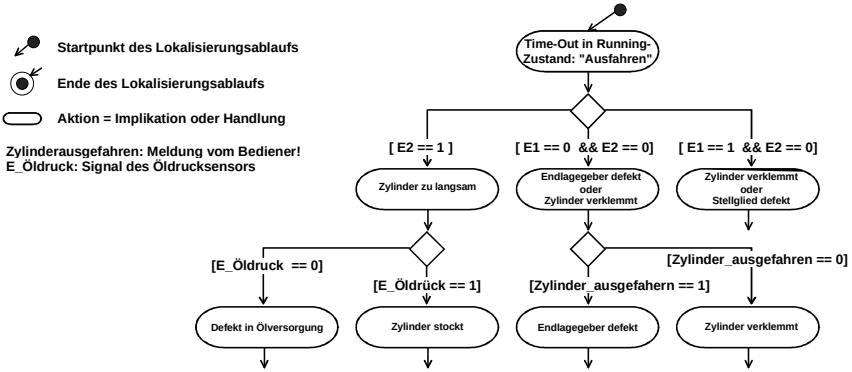


Abbildung 5-14: Lokalisierung durch schrittweise Eingrenzung

5.3.4.4 Auswahl des geeigneten Verfahrens

Das hier entwickelte Referenzkonzept einer störungstoleranten Steuerung unterstützt, wie bereits beschrieben, die drei Varianten der Störungslokalisierung. Bei der Erfassung von Störungen mittels der direkten Komponentenüberwachung liegt es nahe, die Ursachenfindung nach der Methode der Lokalisierung durch Symptom-Ursachen-Beziehung durchzuführen. In einigen Fällen kann diese Methode auch bei der Zustandsüberwachung eingesetzt werden. Die Störungserkennung durch Zeitüberwachung erfordert jedoch ein ausführlicheres Vorgehen und verlangt in der Regel eine schrittweise Eingrenzung der Störungsursache.

5.3.4.5 Detaillierungsgrad der Ursachenfindung

Durch die Störungslokalisierung wird versucht, die genaue Störungsursache ausfindig zu machen. Gleichzeitig wird eine Auskunft über den Zustand des Produktionssystems nach der Störung eingeholt. Durch die Gliederung der kompletten Steuerungsaufgabe in einzelne Steuerungsbausteine, die sich an den Funktionseinheiten des technischen Systems orientieren, beziehen sich die Ergebnisse des Störungslokalisierungsvorgangs in der Regel auch auf diese Funktionseinheiten.

Sinn und Zweck der Störungstoleranz ist grundsätzlich die Fortführung gestellter Aufgaben trotz des Auftretens von Störungen *(Koch 1996, S. 33)*. Im Vergleich zu den zahlreichen Ansätzen, die sich mit der Thematik der Diagnose in Produktionssystemen befassen und dabei eine sehr detaillierte Ursachenfindung von Störungen vornehmen, steht hier der Behebungs- und Reaktionsvorgang nach einer

Störung im Vordergrund. Das Ergebnis der Störungslokalisierung muß hier nur so detailliert sein, daß anschließend die richtige Strategie zur Reaktion und Behebung der Störung gewählt werden kann.

Konnte beispielsweise durch die Störungslokalisierung ermittelt werden, daß die Ursache für die Zeitüberschreitung beim Ausfahren eines Zylinders am fehlenden Endlagegebersignal liegt, so reicht diese Information aus, um die jeweils richtige Reaktionsstrategie zu wählen. Diese könnte unter Umständen das Ignorieren des benötigten Signals sein, wenn die Funktionseinheit ihre Soll-Position tatsächlich erreicht hat. Eine genauere Angabe darüber, ob das fehlende Signal auf einen Kabelbruch, einen Wackelkontakt oder einen defekten Sensor zurückzuführen ist, wird hier nicht untersucht.

5.3.5 Störungsbehebung

Die letzte Phase der Störungsbehandlung ist, wie bereits in Kapitel 2.6.1 beschrieben, die Störungsbehebung. Hier erfolgt zum einen eine Beseitigung der aufgetretenen Störungsauswirkungen und -ursachen. Zum anderen muß hier auch das Wiederaufsetzen des normalen Betriebs erfolgen. Im Vergleich zu zahlreichen steuerungstechnischen Ansätzen zur Störungsbehandlung (siehe Kapitel 3) ist im Rahmen dieser Arbeit die Störungsbehebung von besonderer Bedeutung.

Bevor im folgenden eine detaillierte Beschreibung der Funktionsweise der Störungsbehebung im hier entwickelten Referenzkonzept erfolgt, wird zuerst erläutert, welche Methoden der Störungsbehebung insbesondere aus steuerungstechnischer Sicht von Bedeutung sind.

5.3.5.1 Methoden zur Störungsbehebung

Ein grundlegendes Problem der Störungsbehebung ist die Vielzahl der denkbaren Reaktionsmöglichkeiten, das geringe Potential zu deren Vollautomatisierung sowie die Schwierigkeit, ein einheitliches Schema zur Störungsbehebung zu beschreiben.

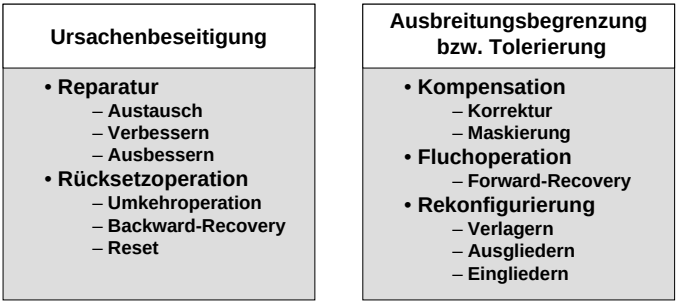


Abbildung 5-15: Methoden zur Störungsbehebung in Anlehnung an Schneider (1994, S. 35)

Eine Zusammenfassung der wesentlichen Methoden zur Störungsbehebung gibt Abbildung 5-15 wieder. Diese können in die zwei Wirkungsklassen Ursachenbeseitigung und Ausbreitungsbegrenzung bzw. Tolerierung gegliedert werden.

Inwiefern und wann diese Methoden eingesetzt werden können, hängt in erster Linie von Art und Ursache der Störung ab. Für eine störungstolerante Steuerung nicht unwesentlich sind auch die verschiedenen Störungsbehebungsmethoden im Zusammenhang mit den Zuständen eines gestörten technischen Systems. In Abbildung 5-16 sind die wesentlichen Varianten der Störungsbehebungsmethoden aus steuerungstechnischer Sicht dargestellt.

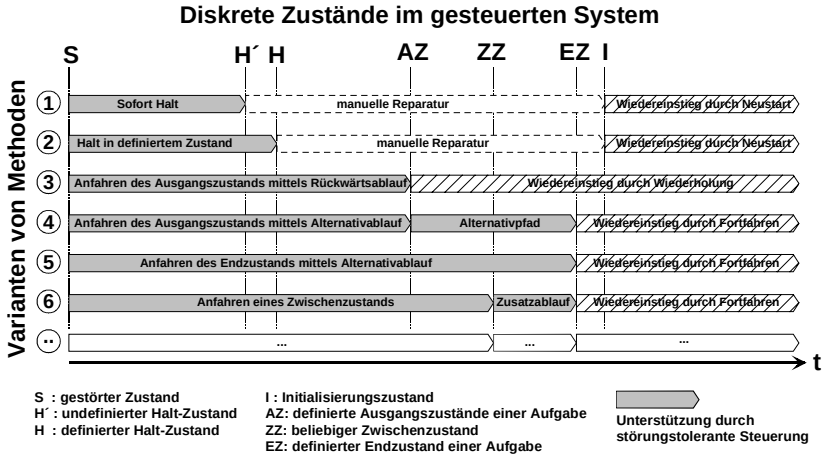


Abbildung 5-16: Methoden zur Störungsbehebung aus steuerungstechnischer Sicht

Die wohl einfachste Methode der Störungsbehebung ist das sofortige Anhalten des gestörten Systems in einem beliebigen Zustand (Abbildung 5-16/1). Dieser Zustand hängt von der Störungsursache und dem Zustand zum Zeitpunkt der Störungserkennung ab. Vergleichbar ist dieses Vorgehen mit dem klassischen „NOT-AUS“ oder dem Soforthalt. Um das System erneut zu starten, ist hier meist ein manuelles Eingreifen erforderlich, das auch mit einer Reparatur verbunden sein kann. Zum Neustarten muß das System in einen definierten Initialisierungszustand gebracht werden.

Eine andere Störungsbehebungsmethode ist das Anhalten des Systems in einem definierten Zustand, vorausgesetzt, das System kann noch bis zum definierten Zustand weiterfahren (Abbildung 5-16/2). Als Beispiel sei das Anhalten zum Taktende ober bei der Beendigung einer bestimmten Bearbeitung zu nennen. Vorteilhaft an dieser Methode ist zum einen der leichtere Zugang zu sonst unzugänglichen Bereichen, besonders dann, wenn Reparaturarbeiten in Bereichen notwendig sind, die eine Gefahrenzone für den Bediener darstellen. Zum anderen vereinfachen die definierten Halte-Zustände das Anfahren des Initialisierungs-zustands, von dem aus mit dem Betrieb fortgefahren wird.

In zahlreichen Fällen reicht zur Störungsbehandlung jedoch eine einfache Wiederholung der gestellten Aufgabe (Abbildung 5-16/3). Dazu muß das System lediglich in einen Ausgangszustand gebracht werden, von dem aus der fehlgeschlagene Vorgang nochmals wiederholt wird. Die Wiederholung stellt dann den Wiedereinstieg dar. Diese Methode der Störungsbehebung wird auch als „Backward-Recovery“ bezeichnet *(Hofmann 1990, S. 43).* Als Beispiel sei hier ein fehlgeschlagener Greifvorgang erwähnt, der durch eine einfache Wiederholung korrigiert wird.

Führt dagegen die einfache Wiederholung nicht zum Ziel, ist es denkbar, im Rahmen der Störungsbehebung einen alternativen Ablauf mit Hilfe einer redundanten Komponente durchzuführen. So wird das System vom angefahrenen Ausgangszustand des gewünschten Ablaufs in den entsprechenden Endzustand geführt (Abbildung 5-16/4). Auf diese Weise kann z. B. ein Transportvorgang von A nach B, der aufgrund eines fehlgeschlagenen Greifvorgangs mehrmals nicht ausgeführt werden konnte, kurzfristig komplett vom Bediener übernommen werden. Nach Erreichen des Endzustands wird der normale Betrieb dann wieder eingeleitet.

Eine weitere Methode der Störungsbehebung sieht die Überführung des fehlgeschlagenen Vorgangs direkt in einen Endzustand vor (Abbildung 5-16/5, 6). Dieses Vorgehen ist auch als „Forward-Recovery“ bekannt *(Hofmann 1990, S. 43)*. Hierbei wird entweder die aufgetretene Störung bewußt ignoriert und der gewünschte Endzustand weiter angefahren (Abbildung 5-16/5), oder es werden weitere Zwischenabläufe eingeleitet, mittels derer das System dann auch den gewünschten Endzustand erreicht (Abbildung 5-16/6). In beiden Fällen erfolgt die Wiederaufnahme

des normalen Betriebs vom entsprechenden Endzustand aus. Ein Beispiel für den ersten Fall ist das Fehlen eines Gebersignals, das durch ein Signal vom Bediener simuliert wird. Ein Beispiel für den zweiten Fall ist eine verklemmte Hebetür, die mit Hilfe des Bedieners in den gewollten Endzustand gebracht wird.

Bevor im folgenden erläutert wird, wie die störungstolerante Steuerung diese unterschiedlichen Methoden unterstützen kann und wie ein einheitliches Schema dafür aussieht, soll zunächst noch auf die Reversibilität von Aktivitäten in Produktionssystemen eingegangen werden.

5.3.5.2 Reversibilität von Abläufen

Im Zusammenhang mit der Behebung von Störungen spielt die Reversibilität von Abläufen eine wesentliche Rolle. Dies ist besonders dann der Fall, wenn zur Störungsbehebung ein definierter Ausgangszustand angefahren werden soll.

In Anlehnung an *Hofmann (1990, S. 80-81)* lassen sich grundsätzlich drei Reversibilitätsklassen für Abläufe bilden (Abbildung 5-17):

- unmittelbar reversibel,
- mittelbar reversibel und
- irreversibel.

Während die verschiedenen abtragenden Fertigungstechnologien, wie z. B. Bohren, Drehen oder Fräsen, in bezug auf ein bearbeitetes Werkstück immer irreversible Vorgänge darstellen, unterscheiden sich die Technologien der Montage hinsichtlich ihrer Reversibilität. So können Schraubvorgänge leicht rückgängig gemacht werden, während beispielsweise das Schweißen ein irreversibler Vorgang ist.

In bezug auf ein Werkstück werden Handhabungs- und Transportvorgänge den ersten beiden Kategorien zugeordnet. Die meisten Handhabungs- und Transportvorgänge sind zwar reversibel, schwierig wird es aber, wenn dabei externe Kräfte, wie z. B. die Gravitation, maßgeblich am Vorgang beteiligt sind. So ist beispielsweise ein Transportvorgang mittels Rutschbahnen nur mittelbar reversibel.

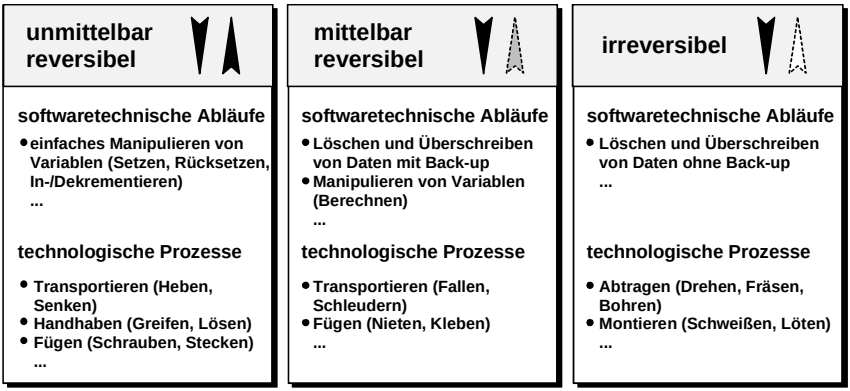


Abbildung 5-17: Reversibilitätsklassen von Abläufen und Prozessen

Da die hier entwickelte störungstolerante Steuerung in erster Linie der Steuerung der Schaltfunktionalität von Produktionssystemen dient, ist die Umkehrbarkeit von Abläufen in bezug auf die Zustände des Produktionssystems selbst von entscheidender Bedeutung. Aufgrund der Tatsache, daß diese Abläufe in der Regel auf einfache, reversible Bewegungsschritte zurückgeführt werden können, sind die meisten unmittelbar reversibel. Wichtig für die Reversibilität ist jedoch, daß bei den Abläufen keine Materialschäden am Produktionssystem selbst, wie z. B. Brüche, entstanden sind.

5.3.5.3 Die Reaktionsstrategie

In Abbildung 5-16 wurden unterschiedliche Methoden zur Störungsbehebung in Zusammenhang mit den Zuständen eines gestörten technischen Systems aufgezeigt. Außer bei den Behebungsmethoden, die ein völliges Anhalten des gestörten Systems erfordern, sieht das Referenzkonzept vor, den gesamten Ablauf der Störungsbehandlung auszuführen, d. h. vom Zeitpunkt der eindeutig erfaßten Störungsursache bis hin zum Wiedereinsetzen des Betriebs.

Die verschiedenen Methoden zur Störungsbehebung, die in Kapitel 5.3.5.1 betrachtet wurden, lassen sich anhand eines einheitlichen Schemas beschreiben, das hier als **Reaktionsstrategie** bezeichnet werden soll. Die Reaktionsstrategie kann somit ein sofortiger Stillstand in einem definierten Zustand sein, um eine manuelle Reparatur zu unterstützen. Die Strategie kann aber auch das Umgehen eines defekten Elements und die einfache Fortführung der Steuerungsaufgabe sein, ungeachtet der Tatsache, daß das technische System nun nicht mehr voll funktionsfähig ist.

Schematisch besteht eine Reaktionsstrategie aus den zwei Zuständen

- **Wiedereinstiegspunkt (WEP)** und
- **Wiederaufsetzpunkt (WAP)**

sowie den beiden Abläufen

- **Beseitigungsvorgang** und
- **Recovery-Vorgang** (Abbildung 5-18).

Was sich hinter diesen vier Elementen verbirgt, wird im folgenden erläutert:

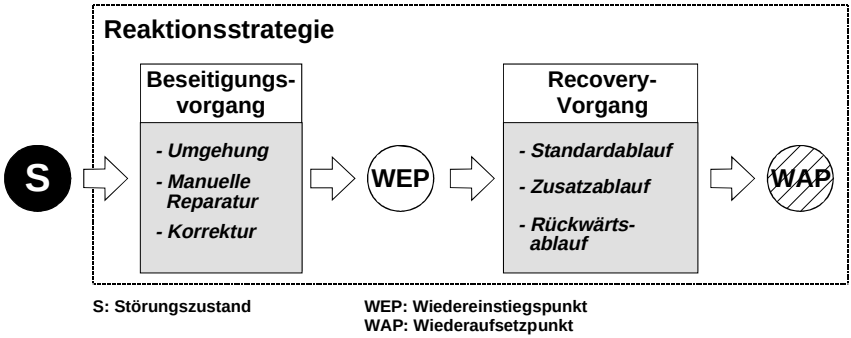


Abbildung 5-18: Schema der Reaktionsstrategie

Der Wiedereinstiegspunkt

Das erste wesentliche Element in einer Reaktionsstrategie ist der sogenannte Wiedereinstiegspunkt. Er beschreibt den Zustand, in dem das reale, zuvor gestörte System und das in der Steuerung abgebildete Zustandsmodell für den störungsfreien Ablauf wieder synchron sind. Denn bei einer Beschreibung von Steuerungsanweisungen mit Hilfe diskreter Zustände muß der Zustand des realen Systems nach Störungseintritt nicht immer mit dem in der Steuerung abgelegten Modell übereinstimmen. Dies ist besonders dann der Fall, wenn für alle Störungsfälle am technischen System nur ein einziger Störungszustand spezifiziert ist. Nach Auftreten einer Störung ist in der Steuerung der Störungszustand aktiv. Das zugehörige reale System kann sich aber in einem beliebigen bzw. auch in einem unbekannten Zustand befinden. Die sogenannte Integrität zwischen Realität und Abbild ist dann nicht mehr gegeben (siehe auch Kapitel 5.2.2). Aufgabe der Störungsbehebung ist es, diese Diskrepanz zwischen den Zuständen zu beseitigen. Bestehende Ansätze definieren einen Ausgangs- oder Initialisierungspunkt für das

technische System, der nach dem Auftreten einer Störung manuell angefahren wird. Nur durch ein erneutes Hochfahren der Steuerungen können der reale Maschinenzustand und der Zustand der Steuerung vereinheitlicht werden. Mit Hilfe der hier definierten Wiedereinstiegspunkte können dagegen beliebige Zustände spezifiziert werden, von denen aus der Maschinen- und Steuerungszustand wieder in Übereinstimmung gebracht werden können, damit die Integrität gegeben ist.

Wurde beispielsweise beim Ausfahren eines Zylinder eine Störung erfaßt, die auf einen defekten Signalgeber zurückzuführen ist, so befindet sich der Zylinder bereits im ausgefahrenen Zustand. Die Steuerung befindet sich aber im Störungszustand. Durch die Spezifikation des Zustandes „Zylinder ist ausgefahren“ als Wiedereinstiegspunkt können der Zustand der Steuerung und der des gesteuerten Systems sehr einfach wieder in Einklang gebracht werden, ohne daß man den Initialisierungszustand anfahren müßte.

Einen Spezialfall stellen Situationen dar, bei denen das gesamte technische System sofort angehalten werden muß (siehe auch Abbildung 5-16/1). Hier ist eine grundlegende Initialisierung unabdingbar. In solchen Fällen kann als Wiedereinstiegspunkt der Haltezustand angegeben werden. Das ist ein Sonderzustand in der Steuerung, der keine Aktionen enthält und nur über eine Initialisierung der Steuerung verlassen werden kann.

Der Beseitigungsablauf

Um den Wiedereinstiegspunkt zu erreichen, ist meistens eine **Beseitigung** der vorliegenden Störung erforderlich (Abbildung 5-18). Darunter ist in erster Linie die Beseitigung von Störungssymptomen oder Störungsauswirkungen zu verstehen. Ob dabei auch die Störungsursache tatsächlich behoben oder aber nur umgangen, d. h. partiell beseitigt *(Wagner 1997, S. 15),* wird, hängt jeweils vom konkreten Störungsfall ab. Wurde die Störung beispielsweise aufgrund eines fehlenden Gebersignals festgestellt, kann das ausgefallene Gebersignal durch eine definierte Bedienereingabe ersetzt und der Betrieb fortgeführt werden. Dadurch wird zwar das Störungssymptom, d. h. der durch das ausgefallene Gebersignal verursachte Stillstand, behoben; die eigentliche Störungsursache, z. B. das aufgrund eines defekten Gebers fehlende Gebersignal, bleibt jedoch bestehen. Da aber das Hauptziel der störungstoleranten Steuerung die Durchführung gestellter Aufgaben trotz des Auftretens von Störungen ist, reicht diese Störungsbeseitigung kurzfristig gesehen völlig aus.

Zur Beseitigung der Störungen kann eine Reaktionsstrategie folgende Aktivitäten beinhalten:

- Umgehungsaktivitäten: Hiermit werden bestimmte Schritte unternommen, um eine Störung bewußt zu ignorieren bzw. zu umgehen.

- Anleitungen zu manuellen Reparaturen: Dies sind konkrete Anweisungen, die sich an den Bediener richten und ihn bei der Durchführung von Reparaturarbeiten anleiten. Dies können Anleitungen zum Austausch defekter Teile sein. Es kann sich aber auch um Anweisungen handeln, konkrete Dienste, die aufgrund der Störung nicht ordnungsgemäß durchgeführt werden konnten, kurzfristig selbst zu übernehmen. All diese Anweisungen dienen dazu, den realen Zustand des gestörten Systems an den definierten Wiedereinstiegspunkt anzupassen.
- Korrekturen: Dies sind Aktivitäten, mit denen datentechnische Veränderungen in der Steuerung vorgenommen werden, um den Steuerungszustand an den definierten Wiedereinstiegspunkt anzupassen. Hier können z. B. Stückzahl oder Zustandsangaben vom Bediener erfaßt und der Steuerung zugewiesen werden.

Der Wiederaufsetzpunkt

In zahlreichen Fällen wird der Wiedereinstiegspunkt auch den Zustand beschreiben, von dem aus der normale Betrieb fortgesetzt werden kann. In einigen Situationen ist der Wiedereinstiegspunkt dafür jedoch nicht geeignet. Dies ist besonders dann der Fall, wenn an der fehlgeschlagenen Aktion mehrere Teilnehmer beteiligt waren und diese zuerst wieder synchronisiert werden müssen. Hierfür gibt es in der Reaktionsstrategie einen weiteren Punkt, der als Wiederaufsetzpunkt bezeichnet wird.

Kommt es beispielsweise im Produktionsbetrieb bei einem Transportvorgang von A nach B zu einer Störung, die auf einen verklemmten Greifer zurückzuführen ist, könnte der Wiedereinstiegspunkt jener Zustand sein, bei dem die Verklemmung behoben ist. Der Wiederaufsetzpunkt ist dagegen der Ausgangspunkt des Transportvorgangs.

Der Recovery-Vorgang

Um nach Erreichen des Wiedereinstiegpunktes einen gesonderten Wiederaufsetzpunkt anzufahren, ist ein **Recovery-Vorgang** notwendig. Dieser stellt den zweiten Ablauf in der Reaktionsstrategie dar und wird in der Regel alleine von der Steuerung, ohne Einbindung des Bedieners, durchgeführt.

Der Recovery-Vorgang kann einerseits ein bereits für den störungsfreien Betrieb verwendeter Prozeß sein, der im Rahmen der Störungsbehebung ebenfalls durchlaufen wird. Andererseits kann er aber auch ein zusätzlicher Vorgang sein, der nur im Störungsfall eingeleitet wird, wie z. B. ein Ablauf zum vorzeitigen Ausschleusen defekter Bauteile aus einem Bearbeitungsraum.

Ist der Wiederaufsetzpunkt der Ausgangszustand des fehlgeschlagenen Dienstes, kann dieser durch die Backward-Recovery-Methode erreicht werden. Hierzu spielt die Reversibilität von Abläufen eine wesentliche Rolle. Im Zusammenhang mit den

Diensten, die ein Steuerungsbaustein zur Verfügung stellt, werden zwei Arten von Backward-Recovery-Abläufen unterschieden:

- Rückwärtsdienste: Diese Dienste stellen Abläufe dar, die einen kompletten zur Verfügung gestellten Dienst rückgängig machen können. Ein Rückwärtsdienst ist ein eigener Dienst und wird immer vom selben Steuerungsbaustein zur Verfügung gestellt, der den Vorwärtsdienst anbietet. Als Beispiel sei hier der Rückwärtsdienst „Tür schließen“ genannt, der den Vorwärtsdienst „Tür öffnen“ rückgängig macht. Wichtig ist hier, daß beide Dienste vom Steuerungsbaustein zur Verfügung gestellt werden.
- Rückwärtspfade: Ein Rückwärtspfad ist eine Zusammenfassung einzelner Schritte zu einem großen Ablauf, der einen anderen Ablauf komplett oder nur zum Teil revidiert. Im Unterschied zum Rückwärtsdienst ist der Rückwärtspfad kein eigenständiger Dienst, sondern nur ein Teil eines Dienstes. Ein Beispiel für einen Rückwärtspfad ist ein Ablauf, der den Greifarm eines Werkzeugwechslers von einem ausgefahrenen Zustand in seine Parkposition zurückführt.

5.3.5.4 Einbettung der Reaktionsstrategie in eine zustandsorientierte Steuerungsanweisung

Nachdem im vorherigen Kapitel beschrieben wurde, aus welchen Elementen eine Reaktionsstrategie besteht, wird hier erläutert, wie im Referenzkonzept die Reaktionsstrategie in die zustandsorientierte Steuerungsanweisung eingebettet wird.

Anders als bei den Steuerungsabläufen, die während der Ausführung von Diensten erfolgen, erfordern die Beseitigungsabläufe der Reaktionsstrategie eine Interaktion mit dem Bediener. Diese Abläufe sind in erster Linie Dialogsequenzen, die Meldungen an den Bediener schicken und Eingaben von ihm fordern. Die Beschreibung dieser Dialogsequenz ist unabhängig von den Idle- und Running-Zuständen des technischen Systems.

Wie eine komplette Reaktionsstrategie mit ihren vier Elementen (Beseitigungsablauf, Wiedereinstiegspunkt, Recoveryablauf und Wiederaufsetzpunkt) in die zustandsorientierte Beschreibung einer Steuerungsanweisung erfolgt, zeigt Abbildung 5-19. Hier ist der Beseitigungsablauf ein eigenständiger Ablauf, der von der Steuerungsanweisung losgelöst ist. Dagegen sind die beiden Punkte Wiedereinstiegspunkt und Wiederaufsetzpunkt Zustände des gesteuerten Systems und somit Zustände im entsprechenden Zustandsgraph. Hier kann der Wiedereinstiegspunkt jeden beliebigen Zustand des Systems einnehmen, während der Wiederaufsetzpunkt immer ein Idle-Zustand sein muß, von dem aus ein Dienst gestartet wird. Der Recovery-Ablauf ist die Kette der Running-Zustände, die zwischen dem Wiedereinstiegs- und Wiederaufsetzpunkt liegen.

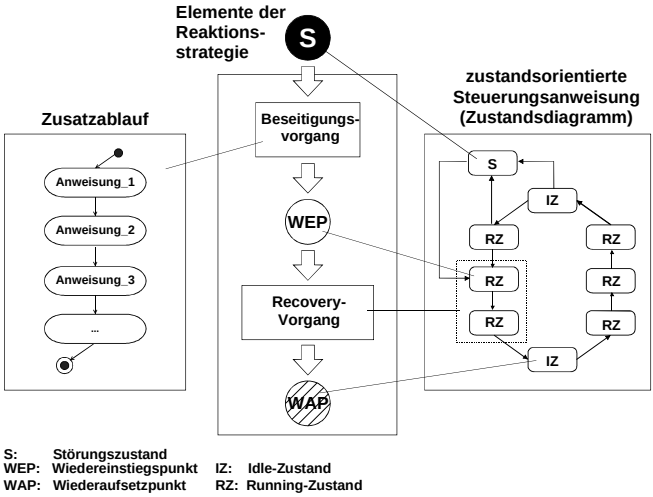


Abbildung 5-19: Einbettung einer Reaktionsstrategie in die zustandsorientierte Steuerungsanweisung

5.3.5.5 Auswahl der Reaktionsstrategie

Im Referenzkonzept der störungstoleranten Steuerung wird jeder in der Steuerung bekannten Störungsursache mindestens eine Reaktionsstrategie zugeordnet. Kommt es während der Laufzeit zu einer Störung, wird von der Steuerung im Rahmen der Störungsbehebung die am besten geeignete Reaktionsstrategie ausgewählt und anschließend abgearbeitet. Dabei erfolgt die Auswahl der einzuleitenden Reaktionsstrategie zunächst in Abhängigkeit von den Ergebnissen der Störungslokalisierung.

Bei Störungsursachen mit jeweils nur einer zugehörigen Reaktionsstrategie ist diese Zuordnung eindeutig. Stehen jedoch mehrere Reaktionsstrategien zur Verfügung, ist ein entsprechendes Auswahlverfahren erforderlich. Dieses kann beispielsweise auf Basis von Erfolgschancen einer Reaktionsstrategie oder auch mit Hilfe der Historie der Störungsbehandlung erfolgen. So könnte beispielsweise auf eine neue Strategie zurückgegriffen werden, wenn die zuerst gewählte Strategie mehrmals fehlgeschlagen ist.

Da der Bediener im Rahmen der Störungsbehebung in den meisten Fällen unabdingbar ist, kann er auch als Entscheidungsträger zur Auswahl der richtigen Strategie mit einbezogen werden. Wichtig ist nur, daß hierzu die notwendige Information richtig aufbereitet und dem Bediener verständlich vermittelt wird.

5.4 Der störungstolerante Steuerungsbaustein

Wie bereits zu Beginn dieses Kapitels beschrieben, ist die störungstolerante Steuerung modular aufgebaut. Sie besteht aus den sogenannten Steuerungsbausteinen, die verschiedene Dienste anbieten und zur Steuerung von Funktionseinheiten, Baugruppen oder gar kompletten Aggregaten eingesetzt werden. Das Wesentliche an den Bausteinen ist die Störungstoleranz. Dazu verfügt jeder Baustein über eigene Funktionen und Mechanismen zur Störungserkennung, Störungslokalisierung und Störungsbehebung.

Wie diese Steuerungsbausteine aufgebaut sind und wie sie mit Hilfe diskreter Zustände komplette Störungsbehandlungsabläufe durchführen, wird im folgenden erläutert.

5.4.1 Aufbau

Zur Duchführung der unterschiedlichen, oben erläuterten Störungsbehandlungsmethoden verfügt jeder Steuerungsbaustein zusätzlich zu den Steuerungsanweisungen auf Basis von Zustandsgraphen über einen weiteren Algorithmus, der hier als **Fehlerbaum** bezeichnet wird (Abbildung 5-20). Dieser Algorithmus trägt diesen Namen, da er ein sequentieller Ablauf ist, der mittels eines ähnlich strukturierten Graphen dargestellt werden kann, wie der Graph aus der Fehlerbaumanalyse der Sicherheitstechnik *(DIN 25424 Teil 1 1981, DIN 25424 Teil 2 1990)*. Im Unterschied zum Graphen aus der Fehlerbaumanalyse werden hier jedoch nicht nur Symptom-Fehler-Beziehungen aufgezeigt, sondern es wird ähnlich dem Fehlersuchlaufplan nach *Pfeifer & Richter (1993, S. 93)* ein komplexer Ablauf beschrieben, der zur Störungsbehandlung im störungstoleranten Steuerungsbaustein herangezogen wird.

Der Fehlerbaum verfügt über zahlreiche Mechanismen zur Erfassung und Manipulation von Steuerungssignalen und Steuerungsvariablen. Zusätzlich verfügt er über unterschiedliche Mechanismen, mit denen standardisierte Bedienerdialoge geführt werden können.

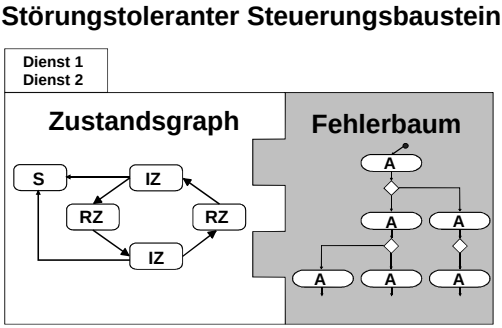


Abbildung 5-20: Der Fehlerbaum als Bestandteil des störungstoleranten Steuerungsbausteins

Der Fehlerbaum ist für die Phasen der Störungslokalisierung und den ersten Teil der Störungsbehebung, d. h. der Störungsbeseitigung während einer Reaktionsstrategie, zuständig. Er ist in die folgenden fünf Abschnitte gegliedert (Abbildung 5-21):

- **Quelle:** Wie bereits erläutert, beschreibt der Fehlerbaum einen sequentiellen Ablauf. Die Einstiegspunkte dieses Ablaufs werden als Quellen bezeichnet. Sie sind die Wurzeln der baumartigen Struktur des Fehlerbaums und repräsentieren jeweils eine ganz bestimmte Störung, wie z. B. Zeitüberschreitung in einem bestimmten Running-Zustand. Da im Zustandsgraphen mehrere Störungen erfaßt werden können, besitzt der Fehlerbaum auch mehrere Quellen.
- **Bereich der Ursachenfindung:** Von den Quellen gehen zahlreiche Äste und Verzweigungen aus, die Aktivitäten beinhalten, um die Ursache der entsprechenden Störung zu erfassen. Dabei erfolgt die Erfassung auf Basis der drei zuvor beschriebenen Lokalisierungsverfahren. Um Zustandsinformationen über das gesteuerte System vom Bediener zu erfassen, können die Aktionen auch Dialogfenster auf einem Bedienfeld aktivieren, die den Bediener mittels gezielter Fragen in den Lokalisierungsvorgang integrieren.
- **Bereich zur Ursachenbekanntgabe:** Nach erfolgreicher Ursachenfindung muß das erzielte Ergebnis aufbereitet und den entsprechenden Interessenten mitgeteilt werden. Hierzu zählt in erster Linie der Bediener, der die Informationen über Fenster auf dem Bedienfeld bekommt. Als Darstellungsmedium dieser Informationen kann hier auf verschiedene Formate zurückgegriffen werden. So können es einfache Textanweisungen oder einfache Bilder bzw. Graphiken sein,

die den Bediener über die aktuelle Störungsursache informieren. Vorstellbar sind aber auch Hypertexte oder Videosequenzen, die den Bediener mit noch weiteren Zusatzdaten informieren. Die Aktionen zur Aktivierung dieser verschiedenen Fenster sind in diesem Abschnitt des Fehlerbaumes eingetragen.

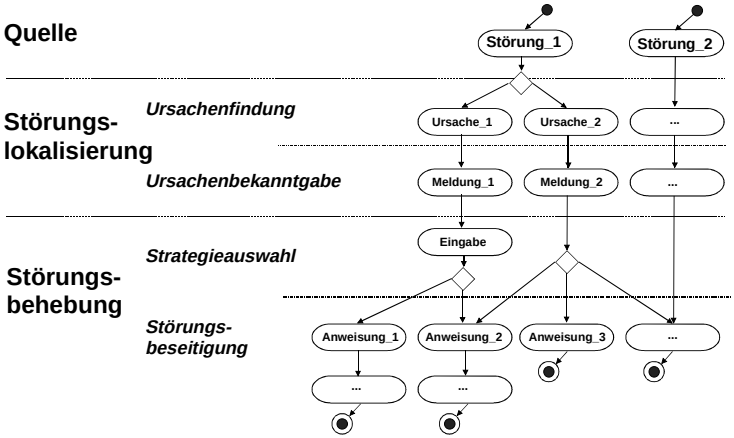


Abbildung 5-21: Abschnitte des Fehlerbaums

- **Bereich der Auswahl einer Reaktionsstrategie:** Die Auswahl der richtigen Störungsbehebungsstrategie, die im Anschluß an die Ursachenfindung durchgeführt wird, ist eine zentrale Aufgabe des Fehlerbaums. Je nach Störungsart und Anzahl der möglichen Strategien kann diese Auswahl automatisch oder in Abstimmung mit dem Bediener erfolgen.
- **Bereich zur Störungsbeseitigung:** Das ist der letzte Abschnitt in einem Fehlerbaumzweig. Hier werden alle Aktivitäten eingetragen, die der Störungsbeseitigung dienen. Da diese in erster Linie in Zusammenarbeit mit dem Bediener erfolgen, enthält dieser Abschnitt zahlreiche Aktionen, die Dialoge mit dem Bediener führen. Zur Darstellung dieser Anweisungen kann auf die gleichen Medien zurückgegriffen werden wie bei der Ursachenbekanntgabe.

Wie bereits beschrieben, führt der Fehlerbaum im Rahmen einer Reaktionsstrategie die Störungsbeseitigung aus. Die ist aber nur der erste Teil einer Reaktionsstrategie. Der noch anstehende Recovery-Vorgang ist ein Teil der zustandsorientierten Steuerungsanweisung. Wie die komplette Störungsbehandlung, d. h. von der Störungserkennung bis hin zum Wiederaufsetzen des normalen Betriebs, in einem Steuerungsbaustein des Referenzkonzepts erfolgt, wird im folgenden beschrieben.

5.4.2 Bausteininterner Störungsbehandlungsablauf

Im Steuerungsbaustein teilen sich die zustandsorientierte Steuerungsanweisung (Zustandsgraph) und der Fehlerbaum den gesamten Störungsbehandlungsablauf. Während der Zustandsgraph für alle zeitkritischen, reaktiven Vorgänge zuständig ist, führt der Fehlerbaum die zeitunkritischen, bedienerintegrierten Abläufe aus. Die einzelnen Schritte der Störungsbehandlung zeigt Abbildung 5-22.

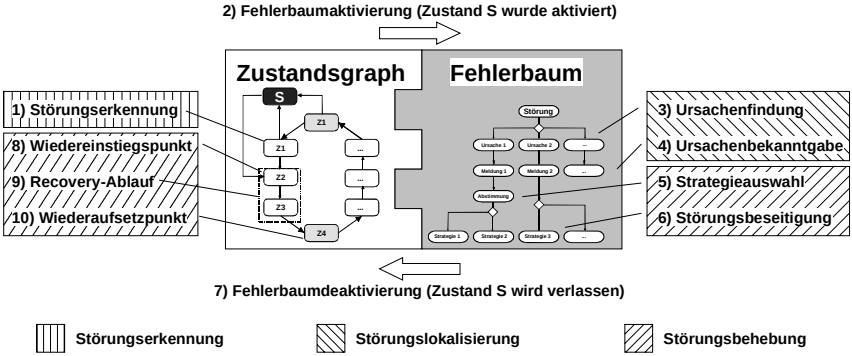


Abbildung 5-22: Gesamtablauf der Störungsbehandlung in einem Steuerungsbaustein

Mit der Erfüllung der Weiterschaltbedingung einer Erkennungstransition im Zustandsgraphen eines Steuerungsbausteines wird zunächst eine Störung erkannt und erfaßt (Abbildung 5-22/1). Das Feuern einer solchen Transition führt zur Aktivierung des Störungszustands. Mit dieser Aktivierung wird gleichzeitig der Abarbeitungsalgorithmus des Fehlerbaums gestartet und die Steuerungskontrolle an den Fehlerbaum übergeben (Abbildung 5-22/2). Während der gesamten Fehlerbaumabarbeitungsphase bleibt der Störungszustand aktiv. Erst nach Ende der Fehlerbaumabarbeitung wird die Steuerungskontrolle an den Zustandsgraphen zurückgegeben und der Störungszustand verlassen.

In Abhängigkeit der aufgetretenen Störung wird im Fehlerbaum die zugehörige Störungsquelle angesprungen und von dort aus die Abarbeitung gestartet. Im Fehlerbaum werden die vier oben beschriebenen Abschnitte durchlaufen. Es erfolgt die Lokalisierung der Störung (Abbildung 5-22/3), eine Bekanntgabe der Störungsursache (Abbildung 5-22/4), die Auswahl der richtigen Reaktionsstrategie (Abbildung 5-22/5) und zuletzt die Beseitigung aller Störungssymptome und Ursachen, die zum Anfahren des zugehörigen Wiedereinstiegspunktes benötigt werden (Abbildung 5-22/6). Dabei wird der Bediener bei all diesen Aufgaben mit Hilfe

von Dialogen in die Abläufe integriert. Am Ende der Fehlerbaumabarbeitung wird die Kontrolle mit einer Kennung, die die ausgewählte Reaktionsstrategie spezifiziert, wieder an den Zustandsgraphen übergeben (Abbildung 5-22/7).

Basierend auf dieser Kennung wird im Zustandsgraphen durch eine Transition der Störungszustand verlassen und derjenige Zustand aktiviert, der als Wiedereinstiegspunkt definiert wurde (Abbildung 5-22/8). Ist in der Reaktionsstrategie auch ein Wiederaufsetzpunkt spezifiziert, wird zudem der im Zustandsgraphen abgelegte Ablauf zwischen Wiedereinstiegs- und Wiederaufsetzpunkt abgearbeitet (Abbildung 5-22/9). Erst mit dem Erreichen des Wiederaufsetzpunktes ist die komplette Störungsbehandlung beendet (Abbildung 5-22/10).

Konnte durch diesen Störungsbehandlungsablauf die Störung in einem Dienst völlig behoben werden, erfolgt die Fertigstellung des ursprünglich angeforderten Dienstes. Konnte die Störung jedoch nicht behoben werden, muß dies dem Auftraggeber mitgeteilt und eine bausteinübergreifende Störungsbehandlung eingeleitet werden.

5.4.3 Bausteinübergreifender Störungsbehandlungsablauf

Bei der Durchführung eines Dienstes versucht der störungstolerante Baustein, die ihm gestellte Aufgabe vollständig abzuarbeiten. Mit Hilfe eines Rückmeldesignals wird dem Auftraggeber die erfolgreiche Beendigung des Dienstes mitgeteilt. Kommt es jedoch während der Durchführung zu einer Störung, versucht der Baustein die Störung lokal zu behandeln, ohne seinen Auftraggeber zu benachrichtigen. Auch hier bekommt der Auftraggeber nach Abschluß des Dienstes ein Rückmeldesignal für die erfolgreiche Beendigung des Dienstes.

Erst wenn ein Steuerungsbaustein die Störung nicht behandeln konnte, wird der Auftraggeber über den Mißerfolg des Dienstes informiert und in die Störungsbehandlung mit einbezogen (Abbildung 5-23). Nun versucht der Auftraggeber einen entsprechenden Störungsbehandlungsablauf einzuleiten. Hierfür benötigt er jedoch nicht nur eine Meldung über den Mißerfolg des geforderten Dienstes, sondern auch weitere Angaben über den aktuellen Zustand des gestörten Steuerungsbausteins und den Zustand des vom Baustein gesteuerten Systems. In Abhängigkeit von diesen Zustandsangaben kann die Störungsbehandlung beim Auftraggeber unterschiedlich ausfallen. So liegt es nahe, daß ein gestörter Steuerungsbaustein, der sich nach fehlgeschlagener Dienstausführung wieder im Ausgangszustand befindet, durchaus vom selben Auftraggeber mit einer Wiederholung des Dienstes beauftragt werden kann. Als Beispiel sei hier eine einfache Verklemmung in der Aktorik einer Funktionseinheit genannt, die nach dem ersten Versuch in den Ausgangszustand zurückfährt und mit einfacher Wiederholung behandelt werden kann. Wurde jedoch mit einer fehlgeschlagenen Dienstausführung die Steuerungsabarbeitung im Baustein angehalten, während sich die Funktionseinheit in einem

nicht definierten Zustand befindet, muß der Auftraggeber anders reagieren und im Zweifelsfall auch selbst seine eigene Abarbeitung anhalten.

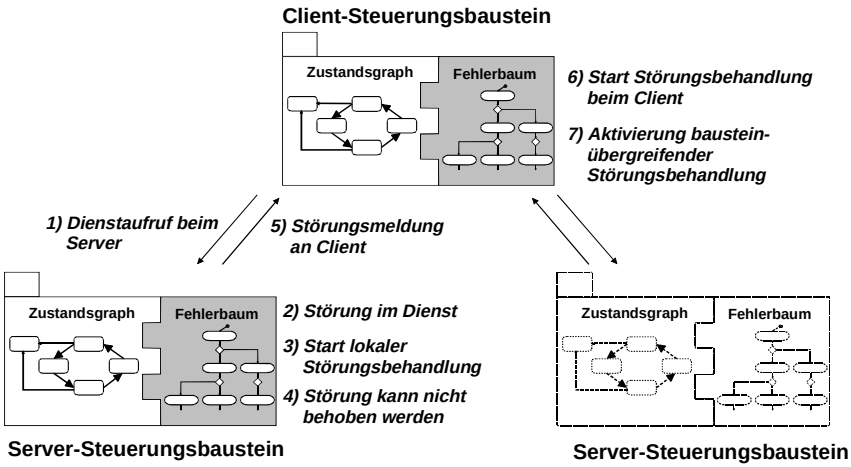


Abbildung 5-23: Bausteinübergreifende Störungsbehandlung

5.5 Zusammenfassung

In diesem Kapitel wurde ein Referenzkonzept für eine störungstolerante Steuerung vorgestellt. Dieses Referenzkonzept soll als funktionale und strukturelle Gestaltungsrichtlinie für derartige Steuerungen dienen.

Das Konzept ist modular aufgebaut und besteht aus sogenannten störungstoleranten Steuerungsbausteinen. Jeder Baustein stellt nach außen hin sogenannte Dienste zur Verfügung. Die Beschreibung der Steuerungsanweisungen für den störungsfreien Fall basieren auf den diskreten Zuständen des zu steuernden Systems. Dafür verfügt jeder Steuerungsbaustein über einen entsprechenden Zustandsgraphen. Für die Störungsbehandlung enthält der Zustandsgraphen weiterhin Anweisungen für die Störungserkennung sowie für den Recovery-Vorgang. Zusätzlich zum Zustandsgraph enthält jeder Steuerungsbaustein einen sequentiellen Ablauf zur automatischen und bedienerunterstützten Lokalisierung und Beseitigung von Störungen. Alle Abläufe,

die in einem Steuerungsbaustein erfolgen, können anhand einer graphischen Beschreibungstechnik spezifiziert werden; dabei wurden hier die UML-Diagramme eingesetzt.

6 Methodisches Vorgehen zur Entwicklung störungstoleranter Steuerungen

6.1 Übersicht

In diesem Kapitel wird ein methodisches Vorgehen zur Entwicklung störungstoleranter Steuerungsanwendungen, die nach dem Referenzkonzept zu entwerfen sind, beschrieben. Dabei steht hier die Entwicklung der Steuerungssoftware im Vordergrund.

Unter Berücksichtigung der Anforderungen aus Kapitel 3.3 beinhaltet das methodische Vorgehen eine graphische Beschreibungstechnik, die für alle Phasen der Softwareentwicklung eingesetzt werden kann. Des weiteren beinhaltet das methodische Vorgehen eine spezifische Vorgehensweise zur schrittweisen Entwicklung von störungstoleranten Steuerungsanwendungen unter Einsatz der entsprechenden Beschreibungstechnik (Abbildung 6-1).

In diesem Kapitel wird zunächst die hier spezifizierte Beschreibungstechnik erläutert. Anschließend erfolgt die Beschreibung der Vorgehensweise zur Entwicklung störungstoleranter Steuerungen.

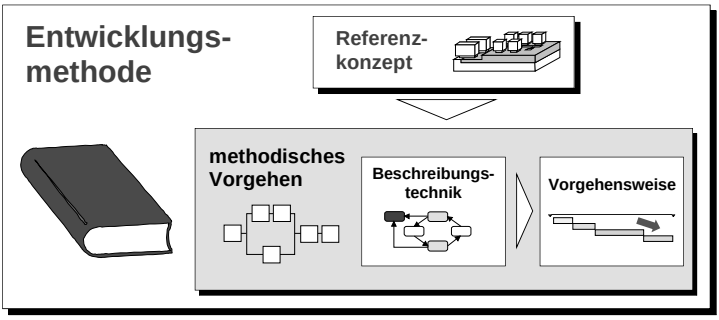


Abbildung 6-1: Elemente des methodischen Vorgehens

6.2 Beschreibungstechnik

6.2.1 Übersicht

Im vorherigen Kapitel wurde ein Referenzkonzept für störungstolerante Steuerung vorgestellt. Zur grundsätzlichen Veranschaulichung von Funktionalität und Aufbau dieses Konzepts wurden Zustandsgraphen und Aktivitätsdiagramme der UML verwendet. Sollen jedoch detaillierte Angaben zur konkreten Steuerungsanwendung einfach erfaßt und dokumentiert werden, sind zum einen einige Erweiterungen in diesen Diagrammen und zum anderen weitere Diagramme erforderlich. In diesem Kapitel wird eine Beschreibungstechnik spezifiziert, die auf Basis der UML die notwendigen Erweiterungen enthält. Diese Beschreibungstechnik dient als Werkzeug für alle Phasen einer methodischen Entwicklung störungstoleranter Steuerungen. Mit ihr

- werden alle Anforderungen an die zu entwickelnde störungstolerante Steuerungsanwendung, insbesondere hinsichtlich der Störungstoleranz, erfaßt.
- wird ein detailliertes Modell der entsprechenden Anwendung, das nach dem Referenzkonzept aufgebaut ist, beschrieben.

Die Elemente und Diagramme dieser Beschreibungstechnik sind in folgende zwei Kategorien gegliedert:

- **Störungsbehandlungstabellen:** Dies sind spezielle Regeltabellen, die rein der Erfassung von Anforderungen dienen. Hierzu zählen die ***Ursachenfindungs***- und ***Störungsbehebungstabellen***.
- **Modellelemente und -diagramme:** Diese Elemente und Diagramme, werden zur Beschreibung des Softwaremodells und anschließend zur Implementierung verwendet. Hierzu gehören die ***„störungstolerante Steuerungsklasse“***, der ***„STB-Zustandsgraph“*** und das ***„STB-Aktivitätsdiagram“***.

Diese beiden Kategorien werden im folgenden einzeln beschrieben:

6.2.2 Störungsbehandlungstabellen

Ergänzend zur UML werden im folgenden zwei Arten von Tabellen vorgestellt, die der systematischen Analyse und Erfassung von Anforderungen an die Störungsbehandlungsfunktionalität einer störungstoleranten Steuerung dienen. Mit diesen Tabellen kann der Entwickler einer Steuerung festhalten, welche Methoden der Störungserkennung, Störungslokalisierung und Störungsbehebung in den einzelnen Bausteinen der Steuerung zu verwenden sind.

6.2.2.1 Ursachenfindungstabellen

Die Ursachenfindungstabellen ähneln in ihrem Aufbau den aus der Softwaretechnik bekannten Entscheidungstabellen *(Frick 1995, S. 141)* (Abbildung 6-2/a). Sie dienen der Festlegung des geforderten Störungslokalisierungsverhaltens der Steuerung. Anhand dieser Tabellen spezifiziert der Entwickler, welche Methoden der Ursachenfindung die Steuerung für jeweils eine konkret von der Steuerung erfaßbare Störung einsetzen soll.

Abbildung 6-2/b zeigt den schematischen Aufbau einer Ursachenfindungstabelle. Hier sind, wie bei den Regeln einer Entscheidungstabelle, alle möglichen elementaren Störungsursachen aufgelistet, die zu der von der Steuerung erfaßbaren Störung führen können. Als Bedingungen der Störungsursachen werden alle Ereignisse und Einzelsymptome aufgezählt, die unmittelbar mit der Störung zusammenhängen und die für die von der Steuerung durchzuführende Ursachenfindung wesentlich sind. Dabei wird zwischen automatisch erfaßbaren und nur manuell erfaßbaren Symptomen unterschieden. Während erstere direkt von der Steuerung mittels entsprechender Sensoren erkannt werden können, sind letztere nur vom Bediener erfaßbar, da hierfür weder Sensoren vorgesehen sind noch eingesetzt werden können oder auf ein besonderes Erfahrungswissen zurückgegriffen werden muß.

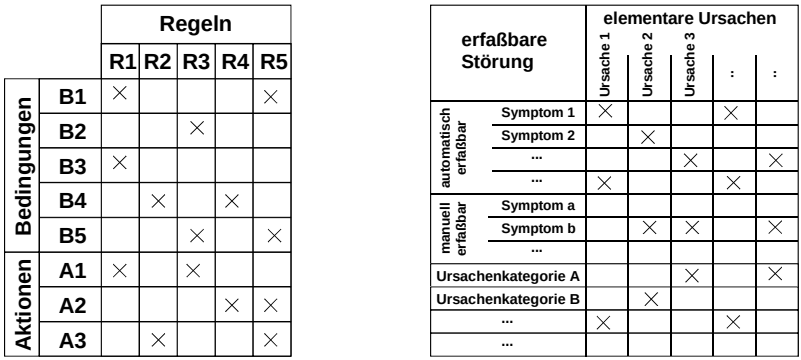


Abbildung 6-2: Entscheidungstabellen und Ursachenfindungstabellen

Die Aktionseinträge der Ursachenfindungstabellen sind Störungsmeldungen, die am Bedienpult erscheinen. Gleichzeitig werden mit diesen Aktionen sogenannte **Ursachenkategorien** definiert, die eine Zusammenfassung mehrerer Ursachen sind.

Kann beispielsweise die Ursache für die Zeitüberschreitung beim Ausfahren eines Zylinders ein defekter Sensor oder eine lose Klemme für das Eingangssignal sein, können diese Elementarursachen der Ursachenkategorie „Defekt im Geberstrang“ zugeordnet werden. Mit den Ursachenkategorien werden elementare Störungsursachen zu Gruppen zusammengefaßt, deren Detaillierungsgrad ausreichend ist, um eine adäquate Störungsbehebung auszuwählen und einzuleiten.

Die Spezifikation der Störungsbehebungsfunktionalität einer Steuerung erfolgt in einer eigenen Tabelle, die hier als **Behebungstabelle** bezeichnet wird.

6.2.2.2 Behebungstabellen

Auch diese Tabellen sind eine besondere Form von Entscheidungstabellen. Sie dienen der Spezifikation des geforderten Störungsbehebungsverhaltens der Steuerung. Mit einer Behebungstabelle (Abbildung 6-3) wird angegeben, welche verschiedenen Behebungsverfahren für eine Ursachenkategorie in Frage kommen. Jede Regelspalte ist eine Alternative. Die Bedingungen der Tabelle stellen die unterschiedlichen Voraussetzungen dar, die zum Einleiten der entsprechenden Behebung erfüllt sein müssen. Die Bedingungen spezifizieren somit den Auswahlvorgang der einzuleitenden Störungsbehebung.

Ursachen-kategorie A		**R1**	**R2**	**R3**	**R4**	**R5**
Strategie-auswahl	**Bedingung 1**	X			X	
	Bedingung 2		X			
	...			X		X
Reaktionsstrategie **Beseitigung**	**Aktivität 1**	X			X	
	Aktivität 2					X
	...			X		
	...				X	
Reaktionsstrategie **Recovery**	**WEP**	**Z1**	**Z4**	**H**	**Z2**	**H**
	WAP	**Z3**	**H**			
	Meldung	**M1**		**M4**		

Abbildung 6-3: Störungsbehebungstabellen

Die Summe der Aktionen, die für eine Regelspalte angegeben sind, spezifiziert jeweils eine Reaktionsstrategie (siehe auch Kapitel 5.3.5). Diese Aktionen sind einerseits Aktivitäten, die zur Beseitigung von Störungssymptomen und -ursachen beitragen. Sie können manuelle Aktivitäten sein, die Bedieneranweisungen

beinhalten, oder automatische Aktivitäten darstellen, die die Steuerung selbständig durchführt. Die Aktionen der Tabelle sind andererseits Angaben zum Recovery-Vorgang. Dazu gehören Angaben zum Wiedereinstiegs- und Wiederanlaufpunkt sowie Angaben über eventuelle Meldungen, die an andere Steuerungsbausteine im Rahmen einer bausteinübergreifenden Störungsbehandlung geschickt werden sollen.

6.2.3 Modellelemente und -diagramme

Wie in der UML sind die Modellelemente und -diagramme der hier spezifizierten Beschreibungstechnik graphische Notationen mit einem zugehörigen Metamodell. Dieses Metamodell basiert auf dem in Kapitel 5 beschriebenen Referenzkonzept einer störungstoleranten Steuerung. Mit diesen Elementen und Diagrammen kann der Entwickler ein detailliertes Modell der zu entwickelnden Anwendung erstellen und dies anschließend zur Implementierung heranziehen.

6.2.3.1 Die störungstolerante Steuerungsklasse

Ähnlich der objektorientierten Software basiert der in Kapitel 5 beschriebene störungstolerante Steuerungsbaustein nach dem Metamodell auf einer Klassendefinition, die alle seine Eigenschaften beschreibt und hier als störungstolerante Steuerungsklasse bezeichnet wird. Der störungstolerante Baustein selbst ist ein Objekt, das basierend auf der Klassendefinition entsteht. Er ist somit eine Instanz einer Steuerungsklasse.

Die störungstolerante Steuerungsklasse ist nach außen hin eine Art Black box, die ihr eigenes, von anderen Klassen unabhängiges Verhalten führt. Sie ist eine aktive Klasse, d. h. ihre Instanzen stellen zur Laufzeit nebenläufige Prozesse[4] dar. In bezug auf die UML ist die störungstolerante Steuerungsklasse die Erweiterung einer UML-Klasse. Derartige Erweiterungen oder Ergänzungen werden in der UML als Stereotyp bezeichnet *(Oesterreich 1997, S. 177).* Graphisch werden Stereotypen und ihre Instanzen durch ein Klassen- bzw. Objektsymbol mit dem Vermerk <<Stereotyp_name>> im Klassen- bzw. Objektdiagramm dargestellt. Für die störungstolerante Steuerungsklasse wird der Vermerk <<STB>> (Störungs-behandelnd) gewählt (Abbildung 6-4).

Jede Steuerungsklasse stellt zahlreiche Dienste zur Verfügung, die von anderen Klassen aufgerufen werden können. Die Interaktion zwischen den verschiedenen Klassen erfolgt mit Hilfe von Nachrichten, die binäre Signale oder einfache Datentypen sind. Dazu definiert die Klasse zahlreiche Attribute in Form von

[4] Zwei Prozesse oder Vorgänge heißen nebenläufig, wenn sie voneinander unabhängig bearbeitet werden können.

Eingangs- oder Ausgangsvariablen. Zusätzlich zur Interaktion zwischen den Klassen dienen diese Attribute der Anbindung an die binären Ein- und Ausgänge der Steuerung sowie einer nachrichtenbasierten Anbindung an die Benutzeroberfläche des Bedienerpults der Steuerung.

Zur graphischen Darstellung der Dienste, die eine Klasse bzw. Instanz anbietet, kann hier das Schnittstellensymbol der UML (Lolly-Symbol) verwendet werden. Mit dem Schnittstellensymbol wird gleichzeitig dargestellt, welche Eingangs- und Ausgangsvariablen mit dem Aufruf bzw. zur Bestätigung des entsprechenden Dienstes verbunden sind (Abbildung 6-4). In zahlreichen Fällen greift eine Klasse bzw. die Instanz selbst auf die Dienste externer Klassen zurück. Auch dies erfolgt mittels Eingangs- und Ausgangsvariablen, die in der Klasse definiert sind. Um diese externen Dienste im Klassendiagramm aufzuzeigen, werden hier in Anlehnung an die UML gestrichelte Pfeile verwendet, an die die jeweiligen Eingangs- und Ausgangsvariablen eingetragen werden. Ist der Name des externen Dienstes bekannt, kann auch dieser mit angegeben werden (Abbildung 6-4).

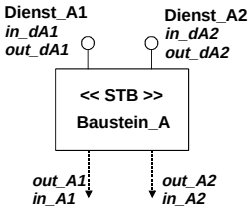


Abbildung 6-4: Graphische Darstellung der störungstoleranten Steuerungsklasse

Um den instantiierten Aufbau einer kompletten störungstoleranten Anwendung zu zeigen, werden hierzu alle störungstoleranten Bausteine in einem Objektdiagramm dargestellt. Zusätzlich zu den Instanzen sind hier noch die Verbindungen zwischen den verschiedenen Bausteinen eingetragen. Bei den Verbindungen handelt es sich immer um eine „Verwendet“-Beziehung zwischen den Klassen. Hier wird der Name des Dienstes so, wie ihn der Serverbaustein definiert, angegeben. Zudem werden die zugehörigen Eingangs- und Ausgangsvariablen sowohl auf Server- als auch auf Client-Seite eingetragen (Abbildung 6-5). Nicht aufgezeigt werden im Diagramm die Anbindungen der Bausteine an die Hardware der Steuerung, d. h. an die physikalischen Eingänge und Ausgänge sowie die Benutzeroberfläche am Bedienerpult.

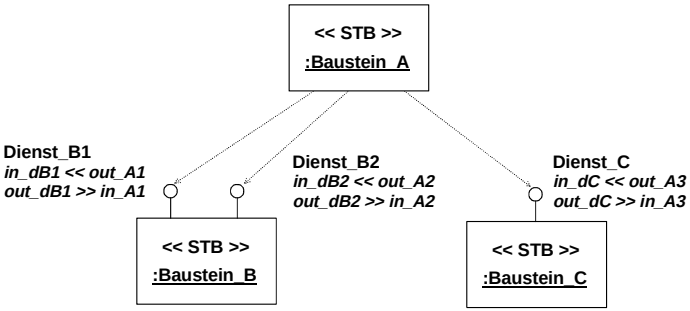


Abbildung 6-5: Graphische Darstellung der Bausteine einer störungstoleranten Anwendung

6.2.3.2 Der STB-Zustandsgraph

Zur Spezifikation des dynamischen Verhaltens einer störungstoleranten Steuerungsklasse verfügt diese nach dem Referenzkonzept über einen eigenen Zustandsgraphen. Dieser beschreibt die störungstolerante Steuerungsfunktionalität der Klasse und wird hier als **STB-Zustandsgraphen** (Störungsbehandelnd) bezeichnet, da er im Vergleich zu UML-Zustandsgraphen über zusätzliche Elemente verfügt.

Grundelemente des Graphen

Grundsätzlich spezifizieren die Zustände des STB-Zustandsgraphen die diskreten Zustände des mit dem Steuerungsbaustein zu steuernden technischen Systems. Diesen Zuständen können Entry- oder Exit-Aktionen zugeordnet werden (siehe auch Kapitel 5.2.2). Im Rahmen der Störungsbehandlungsfunktionalität werden zusätzlich sogenannte bedingte Entry- und Exit-Aktionen benötigt, die nur bei der Erfüllung einer Bedingung ausgeführt werden. Zur Darstellung dieser Sonderaktionen werden die UML-Zustandsgraphen durch zwei Symbole ergänzt (Abbildung 6-6).

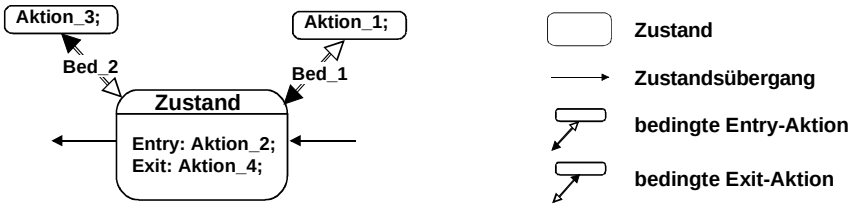


Abbildung 6-6: Bedingte Entry- und Exit-Aktionen

Wie bereits beschrieben, werden im Referenzkonzept Idle- und Running-Zustände unterschieden. Bei Idle-Zuständen befindet sich das modellierte System in einer Ruhelage, Running-Zustände müssen zum Erreichen der Ruhezustände durchlaufen werden. Zur graphischen Unterscheidung der beiden Zustände werden hier die Idle-Zustände grau dargestellt (Abbildung 6-7).

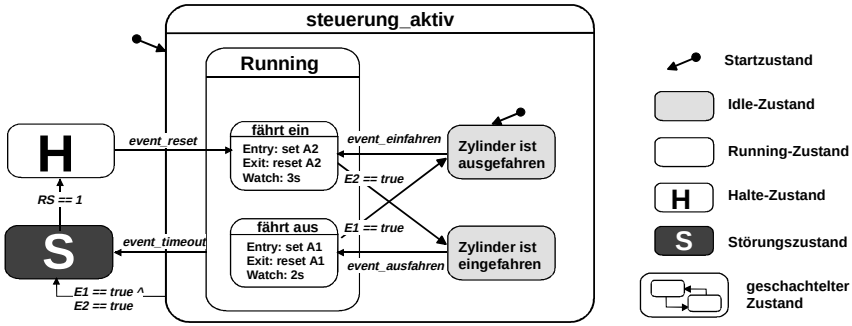


Abbildung 6-7: Grundelemente des STB-Zustandsgraphen

Zusätzlich zu den obengenannten Zuständen besitzt jeder STB-Zustandsgraph einen sogenannten **Halte-Zustand** (siehe auch Kapitel 5.3.5). Dies ist keine Abbildung eines realen Zustands des zu steuernden Systems, sondern ein rein steuerungstechnischer Zustand, in dem keine Stellsignale an das zu steuernde System geschickt werden. Der Steuerungsprozeß wird dadurch sozusagen angehalten. Dieser Zustand kann nur über eine Initialisierung verlassen werden. Graphisch wird er durch ein Zustandssymbol, das ein „H“ enthält, dargestellt (Abbildung 6-7).

Ein weiterer Sonderzustand ist der **Störungszustand** (siehe auch Kapitel 5.3.2), der in jedem STB-Zustandsgraphen genau einmal enthalten ist. Dieser Zustand stellt diejenigen Situationen dar, bei denen eine Störung im zu steuernden System aufgetreten ist. Alle Störungserkennungstransitionen des Graphen münden in diesen Zustand. Graphisch wird er durch ein schwarzgefärbtes Zustandssymbol, das den Buchstaben „S“ enthält, dargestellt (Abbildung 6-7).

Hierarchisierung von Zuständen

Weil Zustandsgraphen in komplexen Anwendungen sehr umfangreich und unübersichtlich sein können, erweiterte *Harel (1987)* die bis dato bekannten Zustandsgraphen mit seinen State Charts um das Konzept der hierarchischen Zustandsgraphen *(Balzert 1996, S. 277)*. Nach diesem Konzept können Zustände „geschachtelt“ werden, so daß ein Oberzustand mehrere Unterzustände umfaßt. Die Transitionen in diesen Graphen können ihren Ursprung und ihr Endziel auf jeder

beliebigen Hierarchieebene haben. Die Verschachtelung von Zuständen beschreibt zunächst semantisch eine exklusive Oder-Beziehung. Dies bedeutet, daß immer nur ein Unterzustand aktiv sein kann, wenn sich ein System in einem Oberzustand befindet. Der Oberzustand ist somit eine Abstraktion der Unterzustände. Im übrigen ist eine Transition, die einen Oberzustand verläßt, gleichzeitig eine Ausgangstransition aller Unterzustände. Dadurch kann die Anzahl der Transitionen in einem Zustandsgraphen erheblich reduziert werden.

Zur vereinfachten Beschreibung des Softwaremodells einer störungstoleranten Anwendung wird das Konzept der hierarchischen Zustände in folgenden Zusammenhängen in STB-Zustandsgraphen eingesetzt:

- **Oberzustand zur Störungserkennung:** Zur gleichzeitigen Überwachung bzw. zur permanenten Überwachung mehrerer Zustände wird hier auf das Konzept der hierarchischen Zustände zurückgegriffen. Hier wird die Ausgangstransition eines Oberzustands gleichzeitig Ausgangstransition aller Unterzustände. Dadurch kann die Anzahl der eingetragenen Störungserkennungstransitionen erheblich reduziert werden, was wesentlich zur Erhöhung der Übersichtlichkeit beiträgt. In Abbildung 6-7 stellt der Zustand „Steuerung_aktiv“ einen derartigen Oberzustand dar.
- **Dienstzustand**: Um die Spezifikation komplexer Dienste einer Steuerungsklasse zu vereinfachen, wird ein sogenannter Dienstzustand eingeführt. Dieser ist eine abstrakte Zusammenfassung mehrerer Running-Zustände, die von einem Idle-Zustand in einen anderen Idle-Zustand führen. Bei der Modellierung einfacher Funktionseinheiten, bei denen ein Dienst nur einen einzigen Running-Zustand enthält, ist die Verwendung des Dienstzustands nicht unbedingt erforderlich. Zur Beschreibung von Diensten, die mehrere nachfolgende Running-Zustände enthalten, wie z. B. einen Koordinationsablauf bei komplexen Transportvorgängen, kann die Verwendung eines derartigen Zustands sehr nützlich sein (Abbildung 6-8). Der Übergang in den Dienstzustand kann durch eine Transition erfolgen, die nur zum Rand des Dienstzustandes führt. Für die untergeordneten Running-Zustände wird dann ein eigener Startzustand benötigt (Abbildung 6-8/ unten). Es können aber auch Transitionen eingetragen werden, die direkt vom Idle-Zustand in den Running-Zustand führen (Abbildung 6-8/ oben).

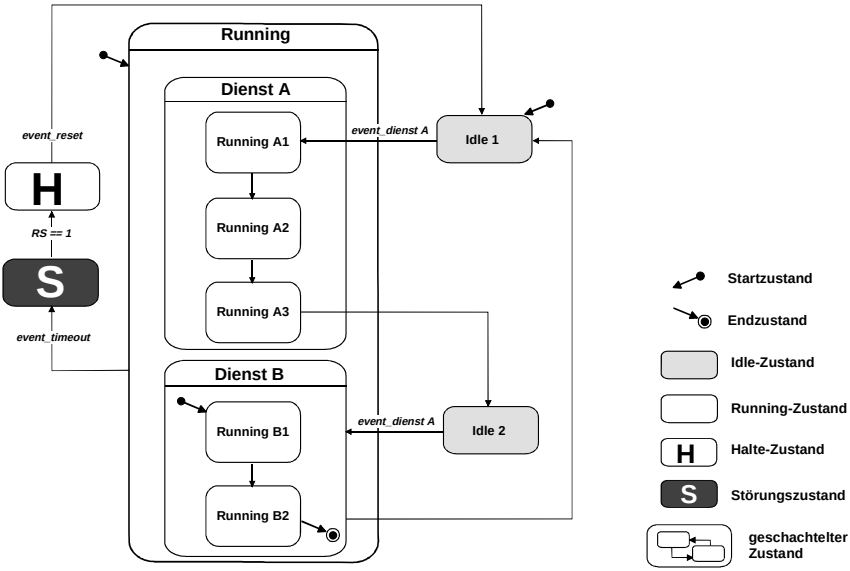


Abbildung 6-8: Der Dienstzustand als geschachtelter Zustand

Umkehrung von Abläufen

Eine weitere hier benötigte Ergänzung der UML-Zustandsgraphen zur vereinfachten Beschreibung des Softwaremodells einer störungstoleranten Steuerung sind die Elemente zur Spezifikation der Umkehrung von Abläufen. Wie bereits in Kapitel 5.3.5 beschrieben, werden in diesem Zusammenhang folgende zwei Arten von Umkehrungen unterschieden:

- Rückwärtsdienst und
- Rückwärtspfad.

Rückwärtsdienste sind, wie der Name schon sagt, eigene Dienste. Um sie zu beschreiben, können herkömmliche Dienstzustände eingesetzt werden. Anders verhält es sich jedoch bei den Rückwärtspfaden. Sie stellen keinen eigenen, kompletten Dienst dar und müssen deshalb als eine Kette von Running-Zuständen aufgebaut werden, die parallel zu den normalen Diensten in einem Zustandsgraphen abgelegt sind. Besonders bei großen komplexen Abläufen mit vielen Einzelschritten, wie dies beispielsweise bei Koordinationsabläufen in Kopfsteuerungen von Transferstraßen der Fall ist, können diese zusätzlichen Ketten zu unübersichtlichen Graphen führen. Um dies zu verhindern, werden in STB-Zustandsgraphen sogenannte **reversible Running-Zustände** und **bidirektionale Transitionen**

definiert. Diese Elemente sind keine neuen Elemente des UML-Metamodells, sondern nur eine graphische Erweiterung, mit der Running-Zustände und die zugehörigen Umkehrungen in einem graphischen Symbol zusammengefaßt werden. Dies gilt ebenfalls für die bidirektionalen Transitionen, die eine graphische Zusammenfassung zweier Transitionen darstellen (Abbildung 6-9).

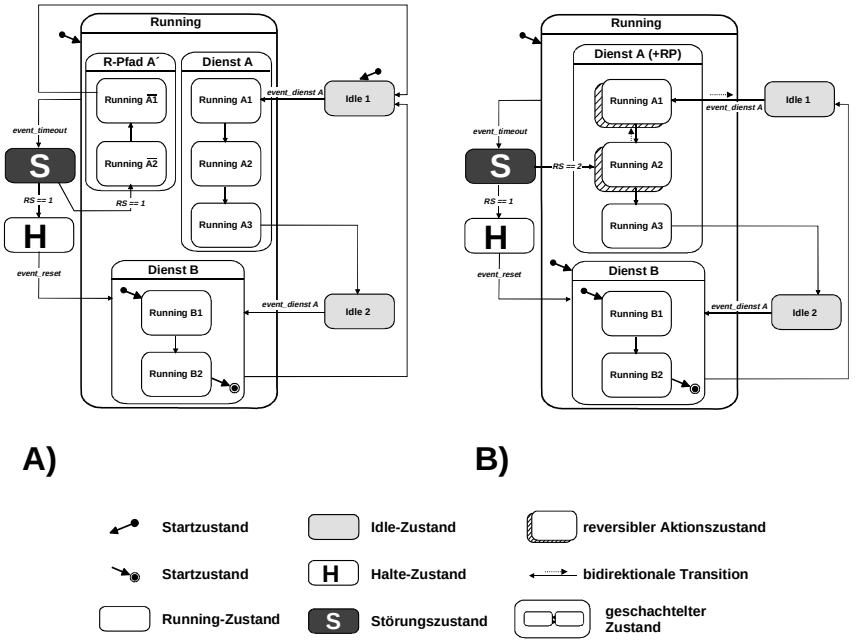


Abbildung 6-9: Rückwärtspfad ohne (A) und mit (B) reversiblen Running-Zuständen und bidirektionalen Transitionen

6.2.3.3 Das STB-Aktivitätsdiagramm

Die Spezifikation des Verhaltens einer störungstoleranten Steuerungsklasse während der Aktivierung des Störungszustands erfolgt mit Hilfe des sogenannten STB-Aktivitätsdiagramms. Dieses Aktivitätsdiagramm spezifiziert ergänzend zum UML-Aktivitätsdiagramm besondere Aktionszustände zur Störungsbehandlung, um den Fehlerbaum aus dem Referenzkonzept für störungstolerante Steuerungen zu beschreiben.

Im STB-Zustandsgraphen münden die verschiedenen Erkennungstransitionen in den Störungszustand. Für jede Transition gibt es dann eine eindeutige Quelle im STB-Aktivitätsdiagramm. Ähnlich ist es mit den Ausgangstransitionen des Störungszustands. So kann jedem Endzustand im STB-Aktivitätsdiagramm eine eindeutige Ausgangstransition vom Störungszustand zugeordnet werden (Abbildung 6-10).

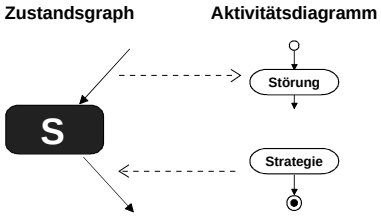


Abbildung 6-10: Kopplung von Zustandsgraph und Aktivitätsdiagramm

Die verschiedenen Aktivitäten zur Störungslokalisierung und -behebung werden im STB-Aktivitätsdiagramm durch Aktionszustände spezifiziert. Generell sind Aktionszustände dadurch gekennzeichnet, daß ihre Ausgangstransitionen im Vergleich zu normalen Zuständen sofort nach Beendigung der eingetragenen Entry-Aktionen feuern und hier keine Weiterschaltbedingungen in den Transitionen vorliegen *(Oesterreich 1997, S. 213)*.

Im STB-Aktivitätsdiagramm werden folgende drei Varianten von Aktionszuständen unterschieden.

- **Aktionszustände zur Manipulation von Systemvariablen**. Bei diesen Zuständen dienen die Entry-Aktionen ausschließlich der Manipulation von Variablen, die die störungstolerante Steuerungsklasse definieren. Diese Aktionen ähneln den herkömmlichen Aktionen eines Zustandsgraphen (Abbildung 6-11/a).
- **Aktionszustand zur Erfassung von Bedienereingaben**: Mit den Aktionen dieser Zustände werden vom Bediener verschiedene Informationen erfaßt, die besonders für die Störungsursachenfindung und für die Wahl der richtigen Reaktionsstrategie wesentlich sind. In diese Zustände werden Texteinträge, wie z. B. Fragen, die auf die vom Bediener gewünschten Eingaben hinweisen, abgelegt. Zudem können in den Zuständen auch Lösungsvorschläge für mögliche Eingaben festgehalten werden. Die Aktionszustände zur Erfassung von Bedienereingaben gelten erst dann als beendet, wenn der Bediener seine erwünschte Angabe korrekt angegeben hat. Abbildung 6-11/b zeigt, wie dieser Aktionszustand graphisch dargestellt wird.

Abbildung 6-11: Varianten von Aktionszuständen in der Beschreibungstechnik der störungstoleranten Steuerung

- **Aktionszustand zur Infoausgabe an den Bediener**: Diese Zustände enthalten Aktivitäten, die der Bereitstellung von Information für den Bediener dienen. Hiermit können beispielsweise ermittelte Störungsursachen dem Bediener mitgeteilt werden. In den Zuständen werden die anzugebenden Informationen als Texteinträge abgelegt. Die Aktionen dieser Zustände gelten erst dann als beendet, wenn der Bediener die an ihn gerichtete Information quittiert. Abbildung 6-11/c zeigt, wie dieser Aktionszustand graphisch dargestellt wird.

6.3 Vorgehensweise

6.3.1 Übersicht

Nachdem im vorherigen Abschnitt die Beschreibungstechnik zur Entwicklung störungstoleranter Anwendungen aufgezeigt wurde, wird hier die zugehörige Vorgehensweise erläutert, die beschreibt, wie unter Verwendung dieser Beschreibungstechnik Steuerungssoftware systematisch in einzelnen Phasen entwickelt werden kann. Dabei wird hier den Anforderungen der Entwicklung störungstoleranter Steuerungen aus Kapitel 3.3 Rechnung getragen. Dies gilt insbesondere für die Parallelisierung der Entwicklung von Steuerungs- und Störungsbehandlungsfunktionalität.

Die hier entwickelte Vorgehensweise unterscheidet grundsätzlich zwei Phasen: die Modellierungsphase und die Implementierungsphase (Abbildung 6-12). Die Modellierungsphase umfaßt die klassischen Phasen der Softwareanalyse und des Softwareentwurfes. Hier stellt das Referenzkonzept aus Kapitel 5 die Gestaltungsrichtlinie zur Modellierung und die Beschreibungstechnik aus Kapitel 6.2 die Sprache zur Spezifikation der Modellierung. Ergebnis der Modellierungsphase ist ein vollständiges Softwaremodell der zu entwickelnden störungstoleranten Steuerung. Auf Basis dieses Modells erfolgt in der Implementierungsphase die Realisierung der Software der Steuerung, wobei hier unterschiedliche Realisierungsansätze unterschieden werden.

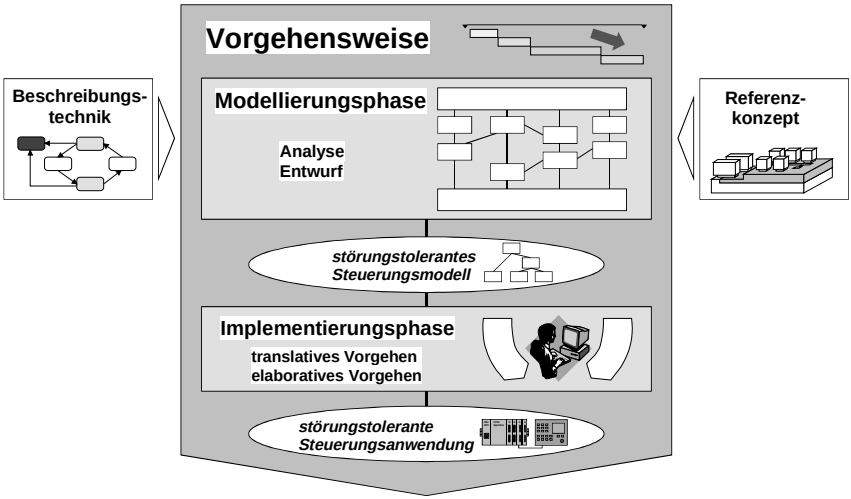


Abbildung 6-12: Vorgehensweise zur Entwicklung störungstoleranter Steuerungsanwendungen

Im folgenden sollen die zwei Phasen der hier entwickelten Vorgehensweise einzeln beschrieben werden.

6.3.2 Modellierung einer störungstoleranten Anwendung

Die Modellierung einer störungstoleranten Anwendung erfolgt in vier Schritten, wie dies in Abbildung 6-13 dargestellt ist. So müssen zunächst die für die Gesamtsteuerungsarchitektur benötigten störungstoleranten Steuerungsklassen identifiziert werden. Dies erfolgt in Abhängigkeit von den Funktionseinheiten und Baugruppen des zu steuernden Systems. Anschließend werden diese Klassen einzeln spezifiziert. Dabei wird zwischen einer Grob- und einer Feinspezifikation unterschieden. Zum Schluß werden die verschiedenen Klassen instantiiert und verknüpft, um die komplette Steuerungsarchitektur zu beschreiben.

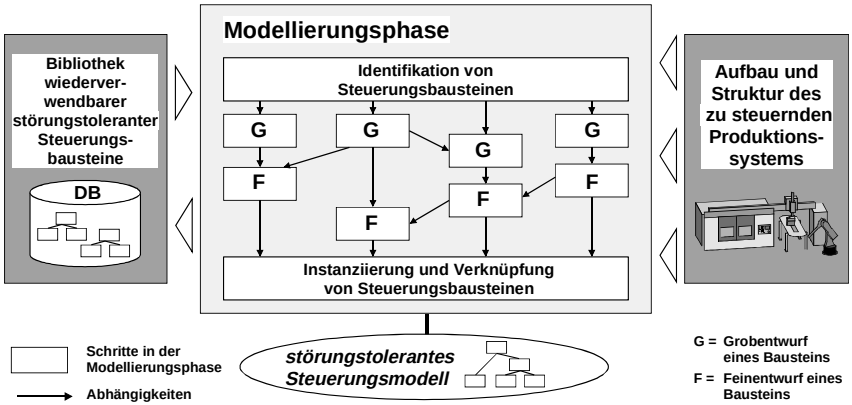


Abbildung 6-13: Schritte der Modellierung einer störungstoleranten Steuerungsanwendung

Während dieser einzelnen Schritte der Modellierungsphase können CASE-Tools eingesetzt werden, um die erfaßten Informationen und die erstellten Softwaremodelle in datentechnischer Form abzulegen. Dies erleichtert die Verwaltung sowie die Wiederverwendung der entsprechenden Daten.

Um das Vorgehen in der Modellierungsphase genauer zu spezifizieren, werden im folgenden die vier Schritte der Modellierungsphase einzeln erläutert. Wesentlich an diesen Schritten ist die Parallelisierung der Entwicklung von Steuerungs- und Störungsbehandlungsfunktionalität.

6.3.2.1 Identifikation der Steuerungsklassen

Grundlage der objektorientierten Softwareentwicklung ist die Definition von Klassen und Objekten, die in erster Linie die Struktur des zu entwickelnden Softwaresystems spezifizieren. Dabei ist besonders die Frage nach dem Prinzip, wie die Klassen bzw. Objekte der betrachteten Software identifiziert werden, von großem Interesse. Hierzu gibt es jedoch keine standardisierte Vorgehensweise *(Douglass 1998, S. 92ff.).* Vielmehr hängt das Vorgehen sehr stark von der zu entwickelnden Applikation ab. Während beispielsweise zur Klassenfindung objektorientierter Software von Informationssystemen hauptsächlich die Akteure und Daten aus Anwendungsfällen (USE-Cases) herangezogen werden *(Jacobsen 1992),* orientiert sich die Spezifikation von Klassen bzw. Objekten bei der Entwicklung objektorientierter Software von Automatisierungssoftware vorwiegend an der Struktur und dem Aufbau des zu steuernden Systems *(Awad u. a. 1996, Douglass 1998).* Da es sich bei einer störungstoleranten Steuerungsanwendung um ein Automatisierungssystem handelt,

stellt die mechanische Struktur des zu steuernden Systems die Grundlage für die Softwarearchitektur und somit die benötigten Steuerungsklassen dar.

Mit einer störungstoleranten Steuerungsklasse kann eine Funktionseinheit, eine Baugruppe, aber auch ein komplettes Aggregat modelliert werden. Um einerseits eine hohe Störungstoleranz zu erhalten und andererseits eine maximale Wiederverwendung bereits erstellter Software zu erreichen, empfiehlt es sich, das Produktionssystem in einzeln ansteuerbare Funktionseinheiten zu gliedern, die von separaten Steuerungsbausteinen gesteuert werden können. So sollte immer versucht werden, eine Steuerungsarchitektur mit einer hohen Modularität zu erhalten. Für die Störungstoleranz hat dies den Vorteil, daß durch eine detaillierte Gliederung der Steuerung in einzelne Bausteine ausführliche Zustandsangaben über das zu steuernde System abgebildet werden können. Diese Angaben sind zur Laufzeit für die Störungserkennung, -lokalisierung und -behebung von großer Bedeutung. Außerdem wird durch eine hohe Modularität die Wiederverwendbarkeit bereits erstellter Software wesentlich erhöht. Die Module können in einer Bibliothek abgelegt und für zukünftige Entwicklungen wiederverwendet werden.

Mit der Modellierung einzelner Funktionseinheiten eines Produktionssystems werden jedoch keine übergreifenden Abläufe in einem Produktionssystem beschrieben. Hierfür sind noch weitere Steuerungsbausteine erforderlich, die die Koordination der Funktionseinheiten für komplexe Abläufe vornehmen. Diese Koordinatoren greifen dann auf die Dienste der Funktionseinheiten zurück.

Durch die Verwendung von Koordinatoren entstehen hierarchische Steuerungsarchitekturen, bei denen auf unterster Ebene die einzelnen Funktionseinheiten modelliert werden. Auf den übergeordneten Ebenen liegen dann die Bausteine zur Koordination komplexer Abläufe. Abbildung 6-14 zeigt unterschiedliche Möglichkeiten zur Strukturierung einer störungstoleranten Steuerung. Vor der gewählten Struktur hängen letztendlich auch die benötigten Steuerungsklassen, die im ersten Schritt der Modellierungsphase zu identifizieren sind, ab. Welche Steuerungsstruktur dabei die geeignetste ist, hängt zwar immer von der konkreten Steuerungsapplikation ab. Grundsätzlich kann jedoch gesagt werden, daß mit zunehmender Anzahl der Steuerungsbausteine die Komplexität eines einzelnen Steuerungsbausteins abnimmt und im allgemeinen, wie bereits beschrieben, die Wiederverwendbarkeit der Steuerungssoftware steigt.

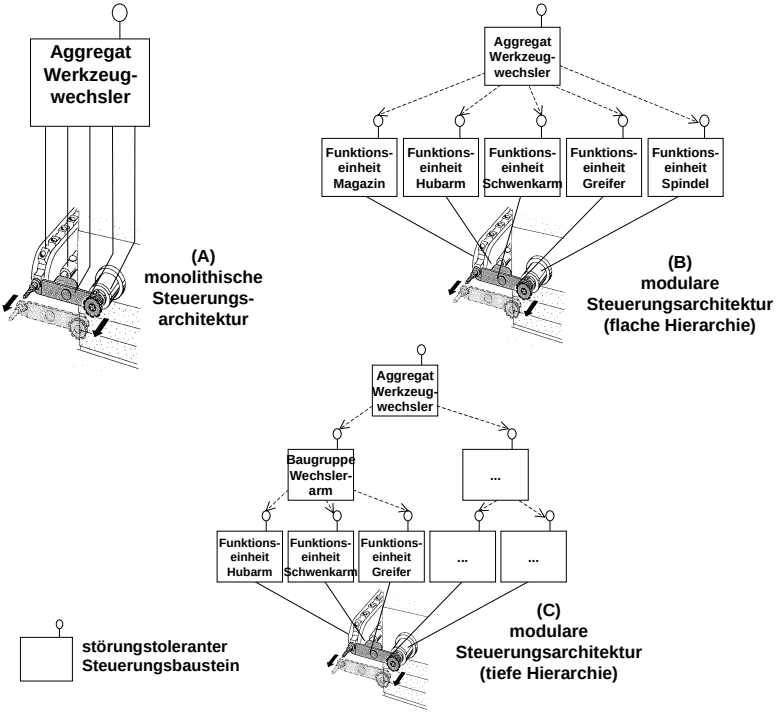


Abbildung 6-14: Varianten zur Strukturierung einer störungstoleranten Steuerung eines Werkzeugwechsel-Aggregates

6.3.2.2 Grobspezifikation

Der zweite Schritt in der Modellierungsphase der hier vorgestellten Vorgehensweise ist die Grobspezifikation der bereits identifizierten Steuerungsklassen. Mit der Grobspezifikation sind folgende Teilaufgaben verbunden:

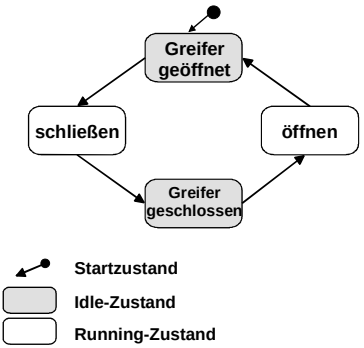
1. Spezifikation der grundlegenden Steuerungsfunktionalität,
2. Spezifikation der zu erfassenden Störungen,
3. Spezifikation der gewünschten Ursachenfindung und
4. Spezifikation der gewünschten Störungsbehebung.

Spezifikation der grundlegenden Steuerungsfunktionalität

Mit dem STB-Zustandsgraphen wird die grundlegende Steuerungsfunktionalität einer Steuerungsklasse beschrieben. Diese Beschreibung basiert auf der Modellierung der zu steuernden mechanischen Komponente anhand von Idle- und Running-Zuständen und der Definition der damit verbundenen Dienste. Dabei sind zwei Arten von Modellierungen zu unterscheiden: die **rein zustandsorientierte** Modellierung und die **dienstorientierte** Modellierung.

Bei einer rein zustandsorientierten Modellierung beschreiben die Zustände des erstellten Zustandsgraphen eindeutig alle diskreten Zustände, die die mechanische Komponente im störungsfreien Fall einnehmen kann. Dabei kommt jeder Zustand im Zustandsgraphen nur einmal vor. Eine derartige Modellierung liegt bei der Beschreibung von elementaren Funktionseinheiten vor (Abbildung 6-15/links).

Reine zustandsorientierte Modellierung der Funktionseinheit Greifer



Dienstorientierte Modellierung der Baugruppe "Wechselarm"

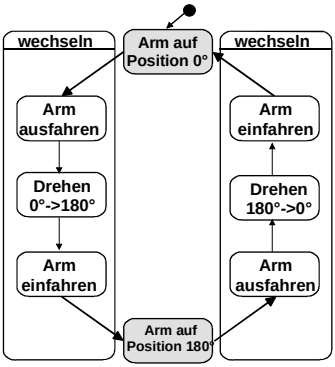


Abbildung 6-15: Rein zustandsorientierte und dienstorientierte Modellierung mechanischer Komponenten

Anders ist dies bei der dienstorientierten Modellierung. Hier werden zwar auch die einzelnen Zustände des technischen Systems anhand von Idle- und Running-Zuständen beschrieben. Dabei kann es aber vorkommen, daß in einem Zustandsgraphen derselbe Running-Zustand an mehreren Stellen vorliegt, weil er in unterschiedlichen Diensten benötigt wird. Diese Art der Modellierung liegt dann vor, wenn ein Koordinator einen komplexen Ablauf oder Prozeß beschreibt oder wenn mehrere elementare Funktionseinheiten eines technischen Systems zu einer Baugruppe zusammengefaßt werden und diese Baugruppe mittels einer einzigen

Steuerungsklasse bzw. eines einzigen Zustandsgraphen modelliert wird (Abbildung 6-15/rechts).

Zur einfachen Erstellung einer rein zustandsorientierten Modellierung einer mechanischen Komponente kann wie folgt vorgegangen werden: Zunächst müssen alle Idle- und Running-Zustände des technischen Systems spezifiziert und in einem Zustandsgraphen abgelegt werden. Anschließend werden die möglichen Zustandsübergänge festgelegt. Auf Basis der zulässigen Übergänge von Idle-Zuständen in Running-Zustände können die zur Verfügung gestellten Dienste der modellierten Komponenten definiert werden.

Für die einfache Erstellung einer dienstorientierten Modellierung einer mechanischen Komponente bietet sich folgender Ablauf an: Zum Aufbau des Zustandsgraphen werden zunächst alle Idle-Zustände der zu modellierenden mechanischen Komponente erfaßt. Anschließend erfolgt die Definition aller Dienste, die die Komponente bzw. Klasse zur Verfügung stellen soll. Je nach Komplexität des Dienstes können hierfür auch Dienstzustände (siehe Kapitel 6.2.3.2) eingesetzt werden. Für die Dienste werden anschließend die benötigten Running-Zustände beschrieben.

Bei der Modellierung einer mechanischen Komponente kann es durchaus vorkommen, daß derselbe Dienst mehrmals im Zustandsgraphen vorliegt und somit ein Zustandsübergang zwischen verschiedenen Idle-Zustands-Paaren ist. Abbildung 6-16 zeigt einen derartigen Fall. Während im Zustandsgraph eines hydraulischen Schwenkkopfes die beiden Idle-Zustände „Position 0°“ und „Position 180°“ eingetragen sind, stellt der Schwenkkopf nach außen nur den Dienst „Drehen um 180°“ zur Verfügung. Ob er dabei eine Links- oder Rechtsdrehung ausführt, bleibt für den Auftraggeber, der den Dienst aufruft, verborgen. Der Dienst selbst ist im Zustandsgraphen jedoch zweimal mit jeweils verschiedenen Running-Zuständen eingetragen.

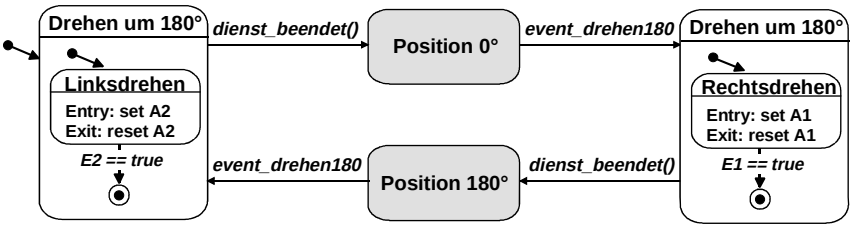


Abbildung 6-16: Doppelter Dienst in einem Zustandsgraphen

Spezifikation der zu erfassenden Störungen

Parallel zur Spezifikation der grundlegenden Steuerungsfunktionalität wird hier mit der Beschreibung der Störungsbehandlungsfunktionalität begonnen. Voraussetzung ist nur, daß die entsprechenden Idle- und Runnig-Zustände der betrachteten Steuerungsklasse bereits vorliegen.

In Abhängigkeit von den Idle- und Running-Zuständen des Zustandsgraphen werden alle Störungen spezifiziert, die von einer störungstoleranten Steuerungsklasse erkannt werden sollen. Die zu erfassenden Störungen werden mittels Störungserkennungstransitionen beschrieben, die in den Störungszustand münden. Mit den Transitionen werden Zeitüberschreitungen, fehlerhafte Signalkombinationen oder Störungsmeldungen von weiteren Komponenten erfaßt. Für die Störungserkennung mittels Zeitüberwachung müssen zudem für die betroffenen Running-Zustände noch die zulässigen Ausführungszeiten angegeben werden. Abbildung 6-17 zeigt die Modellierung der Funktionseinheit „Werkzeuggreifer“ sowie die Störungserkennungstransition des Dienstes „Werkzeug_greifen“.

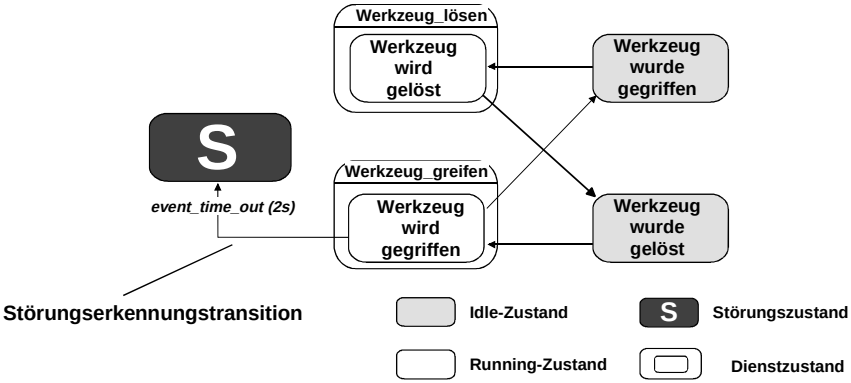


Abbildung 6-17: Störungserkennungstransition des Dienstes „Werkzeug_greifen“

Spezifikation der gewünschten Ursachenfindung

Nach der Angabe der von einer Steuerungsklasse zu erfassenden Störung wird spezifiziert, wie die Ursachenfindung für diese Störungen aussehen soll. Der Entwickler muß festlegen, welches Ursachenfindungsverhalten eine störungstolerante Steuerungsklasse aufweisen soll. Voraussetzung dafür ist, daß Struktur und Störungsverhalten der zu steuernden mechanischen Komponente bekannt sind.

Die klassische FMEA-Analyse (**F**ailure **M**ode and **E**ffect **A**nalysis oder **F**ehler**m**öglichkeits und -**e**influß**a**nalyse) ist eine analytische Methode zur systematischen und vollständigen Erfassung potentieller Fehler *(Reinhart u. a. 1996, S. 86–93).* Aus den Angaben wird dann abgeleitet, wie diese Fehler vermieden werden können. Mit Hilfe spezieller FMEA-Bögen werden systematisch Informationen über Fehlerursachen und Fehlerauswirkungen erfaßt und hinsichtlich ihrer Bedeutung bewertet.

Ähnlich der FMEA-Analyse werden hier mit Hilfe der in Kapitel 6.2.4 beschriebenen Ursachenfindungstabellen alle möglichen Ursachen erfaßt und festgehalten, die zu den von der Steuerung erfaßbaren Störungen führen können. Im Unterschied zur FMEA werden hier aber keine Vorschläge zur Fehlervermeidung gemacht.

In der hier vorgestellten Vorgehensweise wird für jede betrachtete Störung eine eigene Tabelle verwendet. Abbildung 6-18 zeigt die Ursachenfindungstabelle der Störung „Zeitüberschreitung“ des Dienstes „Werkzeug_greifen“ aus Abbildung 6-17.

Zeitüberschreitung beim Dienst Werkzeug_greifen		**Elementare Ursachen**									
		S2 verrutscht	S2 defekt	Kabelbruch (Sensor)	Klemme lose (Sensor)	Kabelbruch (Aktor)	Stellglied defekt	Klemme lose (Aktor)	Aktor klemmt	Leitung verstopft	Leitung leckt
Automatisch erfaßbare Symptome	Eingangssignal: gegriffen	0	0	0	0	1	1	1	1	1	1
	Eingangssignal: gelöst	0	0	0	0	0	0	0	0	0	0
	Öldruckgeber (low)	-	-	-	-	0	0	0	0	0	1
	Ölüberdruck	-	-	-	-	0	0	0	1	1	0
Manuell erfaßbare Symptome	Ist Sensor 1 aus oder an (0/1) ?	0	0	0	0	1	1	1	1	1	1
	Ist Sensor 2 aus oder an (0/1) ?	0	0	0	0	0	0	0	0	0	0
	Wurde das Werkzeug gegriffen (ja/nein) ?	1	1	1	1	0	0	0	0	0	0
Ursachenkategorie	"Es liegt ein Defekt im Geberstrang vor": Graphischer Hinweis auf mögliche Ursachen	X	X	X	X						
	"Es liegt ein Defekt im Stellgliedstrang vor": Graphischer Hinweis auf mögliche Ursachen					X	X	X			
	"Es liegt ein Defekt in der Aktorik vor": Graphischer Hinweis auf mögliche Ursachen								X	X	
	"Es liegt ein Defekt in der Versorgung vor": Graphischer Hinweis auf mögliche Ursachen										X

Abbildung 6-18: Aufbau einer Ursachenfindungstabelle des Dienstes „Werkzeug_greifen“

Zum Aufbau dieser Tabelle werden zunächst die verschiedenen Ursachen der betrachteten Störung in einzelnen Spalten aufgezählt. Anschließend werden in einzelnen Zeilen alle möglichen Symptome aufgelistet, die unmittelbar mit der Störung in Beziehung stehen. Dabei sind die von der Steuerung automatisch erfaßbaren Symptome von denen, die nur durch den Bediener erfaßt werden können, zu unterscheiden. Muß der Bediener mit einbezogen werden, können die Symptome auch als Fragen formuliert werden. Dies erleichtert die nachfolgende Erstellung der jeweiligen Fehlerbäume. In den Zellen der Ursachenfindungstabellen wird festgehalten, welches Symptom auf welche Störung hindeutet.

Nach dieser Festlegung werden anschließend alle Ursachen, die ähnliche Symptome aufweisen, in Ursachenkategorien zusammengefaßt. Dann wird für jede Kategorie eine Aktionszeile eingetragen, die diese Kategorie beschreibt. Auch hier ist die Beschreibung so zu formulieren, daß sie als Meldungstext dienen kann. Zusätzlich kann hier angegeben werden, welches Medium zur Bekanntgabe der Ursachenkategorie verwendet werden soll, z. B. Text, Hypertext oder bestimmte Graphiken.

Beim Erstellen von Ursachenfindungstabellen muß darauf geachtet werden, daß hier keine Angaben zu expliziten Funktionseinheiten, sondern nur zu den verschiedenen Typen von Funktionseinheiten gemacht werden. Dies liegt daran, daß es sich bei der Spezifikation um Angaben für eine Klasse und nicht für eine konkrete Instanz handelt. So dürfen in der Textbeschreibung beispielsweise keine Einträge wie „Fehler im vorderen Zylinder des Greifers“ vorkommen. Vielmehr muß es „Fehler im Zylinder vom Typ 4711“ heißen. Sind jedoch explizite Angaben nicht vermeidbar, können in den Steuerungsklassen hierfür Textkonstanten verwendet werden, die als Platzhalter dienen und zum Zeitpunkt der Instantiierung der Klassendefinition mit dem richtigen Text versehen werden.

Durch das hier beschriebene Vorgehen spezifiziert der Entwickler strukturiert und systematisch, welche Regeln während der Ursachenfindung abzuarbeiten sind. Diese Spezifikation kann sehr einfach in einen Fehlerbaum einer Steuerungsklasse umgewandelt werden, wie dies im Kapitel 6.3.2.3 „Feinspezifikation“ beschrieben wird.

Spezifikation der gewünschten Störungsbehebung

Letzte Aufgabe der Grobspezifikation einer störungstoleranten Steuerungsklasse ist die Erfassung der gewünschten Störungsbehebung, die die Klasse aufweisen soll. Dazu muß der Entwickler für jede Ursachenkategorie spezifizieren, wie die Störungsbeseitigung und der Recovery-Vorgang ablaufen sollen. In der hier vorgestellten Vorgehensweise steht dafür die Störungsbehebungstabelle zur Verfügung (siehe auch Kapitel 6.2.2.2). Dabei wird für jede Ursachenkategorie genau eine Störungsbehebungstabelle verwendet (Abbildung 6-19).

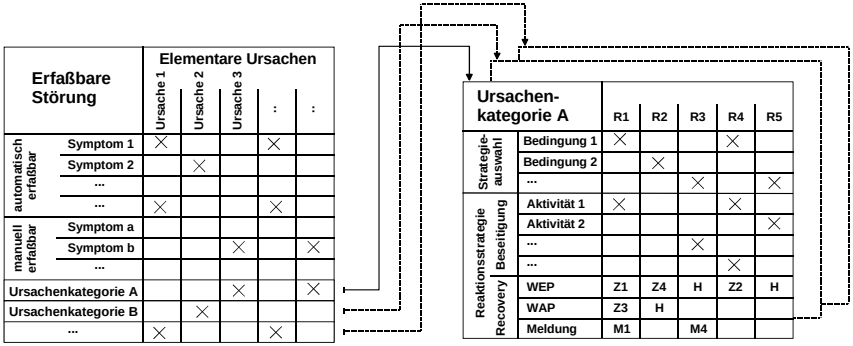


Abbildung 6-19: Pro Ursachenkategorie existiert eine eigene Behebungstabelle

Zunächst werden alle möglichen Reaktionsstrategien, die für eine Ursachenkategorie in Frage kommen, in einer Spalte der Tabelle festgehalten. Dabei bekommt jede Reaktionsstrategie eine eigene Identifikationsnummer. Der Entwickler spezifiziert dann die möglichen Wiedereinstiegs- und Wiederaufsetzpunkte (WEP und WAP). Hiermit verbunden sind auch Angaben darüber, ob die Steuerungsfunktionalität im betroffenen Steuerungsbaustein angehalten werden soll oder nicht.

Auf Basis der Angaben zum WEP werden dann in den einzelnen Zeilen die verschiedenen Störungsbeseitigungsaktivitäten spezifiziert, die für die Reaktionsstrategien zum Erreichen des WEP durchzuführen sind. Diese können zum einen automatisch ausführbare Aktivitäten wie die Manipulation bestimmter Variablen oder die Aktivierung von Sonderabläufen zur Störungsbeseitigung sein. Zum anderen können es aber auch Aktivitäten sein, die der Bediener zur Laufzeit ausführen soll. Für letztere werden Meldungen eingetragen, die sich als Anweisungen an den Bediener richten. Zur Festlegung der Abarbeitungsreihenfolge der Aktivitäten können in den Zellen auch Reihenfolgenangaben gemacht werden.

Eine besondere Bedeutung kommt an dieser Stelle Reaktionsstrategien zu, die ein Anhalten der Steuerungsabarbeitung oder das Zurückfahren von Funktionseinheiten in einen Ausgangszustand vorsehen. Mit diesen Reaktionsstrategien sind Rückmeldungen an den Auftraggeber verbunden, die das Fehlschlagen des Dienstes angeben. Die gewünschten Rückmeldungen werden in einer eigenen Zeile der Behebungstabelle angegeben.

Nach der Beschreibung der Reaktionsstrategien muß noch das Auswahlverfahren spezifiziert werden. Hierzu werden entweder bestimmte Systemvariablen abgefragt oder der Bediener zur Entscheidung herangezogen. Der Steuerungsentwickler spezifiziert die verschiedenen Bedingungen in einzelnen Zeilen und ordnet sie den Reaktionsstrategien zu. Für die Bedingungen, die in Abstimmung mit dem Bediener

erfolgen, werden in der Tabelle die zu stellenden Fragen eingetragen. Abbildung 6-20 zeigt beispielhaft, wie eine derartige Tabelle aussieht. Hier sind die verschiedenen Reaktionsstrategien für die Ursachenkategorie „Defekt im Geberstrang" im gestörten Dienst „Werkzeug_greifen" aus dem vorherigen Beispiel aufgezeigt.

Ursachenkategorie: Defekt im Geberstrang Dienst: Werkzeug greifen			**Reaktions-strategie Nr.1**	**Reaktions-strategie Nr. 2**	**Reaktions-strategie Nr. 3**	**Reaktions-strategie Nr. 4**
Reaktionsauswahl		Können Sie den Defekt beheben ?	ja	nein	nein	nein
		Kann der Defekt ignoriert werden ?	-	ja	nein	nein
		Soll in den Ausgangs-zustand gefahren und Aktion wiederholt werden ?	-	-	ja	nein
		Sind Sie sich sicher ?	ja	ja	ja	nein
Behebung	**Beseitigung**	Meldungen	-	Defekt wird kurzfristig ignoriert. Ablauf wird fortgesetzt!	Maschine fährt in den Ausgangszustand und wiederholt die Aktion!	Maschine wird angehalten!
		Bedieneraktionen	Bitte Defekt beheben und bestätigen!	-	-	-
	Recovery	Wiedereinstiegspunkt	Greifer geschlossen	Greifer geschlossen	Greifer geöffnet	Halt
		Wiederanlaufpunkt	Greifer geschlossen	Greifer geschlossen	Greifer geöffnet	-
		Nachricht an Auftraggeber	-	-	Server in Ausgangs-zustand: M = -1	Server angehalten. Funktionseinheit in Endzustand: M = -4

Abbildung 6-20: Behebungstabelle für die Ursachenkategorie „Defekt im Geberstrang" bei einer Störung im Dienst „Werkzeug_greifen".

Wie die spezifizierten Daten der beiden Störungsbehandlungstabellen in einen Fehlerbaum einer Steuerungsklasse umgewandelt werden, beschreibt die nachfolgende Feinspezifikation einer störungstoleranten Steuerungsklasse.

6.3.2.3 Feinspezifikation

Nach Beendigung der Grobspezifikation der geforderten störungstoleranten Steuerungsfunktionalität einer Steuerungsklasse muß diese nun verfeinert und softwaretechnisch entworfen werden. Dies ist der dritte Schritt in der hier vorgestellten Vorgehensweise zur Entwicklung störungstoleranter Steuerungen (siehe auch Abbildung 6-13).

Im Rahmen der Feinspezifikation werden einerseits die benötigten Eingangs- und Ausgangsvariablen bzw. die Ein- und Ausgänge der Steuerungsklassen spezifiziert. Andererseits erfolgt hier die Umwandlung der Daten der erstellten Störungs-

behandlungstabellen in das Modell der störungstoleranten Steuerungsklasse, d. h. in die entsprechenden Zustandsgraphen und Fehlerbäume (siehe auch Kapitel 6.2.3).

Wie die Spezifikation der Eingangs- und Ausgangsvariablen erfolgt, ist trivial und braucht hier nicht weiter detailliert zu werden. Wesentlich ist an dieser Stelle die Erläuterung, wie die Anforderungen an die Störungsbehandlungsfunktionalität in die störungstoleranten Steuerungsklassen integriert werden. Dafür werden folgende zwei Schritte unterschieden:

1. Ergänzung des erstellten Zustandsgraphen um Störungsbehandlungsfunktionalität und
2. Erstellung des Fehlerbaumes, der eine Detaillierung des Störungszustands darstellt.

Ergänzung des Zustandsgraphen um Störungsbehandlungsfunktionalität

Mit den beiden oben beschriebenen Tabellen wird festgelegt, welche Anforderungen hinsichtlich der Störungslokalisierung und Störungsbehebung an die einzelnen Steuerungsklassen gestellt werden. Ein Teil dieser Anforderungen ist nun im Zustandsgraphen abzubilden. Dabei sind hier nur die Angaben zum Recovery-Vorgang aus den Störungsbehebungstabellen von Interesse. Die restlichen Informationen betreffen den Fehlerbaum. Durch die Recovery-Angaben werden im Zustandsgraphen folgende Elemente eingetragen:

- Ausgangstransitionen des Störungszustands,
- Rückwärtspfade im Zustandsgraphen
- störungsbedingte Rückmeldungen an einen Client-Steuerungsbaustein.

Jede Reaktionsstrategie stellt mit ihren Recovery-Angaben einen individuellen Ablauf dar, der sich im Zustandsgraphen vom Störungszustand bis zum Wiederaufsetzpunkt (WAP) erstreckt. Für diese Abläufe ist jeder Zustandsgraph mit individuellen Ausgangstransitionen aus dem Störungszustand zu erweitern, die in die entsprechenden Wiedereinstiegspunkte (WEP) münden. Abbildung 6-21 zeigt die vier Reaktionsstrategien samt den zugehörigen WEP und WAP aus Abbildung 6-20, eingezeichnet im Zustandsgraphen aus Abbildung 6-17.

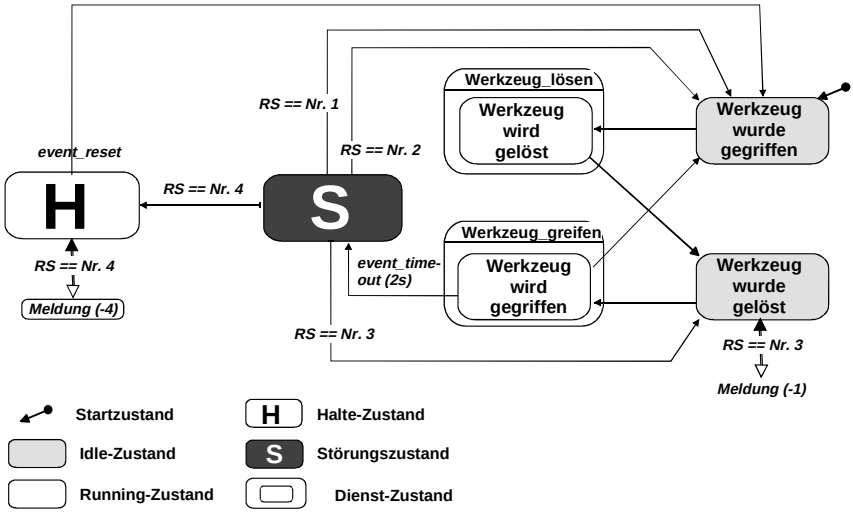


Abbildung 6-21: Ergänzungen im Zustandsgraphen für die Störungsbehandlung

Wurde für eine Reaktionsstrategie als WEP ein Running-Zustand spezifiziert und gleichzeitig für den WAP der Ausgangszustand des Dienstes angegeben, wird im Zustandsgraphen ein Ablauf benötigt, der vom WEP in den Ausgangszustand führt. Verfügt der Zustandsgraph über einen Dienst, der eine derartige Verbindung enthält, ist keine Erweiterung notwendig. Sonst wird ein Rückwärtspfad benötigt (siehe Kapitel 5.3.5). Diese Ergänzungen sind primär in Steuerungsbausteinen notwendig, die eine dienstorientierte Modellierung enthalten, wie dies bei einem Koordinationsbaustein der Fall ist.

Als WAP liegt am Ende einer Reaktionsstrategie immer ein Idle- oder Haltezustand vor. Soll hier der Auftraggeber, d. h. der aufrufende Steuerungsbaustein, mit in die Störungsbehandlung eingebunden werden, sind diesem entsprechende Meldungen zu schicken. Diese Meldungen werden als bedingte Entry-Aktionen an die WAP angehängt (Abbildung 6-21).

Erstellung des Fehlerbaumes

Zusätzlich zu den Ergänzungen, die im Zustandsgraphen vorzunehmen sind, müssen in der Feinspezifikation die restlichen Informationen der beiden Störungsbehandlungstabellen herangezogen werden, um den Fehlerbaum zu erstellen. Der Fehlerbaum kann komplett aus den Angaben der Ursachenfindungs- und Störungsbehebungstabellen einer Steuerungsklasse generiert werden.

Aus den Ursachenfindungstabellen werden die Informationen gewonnen, die für den Ursachenfindungs- und Ursachenbekanntgabebereich im Fehlerbaum notwendig sind (Abbildung 6-22/oben). Pro Tabelle entsteht im STB-Aktivitätsdiagramm eine baumartige Struktur, deren Wurzel ein Aktionszustand ist, der die eingetretene Störung angibt, und deren Blätter Aktionszustände sind, die die entsprechende Ursachenkategorie bekanntgeben. Zwischen diesen Zuständen werden alle Symptomabfragen der Ursachentabellen mittels Verzweigungspunkten und Aktionszuständen zur Erfassung von Bedienereingaben abgebildet.

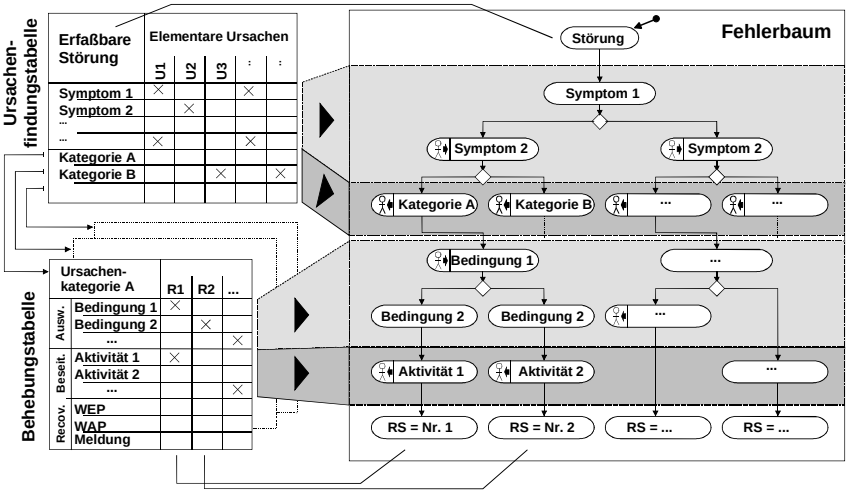


Abbildung 6-22: Generierung des Fehlerbaums aus den Tabelleninformationen

Aus der Behebungstabelle werden die Informationen für die Strategieauswahl und die Strategieaktivierung im Fehlerbaum gewonnen (Abbildung 6-22/unten). Zunächst werden im Fehlerbaum für die verschiedenen Reaktionsstrategien jeweils Aktionszustände eingetragen, die das Ende eines Zweiges im Fehlerbaum darstellen. In diesem Aktionszustand wird festgelegt, welche Reaktionsstrategienummer **(RS)** im Zustandsgraphen nach Beenden des Fehlerbaumes vorliegen soll. Damit wird auch spezifiziert, welche Ausgangstransition des Störungszustands feuert und welche Meldung vom zugehörigen WAP verschickt wird. Zwischen den Aktionszuständen zur Bekanntgabe der Ursachenkategorien und dem Entblättern des Fehlerbaumes werden die Bedingungen und Aktivitäten der Behebungstabellen mittels Verzweigungspunkten und verschiedener Aktionszustände abgebildet.

6.3.2.4 Instantiierung und Verknüpfung einer kompletten Anwendung

Der letzte Schritt der Modellierung einer störungstoleranten Steuerungsanwendung ist die Instantiierung der spezifischen Steuerungsklassen und ihre Verknüpfung.

Durch die Instantiierung einer Steuerungsklasse wird ein konkreter Steuerungsbaustein spezifiziert. Wurde beispielsweise eine Steuerungsklasse zur Steuerung eines hydraulischen Zylinders vom Typ 4711 modelliert, wird mit der Instantiierung festgelegt, für welchen realen Zylinder der entsprechende Steuerungsbaustein eingesetzt wird. Hier kann es durchaus vorkommen, daß eine Steuerungsklasse mehrfach zur Steuerung unterschiedlicher Maschinenkomponenten instantiiert wird.

Nach der Instantiierung müssen die Steuerungsbausteine miteinander verknüpft werden. Dadurch wird festgelegt, welcher Baustein auf welchen Dienst eines anderen Bausteins zurückgreift und welche Eingangs- und Ausgangsvariablen davon im einzelnen betroffen sind.

Nach dem letzten Schritt der Modellierung störungstoleranter Steuerungsanwendungen steht ein komplettes Softwaremodell der zu entwickelnden Steuerung zur Verfügung. Diese Modell ist die Ausgangsinformation der im folgenden beschriebenen Implementierungsphase.

6.3.3 Implementierung einer störungstoleranten Anwendung

Grundsätzlich kann Software, die mittels eines objektorientierten Modells spezifiziert wurde, **translativ** oder **elaborativ** Implementierung werden *(Douglass 1998, S. 204).* Worin sich die beiden Varianten unterscheiden und in welchem Zusammenhang diese Implementierung mit störungstoleranten Anwendungen steht, wird im folgenden kurz beschrieben (Abbildung 6-23).

Translative Implementierung

Unter translativer Implementierung ist zu verstehen, daß die Elemente eines Softwaremodells direkt in ein ausführbares Programm transformiert werden können. Dazu wird entweder ein entsprechender Codegenerator verwendet, der den aus dem Modell noch zu kompilierenden Code in einer geeigneten Programmiersprache generiert. Oder es wird eine virtuelle Maschine eingesetzt, die in Kombination mit dem Modell das zur Laufzeit ausführbare Programm ist. Diese Variante der Implementierung ist die wesentlich effektivere der beiden Implementierungsvarianten. Nachteilig ist nur, daß die hier eingesetzten Tools meist auf ganz bestimmte Anwendungen zugeschnitten sind und somit für die Implementierung beliebiger Softwareapplikationen, wie dies bei Standard-Entwicklungsumgebungen der Fall ist, nicht geeignet sind.

Für die translative Implementierung störungstoleranter Steuerungen sind Tools erforderlich, die auf Basis von Zustandsgraphen und Fehlerbäumen den notwendigen Quelltext bzw. Maschinencode für eine Steuerungsplattform automatisch generieren. Ein derartiges Tool wurden im Rahmen dieser Arbeit entwickelt und wird in Kapitel 7 näher erläutert.

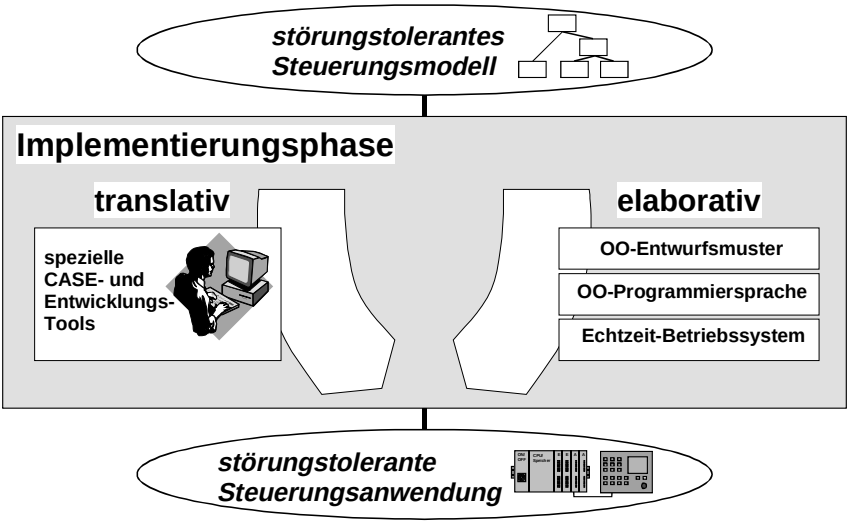


Abbildung 6-23: Implementierungsvarianten störungstoleranter Anwendungen

Elaborative Implementierung

Im Falle einer elaborativen Implementierung wird das erarbeitete Softwaremodell nur als ein detailliertes Analysemodell betrachtet. Dieses Modell muß nochmals weiter detailliert werden, bis es strukturell nur noch aus einzelnen Klassen und Instanzen besteht. Erst dann wird die Software in einer entsprechenden objektorientierten Programmiersprache plattformspezifisch implementiert.

Für das Softwaremodell der störungstoleranten Steuerungsanwendung heißt dies konkret, daß der Zustandsgraph und der Fehlerbaum eines jeden störungstoleranten Steuerungsbausteines zunächst anhand einzelner Klassen und Instanzen beschrieben werden müssen. Anschließend kann eine Implementierung in einer objektorientierten Programmiersprache für ein bestimmtes Echtzeitbetriebssystem erfolgen.

Zur Unterstützung des Feinentwurfes und der anschließenden Implementierung kann hier auf sogenannte Design-Patterns, z. B. nach *Gamma u. a. (1995)* oder *Douglass*

(1998, S. 203–332), sowie auf Implementierungsanleitungen für Echtzeitanwendungen, z. B. nach *Awad u. a. (1996 S. 155 ff.)* zurückgegriffen werden. Diese verschiedenen Anleitungen erläutern beispielsweise, wie vereinfachte Zustandsgraphen mittels Klassen spezifiziert und anschließend ausprogrammiert werden können.

Eine weitere Unterstützung der Implementierung ist die Verwendung bestehender Frameworks zur Entwicklung verteilter Anwendungen. Diese Frameworks müssen aber im klassenbasierten Feinentwurf des Softwaremodells berücksichtigt werden. Beispiele für derartige Frameworks sind die CORBA-Architektur *(Sayegh 1997)* und die OSACA-Kommunikationsplattform für „offene Steuerungen“ *(Pritschow 1996)* (siehe auch Kapitel 2.3.2).

6.3.4 Zusammenfassung

In diesem Kapitel wurde die zweite Säule der hier zu erarbeitenden Methode zur Entwicklung störungstoleranter Steuerungen aufgezeigt. Es wurde das methodische Vorgehen zur Entwicklung störungstoleranter Anwendungen erläutert. Dieses umfasst zunächst eine Beschreibungstechnik, die auf der UML basiert und für alle Phasen der Entwicklung verwendet werden kann. Des weiteren beinhaltet es eine detaillierte Vorgehensweise, die aufzeigt, wie die Beschreibungstechnik in verschiedenen Phasen und Schritten zur systematischen Entwicklung störungstoleranter Anwendungen eingesetzt werden kann.

7 Prototypische Realisierung und beispielhafter Einsatz

7.1 Übersicht

In diesem Kapitel wird zunächst das prototypisch realisierte Werkzeug zur methodischen Entwicklung störungstoleranter Steuerungen beschrieben. Anschließend wird hier ein beispielhafter Einsatz einer störungstoleranten Steuerung aufgezeigt.

7.2 Realisiertes Entwicklungswerkzeug

Für eine translative Implementierung von Software sind, wie bereits im Kapitel 6.3.3 beschrieben, entsprechende Entwicklungstools erforderlich, die direkt aus einem Softwaremodell ausführbare Programme erstellen können.

Im Rahmen dieser Arbeit wurde ein derartiges Tool zur translativen Implementierung von störungstoleranten Steuerungsanwendungen entwickelt *(Reinhart u. a. 1999)*. Dieses Tool, das hier als *Siwes-Tool* bezeichnet wird, besteht zum einen aus einer graphischen Entwicklungsumgebung zur Projektierung von störungstoleranten Steuerungsanwendungen und zum anderen aus einer Laufzeitumgebung, auf der die erstellten Abläufe abgearbeitet werden (Abbildung 7-1). Im folgenden werden diese zwei Bereiche einzeln beschrieben.

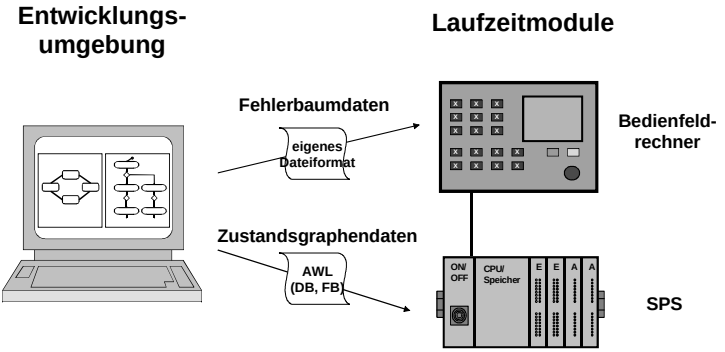


Abbildung 7-1: Aufbau des Siwes-Tools

7.2.1 Entwicklungsumgebung

Die Siwes-Entwicklungsumgebung, die auf Basis des Werkzeuges HiGraph[5] aufgebaut wurde, ermöglicht die Erstellung von störungstoleranten Steuerungsanwendungen, die softwaretechnisch nach dem in Kapitel 5 beschriebenen Referenzkonzept aufgebaut sind.

Zur Beschreibung der Steuerungsanweisungen werden graphische Beschreibungstechniken eingesetzt (siehe auch Kapitel 6.2). Dazu stellt die Siwes-Entwicklungsumgebung drei Editoren zur Verfügung. Diese sind ein Klassendiagramm-Editor, ein Zustandsgraphen-Editor und ein Fehlerbaum-Editor.

Mit dem Klassendiagramm-Editor wird die logische Struktur der zu entwickelnden Steuerungsanwendung beschrieben. Hier können störungstolerante Steuerungsklassen spezifiziert, instantiiert und verknüpft werden.

Der Zustandsgraphen-Editor (Abbildung 7-2/links) ermöglicht es, für jede Steuerungsklasse einen zugehörigen Zustandsgraphen zu erstellen, der das Steuerungsverhalten der Steuerungsklasse beschreibt. Im Zustandsgraphen-Editor werden Idle- und Running-Zustände unterschieden, mit denen die Dienste der Steuerungsklasse beschrieben werden. Außerdem wird für jeden Zustandsgraph ein Störungszustand definiert, der mit dem Fehlerbaum der Steuerungsklasse funktionstechnisch verknüpft ist, sowie ein Halte-Zustand, der das Anhalten der Steuerungsabarbeitung modelliert.

Mit dem Fehlerbaum-Editor (Abbildung 7-2/rechts) können die verschiedenen Abläufe zur Störungslokalisierung und Störungsbeseitigung aufgebaut werden. Dabei stehen dem Entwickler zahlreiche Drag&Drop-Elemente zur Verfügung, die er als Aktivitäten in den Fehlerbaum einsetzen kann. Mit diesen Elementen können unter anderem für die Laufzeitumgebung komplexe Sequenzen von Bedienerdialogen aufgebaut werden.

Zur Vereinfachung der Entwicklung und zur Vermeidung von Eingabefehlern sind der Zustandsgraphen-Editor und der Fehlerbaum-Editor funktional miteinander verknüpft. So wird für jede Transition in einem Zustandsgraphen, die in den Störungszustand mündet, automatisch eine neue Quelle im zugehörigen Fehlerbaum eingetragen. Dadurch erhält jede Störungserkennungstransition automatisch eine entsprechende Startstelle im Fehlerbaum. Gleichzeitig verfügt der Fehlerbaum über ein Eingabefester zur Beschreibung des zum Schluß einer Störungsbehandlung benötigten Recovery-Ablaufs. Mit diesem Dialog wird zum einen der letzte Aktivitätszustand eines Zweiges im Fehlerbaum eingetragen, zum anderen erfolgen hier automatisch die noch anstehenden Erweiterungen im Zustandgraphen zur

[5] HiGraph: Zustandsgraphenbasierte Entwicklungsumgebung der Fa. Siemens zur Programmierung von SPSen *(Braun 1999)*

Beschreibung des zugehörigen Recovery-Ablaufs, wie z. B. das Hinzufügen der Transition, die aus dem Störungszustand in den Wiedereinstiegspunkt mündet.

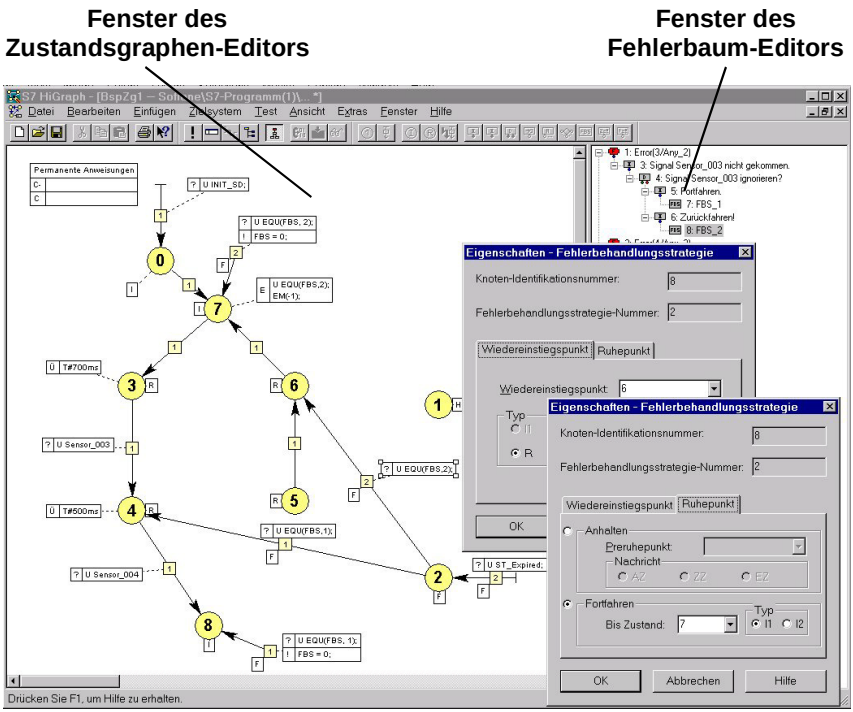


Abbildung 7-2: Editoren der Entwicklungsumgebung des Siwes-Tools

Am Ende der Projektierung einer störungstoleranten Steuerungsanwendung erstellt die Entwicklungsumgebung aus den spezifizierten Diagrammen zwei Sorten von Ausgabedaten. So werden zum einen aus allen Zustandsgraphen Steuerungsanweisungen in Form von AWL-Code generiert. Zum anderen werden die Daten der Fehlerbäume in sogenannten Fehlerbaumdateien abgelegt. Beide Ausgabedaten stehen dann den Laufzeitmodulen zur Verfügung.

7.2.2 Laufzeitmodule

Die Laufzeitmodule des Siwes-Tools bestehen aus einer S7-SPS sowie einem Bedienfeldrechner, auf dem speziell mit Hilfe des Werkzeuges WinCC[6] entwickelte Programme laufen (Abbildung 7-3). Während die SPS zur Steuerung an die Aktoren und Sensoren einer Maschine angeschlossen wird, dient der Bedienfeldrechner als Kommunikationsschnittstelle zum Bediener.

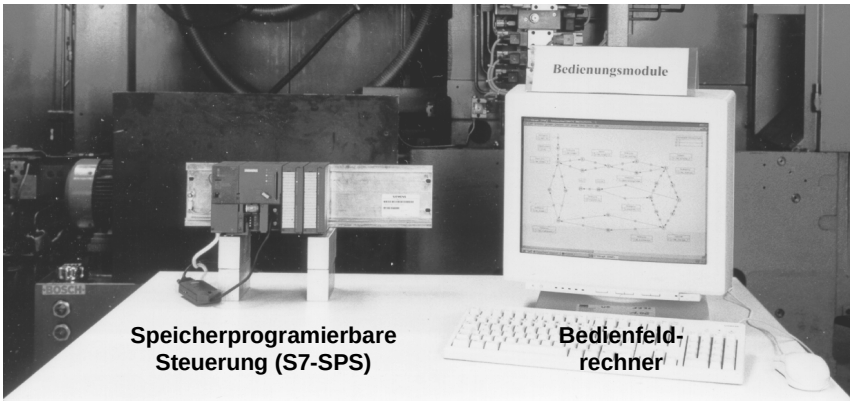


Abbildung 7-3: Laufzeitmodule des Siwes-Tools

Die mit der Entwicklungsumgebung erstellten Daten werden zur Kompilierzeit auf diese zwei Plattformen aufgeteilt. Zur Laufzeit werden auf der SPS die AWL-Steuerungsanweisungen abgearbeitet. Der Bedienfeldrechner stellt dagegen einen speziellen Interpreter zur Verfügung, der die Anweisungen aus den Fehlerbaumdateien abarbeiten kann. Dabei können unterschiedliche Dialoge mit dem Bediener geführt werden.

Die Zusammenarbeit dieser zwei Plattformen sieht zur Laufzeit wie folgt aus (Abbildung 7-4):

Wird während der Abarbeitung eines Zustandsgraphen in der SPS eine Störung erkannt, wird zunächst eine Störungsmeldung an den Bedienfeldrechner geschickt. Dabei verharrt die SPS im Störungszustand des entsprechenden Zustandsgraphen. Mit der Störungsmeldung, die der Bedienfeldrechner empfängt, wird der Fehlerbauminterpreter angestoßen. In Abhängigkeit von den entsprechenden Anweisungen

[6] WinCC: Programmierbares Bedien- und Beobachtungssystem der Fa. Siemens *(Siemens 1997)*

der zugehörigen Fehlerbaumdatei werden nun SPS-Variablen manipuliert und entsprechende Bedienerdialoge aufgebaut. Am Ende der Abarbeitung des Fehlerbaumes wird die Interpretation angehalten und eine Meldung an die SPS geschickt. Mit dieser Meldung wird der Störungszustand des entsprechenden Zustandsgraphen verlassen und der anschließende Recovery-Vorgang gestartet.

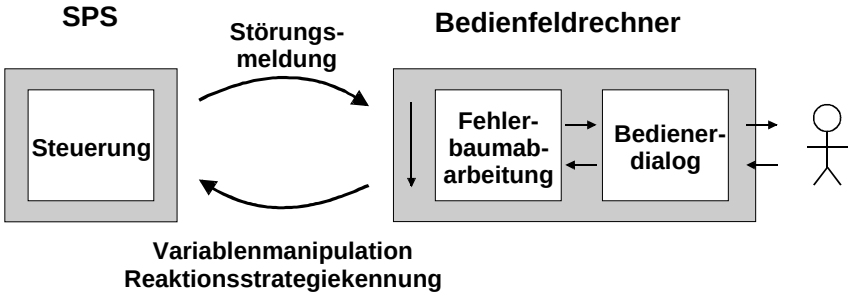
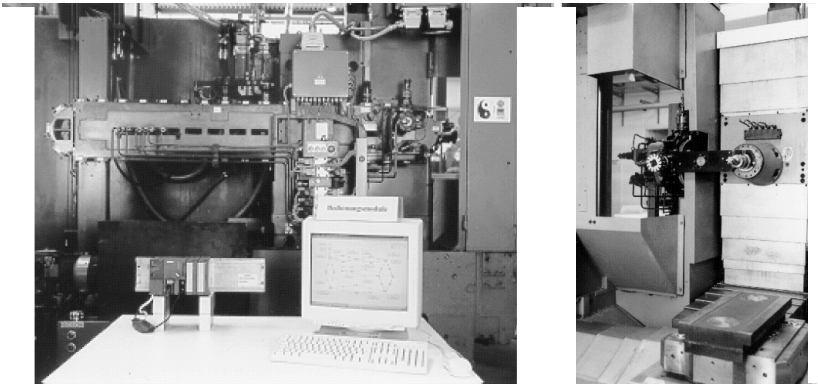


Abbildung 7-4: Zusammenarbeit von SPS und Bedienfeldrechner

7.3 Einsatzbeispiel

7.3.1 Aufbau der Versuchsmaschine

Zur Validierung der hier erarbeiteten Methode zur Entwicklung störungstoleranter Steuerungen wurde eine störungstolerante Steuerungsanwendung für ein Bearbeitungszentrum beispielhaft aufgebaut. Als Einsatzbeispiel dient ein Fräsbearbeitungszentrum der Firma Deckel (DC 30). Hier wurde die SPS der Maschine umgangen und mit Hilfe der Laufzeitmodule des Siwes-Tools das Werkzeugwechselaggregat der Maschine störungstolerant gesteuert. Abbildung 7-5 zeigt den mechanischen Aufbau des Werkzeugwechselaggregats und die Laufzeit-module des Siwes-Tools.



Seitenansicht **Frontansicht**

Abbildung 7-5: Werkzeugwechselaggregat als Einsatzbeispiel

7.3.2 Modellierung der Steuerungsaufgabe

Der zur steuernde Vorgang besteht der Reihenfolge nach aus folgenden Schritten:

1. Bereitstellung des einzuwechselnden Werkzeugs durch Vor- oder Rückwärtslauf des Kettenmagazins;
2. Öffnen der Magazintür, die den Bearbeitungsraum vom Wechsleraggregat samt zugehörigem Magazin trennt;
3. Drehung des Schwenkarms um 90° zur Durchführung des eigentlichen Wechselvorgangs. Dadurch greift der Wechselarm gleichzeitig die Werkzeuge im Kettenmagazin und in der Spindel;
4. Verriegelung der Werkzeuge im Werkzeuggreifer und Entriegelung im Kettenmagazin sowie in der Spindel;
5. Ausfahren der Werkzeuge mit dem Wechselarm aus dem Magazin und der Spindel;
6. Drehung des Wechselarms um 180°, um die beiden Werkzeuge auszutauschen;
7. Einfahren des Wechselarms, um die Werkzeuge wieder in das Magazin und die Spindel einzusetzen;
8. Verriegelung der Werkzeuge in der Spindel und im Kettenmagazin sowie Entriegelung im Werkzeuggreifer;
9. Zurückdrehen des Schwenkarms um 90° und Beenden des Wechselvorgangs.

Zur störungstoleranten Steuerung des Werkzeugwechselaggregats ist die Steuerungssoftware in einzelne, störungstolerante Steuerungsbausteine gegliedert, die jeweils einen eigenen Zustandsgraphen und Fehlerbaum besitzen. Dabei ist jeder Baustein für die autarke Ansteuerung einer einzelnen Funktionseinheit des Wechselaggregats zuständig. Unterschieden werden die Steuerungsbausteine der Funktionseinheiten *Greifer, Wechselarm, Schwenkarm, Spindel, Kettenmagazin* und *Schutztür*. Darüber hinaus wird ein weiterer Baustein für die Koordination der einzelnen Funktionseinheiten verwendet (Abbildung 7-6). Im folgenden wird die Modellierung der Bausteine „Koordinator", „Wechselarm" und „Greifer" erläutert.

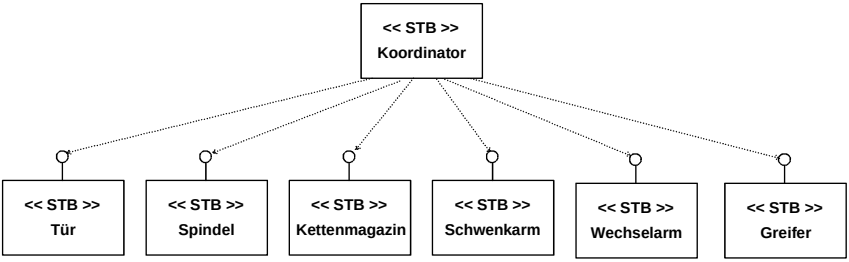


Abbildung 7-6: Struktur und Bausteine der Steuerungssoftware

Den Aufbau des Zustandsgraphen des Bausteins „Koordinator" zeigt Abbildung 7-7. Dieser Zustandsgraph, der zunächst keine Störungsbehandlungsangaben enthält, ist für die Gesamtfunktionalität des Werkzeugwechselaggregats zuständig und stellt nach außen zwei Dienste zur Verfügung. Diese Dienste werden von der NC-Steuerung aufgerufen. Der erste Dienst (v*orbereiten*) dient der Bereitstellung der geforderten Werkzeuge und führt vom ersten Idle-Zustand nach der Initialisierung (*Ausgangszustand*) zum nächsten Idle-Zustand (*WZ-bereit*). Mit diesem Dienst wird das gewünschte Werkzeug im Kettenmagazin an die Wechselposition gefahren. Hierfür greift der Koordinator im Running-Zustand (*WZ-positionieren*) auf einen Dienst des Werkzeugmagazins zurück. Der zweite Dienst des Koordinators (w*echseln*) ist der Wechselvorgang und führt vom Idle-Zustand *WZ-bereit* wieder in den *Ausgangszustand* zurück. Hierfür greift der Koordinator in seinen Running-Zuständen *Tür-öffnen, Arm-einschwenken, WZ-greifen, Greifer-drehen, WZ-lösen, Arm-ausschwenken* und *Tür-schließen* auf die Dienste der untergeordneten Bausteine zurück. Dadurch wird das Werkzeug aus dem Magazin entnommen und mit dem Werkzeug aus der Spindel ausgetauscht.

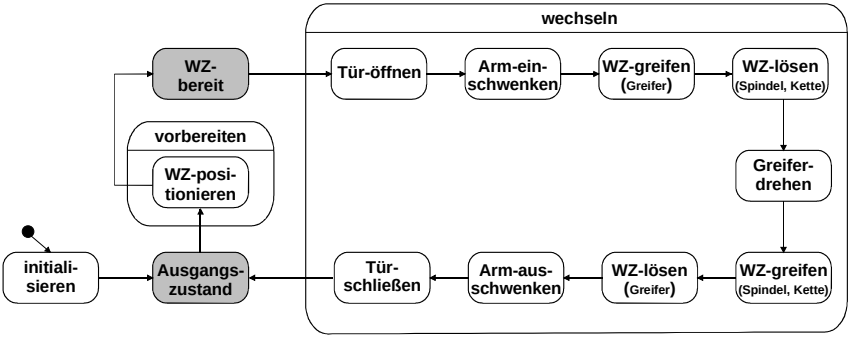


Abbildung 7-7: Der Zustandsgraph des Koordinators ohne Störungsbehandlung

In Abbildung 7-8 sind vereinfacht und zunächst ohne Störungsbehandlung die Zustandsgraphen der Steuerungsbausteine für die Funktionseinheiten *Greifer* und *Wechselarm* dargestellt. Hier bietet der Steuerungsbaustein des *Greifers* die beiden Dienste *WZ-greifen* und *WZ-lösen* an. Dazu enthält der Zustandsgraph zusätzlich zum Initialisierungszustand die zwei Idle-Zustände *gegriffen* und *gelöst* sowie die beiden Running-Zustände *schließt* und *öffnet*, die direkt auf die Stellglieder der Funktionseinheit zurückgreifen. Der Baustein der Funktionseinheit *Wechselarm* bietet dagegen nach außen nur einen einzigen Dienst an. Dies ist der Dienst *Wechseln*, mit dem ein Werkzeug aus der Wechselposition im Kettenmagazin gegen das Werkzeug in der Spindel ausgetauscht wird. Für den Dienstaufrufenden, d. h. den Koordinator, ist es unerheblich, ob dabei eine Links- oder eine Rechtsdrehung des Wechselarms erfolgt. Diese Detailinformation steckt im Zustandsgraphen der zu steuernden Funktionseinheit. Im Graphen ist der Dienst zweimal eingetragen und führt vom Idle-Zustand *Arm-auf-Position-0°* zum Idle-Zustand *Arm-auf-Position-180°* sowie vom Zustand *Arm-auf-Position-180°* zurück zum Zustand *Arm-auf-Position-0°*. Bei der Dienstausführung werden immer drei Running-Zustände durchlaufen, die direkt auf die Stellglieder der Funktionseinheit *Wechselarm* zurückgreifen. Durch die Running-Zustände wird der Wechselarm ausgefahren, um 180° gedreht (links oder rechts) und dann wieder eingefahren (Abbildung 7-8).

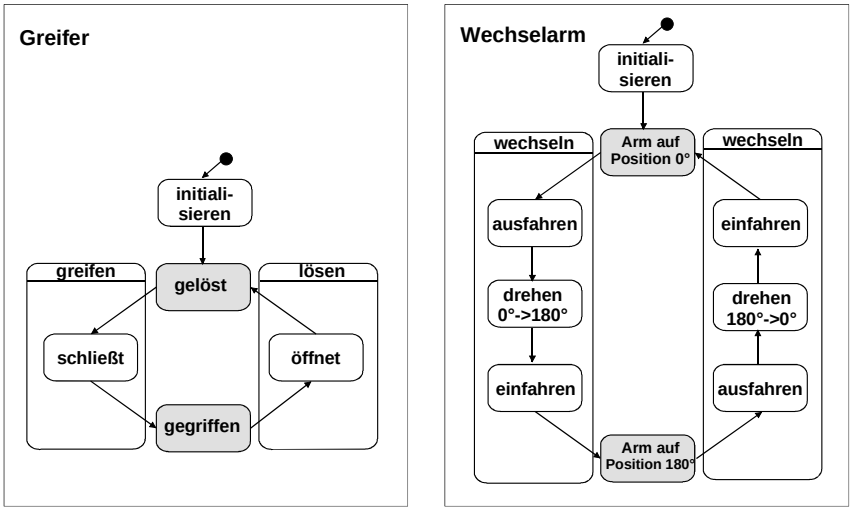


Abbildung 7-8: Die vereinfachten Zustandsgraphen der Funktionseinheiten „Wechselarm“ und „Greifer“

7.3.3 Modellierung der Störungsbehandlungsfunktionalität

Für die Störungsbehandlung in den Steuerungsbausteinen sind in den einzelnen Zustandsgraphen noch weitere Zustände und Transitionen eingetragen. Außerdem werden Fehlerbäume verwendet, die für die Störungslokalisierung und -behebung zuständig sind. Im folgenden sollen zwei modellierte Störungsbehandlungsstrategien näher erläutert werden.

7.3.3.1 Störung beim Ausfahren des Greifarms

Zur Behandlung von Störungen beim Ausfahren des Wechselarms während des Dienstes *wechseln* wird die Steuerungssoftware des Steuerungsbausteines „Greifer“ wie folgt modelliert (Abbildung 7-9). Zum Abfangen von Störungen sind im Zustandsgraphen an den Running-Zuständen, die das Ausfahren des Wechselarms bewirken, jeweils Erkennungstransitionen mit Zeitüberwachung angebracht. Im zugehörigen Fehlerbaum sind für diese Transitionen entsprechende Quellen eingetragen. Da mit der Zeitüberwachung nicht erfaßt werden kann, ob die Störungsursache im Geberstrang oder im Aktorstrang liegt, muß dies zunächst näher untersucht werden. Um die Störungsursache zu lokalisieren, ist im Fehlerbaum eine Frage an den Bediener eingetragen. Er wird gefragt, ob der Greifarm ausgefahren wurde oder nicht.

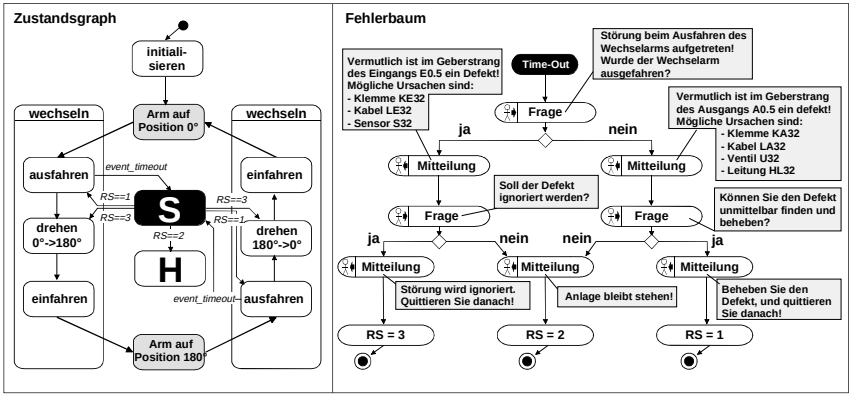


Abbildung 7-9: Zustandsgraphen- und Fehlerbaumeinträge für eine Störung beim Ausfahren des Wechselarms

Teilt der Bediener der Steuerung mit, daß der Greifer nicht ausgefahren wurde, liegt es nahe, daß die aufgetretene Störung in der Aktorik oder dem entsprechenden Stellgliedstrang liegt. Diese Vermutung wird dem Bediener in einem weiteren Dialog, der ebenfalls im Fehlerbaum eingetragen ist, mitgeteilt. Als weitere Reaktion auf diese Störung ist die Steuerung so programmiert, daß, falls der Bediener die Störung nicht sofort beheben kann, der Steuerungsbaustein in den Halte-Zustand versetzt wird. Es erfolgt eine Meldung an den Bediener, die ihn über mögliche Störungsursachen informiert und gleichzeitig fragt, ob er die Störung vor Ort beheben kann oder nicht. Kann er sie vor Ort beheben, wird er explizit aufgefordert, dies zu tun und anschließend die erfolgreiche Behebung zu quittieren. Mit der Quittierung wird im Fehlerbaum die RS-Variable des Zustandsgraphen auf einen Wert gesetzt, der die Ausgangstransition vom Störungszustand in den Running-Zustand *ausfahren* feuern läßt (RS = 1). Dadurch kann dieser Running-Zustand erneut aktiviert und der Greifer ausgefahren werden. Kann der Bediener die Störung jedoch nicht vor Ort beheben, wird ihm mitgeteilt, daß der Steuerungsbaustein in den Halte-Zustand geht. Hier wird die RS-Variable auf einen Wert gesetzt, der die Ausgangstransition des Störungs-Zustands feuern läßt, die in den Halte-Zustand mündet (RS = 2). Im Halte-Zustand wird dann eine Meldung an den Koordinator geschickt, die ihn über das Halte-Ereignis im Baustein des Wechselarms informiert und das Anhalten des Koordinators und der restlichen Bausteine veranlaßt.

Teilt der Bediener der Steuerung mit, daß der Greifer ausgefahren wurde, muß die Störung wohl im Signalgeberstrang liegen. Auch diese Information wird dem Bediener mitgeteilt. Nach Quittierung der Mitteilung wird der Bediener gefragt, ob aus mechanischer Sicht mit dem Dienst *Wechseln* fortgefahren werden kann. Verneint der Bediener die Antwort, wird analog zum vorherigen Vorgehen die Steuerung

angehalten und der Halte-Zustand angefahren (RS = 2). Beantwortet der Bediener die Frage mit einem „Ja“, heißt das, daß die Steuerung das fehlende Eingangssignal zur Quittierung des Ausfahrens des Wechselarms ignorieren soll. Folglich wird im Fehlerbaum die RS-Variable auf denjenigen Wert gesetzt, der die Ausgangstransition feuern läßt, die in den nächsten Running-Zustand nach dem Zustand *ausfahren* mündet (RS = 3). Die Steuerung fährt nun mit dem Werkzeugwechsel fort, als ob es zu keiner Störung gekommen wäre. Da hier jedoch die eigentliche Störungsursache nicht behoben wurde, kommt es beim nächsten Aufruf des Dienstes *wechseln* erneut zur Zeitüberschreitung und der Aktivierung dieser Störungsbehandlungsstrategie.

7.3.3.2 Störung beim Greifen des Werkzeuges

Zur Behandlung von Störungen beim Greifen von Werkzeugen im Kettenmagazin bzw. in der Spindel wird die Software wie folgt modelliert. Dabei ist hier eine Einbeziehung des übergeordneten Koordinators in den Störungsbehandlungsablauf erforderlich (Abbildung 7-10).

Im Zustandsgraphen des Steuerungsbausteins des Greifers führt eine Zeitüberwachungstransition vom Running-Zustand *greifen* in den entsprechenden Störungszustand. Auch hier können die Ursachen für die Zeitüberschreitung im Geber- oder im Aktorstrang der Greifervorrichtung liegen. Analog zum oben beschriebenen Fehlerbaum werden hier Informationen vom Bediener erfaßt, die Auskunft darüber geben sollen, wo die vermutete Ursache liegt und wie darauf reagiert werden soll. Hier werden zwei besondere Fälle beschrieben.

Ist die Ursache für die Störung die Verklemmung des auszuwechselnden Werkzeugs, könnte der Greifer zunächst geöffnet, das Werkzeug richtig plaziert und anschließend der Vorgang wiederholt werden. In bezug auf den Dienst *greifen* würde dies bedeuten, daß der Dienst zunächst revidiert würde, um anschließend erneut aufgerufen zu werden. Im Zustandsgraphen des Greifers führt deshalb die Reaktionsstrategie (RS = 6) in den Running-Zustand *lösen*, durch den der Vorgang zunächst revidiert wird (Abbildung 7-10). Mit Hilfe der bedingten Eintrittsaktion im Zustand *gelöst* wird eine Meldung an den Koordinator geschickt, die mitteilt, daß der Dienst fehlgeschlagen ist und daß sich der Greifer im Ausgangszustand des geforderten Dienstes befindet (EM=-1). Zum Abfangen dieser Störungsmeldung gibt es im Koordinator eine Überwachungstransition, die in den Störungs-Zustand des Koordinators mündet und durch die Meldung EM==-1 feuert. Dadurch wird der Fehlerbaum des Koordinators aktiviert. Im Fehlerbaum ist keine weitere Ursachenfindung erforderlich. Vielmehr wird hier nur die Wiederholung des aufgerufenen Dienstes beim Greifer veranlaßt und die entsprechende RS-Variable im Zustandsgraphen des Koordinators auf den richtigen Wert gesetzt (RS = 7).

Konnte der Dienst trotz mehrmaliger Wiederholung nicht richtig ausgeführt werden, weil es beispielsweise in der Greifvorrichtung tatsächlich zu einem Defekt in der Aktorik gekommen ist, muß eine andere Reaktion eingeleitet werden. Die Entscheidung dafür kann manuell oder nach einer definierten Anzahl von Wiederholungen automatisch erfolgen. Soll der Dienst nicht weiter wiederholt werden, steht im Zustandsgraphen des Koordinators ein Rückwärtspfad für den komplexen Dienst *wechseln* zur Verfügung, der das gesamte Werkzeugwechselaggregat in den Ausgangszustand zurückfährt (Abbildung 7-11).

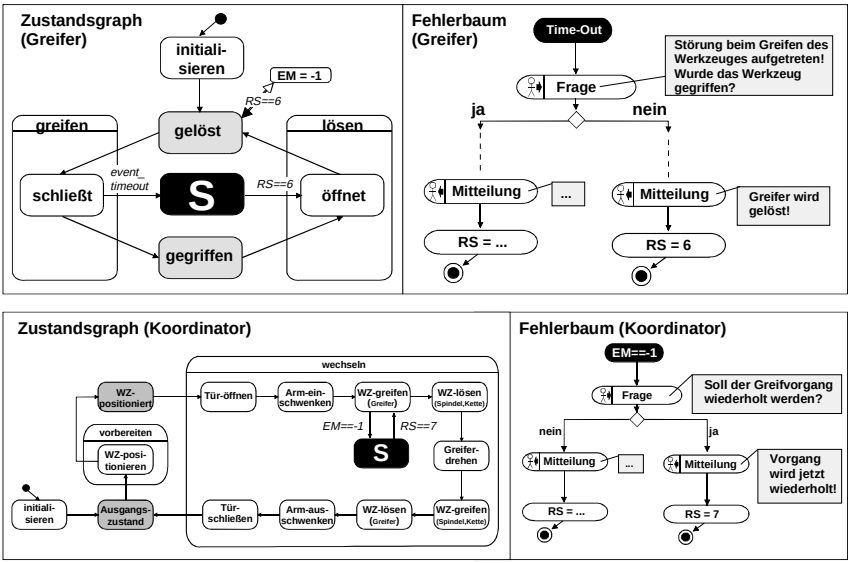


Abbildung 7-10: Störungsbehandlungseinträge in den Zustandsgraphen und in die Fehlerbäume der Steuerungsbausteine „Greifer und Koordinator“

Wird im Fehlerbaum des Koordinators RS = 9 gesetzt, führt dies zur Abarbeitung des Rückwärtspfads bis zum Ausgangszustand des Dienstes. In diesem Zustand kann der Bediener das Werkzeug aus dem Magazin entnehmen. Des weiteren ist dieser Ausgangszustand ein Zustand, in dem das Bearbeitungszentrum komplett angehalten und heruntergefahren werden kann, um hier einen manuellen Reparatureingriff am Greifer vorzunehmen. Beim Wiederhochfahren des Bearbeitungszentrums sind dann keine komplexen Einfahrvorgänge mehr notwendig, da es in einem zugelassenen definierten Zustand angehalten worden ist.

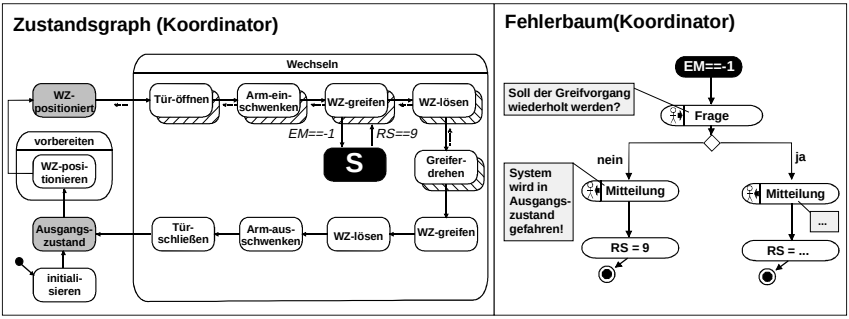


Abbildung 7-11: Rückwärtspfad im Zustandsgraphen des Koordinators

8 Bewertung

Mit der vorliegenden Arbeit wird eine Methode zur produktiven Entwicklung störungstoleranter Steuerungen vorgestellt. Hierfür wurde in Kapitel 5 ein Referenzkonzept entwickelt, das als Gestaltungsrichtlinie für störungstolerante Steuerungen dienen soll. Zudem wurde in Kapitel 6 ein methodisches Vorgehen zur Entwicklung störungstoleranter Anwendungen beschrieben. In diesem Kapitel soll nun untersucht werden, inwiefern diese Ergebnisse tatsächlich die Produktivität des Entwicklungsprozesses erhöhen und einer wirtschaftlichen Softwareentwicklung dienen.

Nach *Balzert (1998, S. 8 ff.)* kann Produktivitätssteigerung im Zusammenhang mit Softwareentwicklung folgendes bedeuten:

- Softwareprodukte schneller, d.h. in kürzerer Kalenderzeit, zu entwickeln,
- Softwareprodukte mit einem höheren Return of Invest zu entwickeln, d.h. mit niedrigeren Kosten, oder
- Software mir besserer Qualität zu entwickeln, wobei sich die Qualität nicht nur auf die Funktionalität und die Zuverlässigkeit beschränkt, sondern auch Aspekte wie Effizienz, Änderbarkeit und Übertragbarkeit beinhaltet.

Für die Entwicklung störungstoleranter Steuerungssoftware bedeutet eine Produktivitätssteigerung auch, daß der Mehraufwand zur Integration von Störungsbehandlungsfunktionen in Steuerungsabläufen durch das effiziente Vorgehen besser kompensiert werden kann.

Zur allgemeinen Steigerung der Produktivität eines Softwareentwicklungsprozesses können unterschiedliche Faktoren beitragen. *Balzert (1998, S. 12)* teilt diese in folgende Gruppen auf:

- Produkteinflüsse
- Prozeßeinflüsse
- Mitarbeitereinflüsse
- Managementeinflüsse

Während Aspekte wie die Produktkomplexität und die Produktgröße zu den Produkteinflüssen gehören, zählen Faktoren wie Methodeneinsatz, CASE-Einsatz und Wiederverwendbarkeit zu den Prozeßfaktoren, die auf die Entwicklungsproduktivität Einfluß haben. Mitarbeitereinflüsse sind Punkte wie Anwendungserfahrung und Spracherfahrung. Zu den Managementeinflüssen zählen Aspekte wie die Teamgröße und die Ausstattung der Arbeitsumgebung.

In Abbildung 8-1 sind nach *Balzert (1998)* die wesentlichen Einflußfaktoren auf die Produktivität der Softwareentwicklung aufgezeigt. Auf Basis zahlreicher Studien von

Boehm (1987), DeMarco & Lister (1987), Grady (1992) und *Maxwell u.a. (1996)* werden einigen dieser Faktoren Verhältniszahlen zugeordnet, die eine quantitative Angabe über die mögliche Produktivitätssteigerung geben. So bedeutet beispielsweise das Verhältnis 1:1,65 im Zusammenhang mit dem Faktor "CASE-Tools", daß sich durch den Einsatz von CASE die Produktivität bis um das 1,65-fache verbessern kann. Auch wenn für einen konkreten Softwareentwicklungsvorgang der tatsächliche Erfüllungsgrad der verschiedenen Einflußfaktoren nicht immer anhand faßbarer Zahlen quantifiziert werden kann, geben die Verhältniszahlen aus Abbildung 8-1 Aufschluß darüber, welche Faktoren mehr und welche weniger die Produktivität steigern.

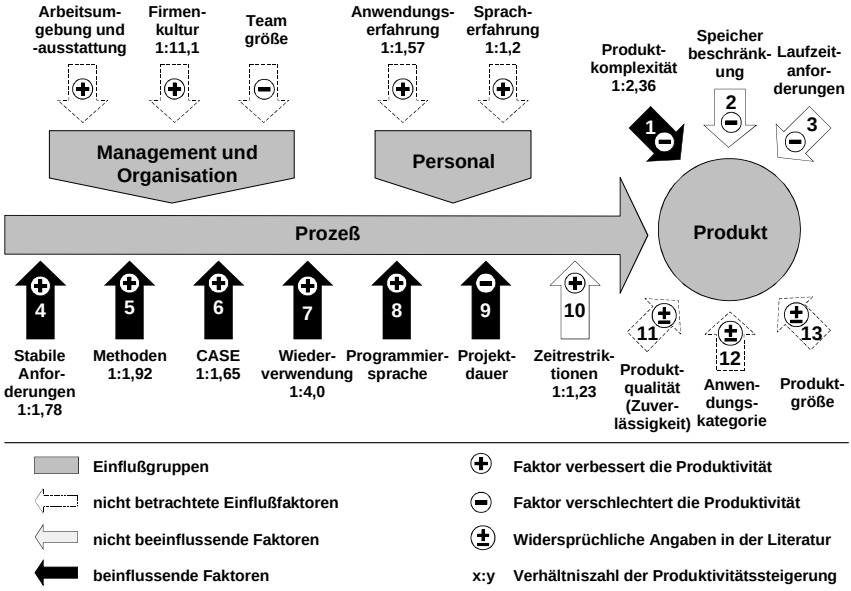


Abbildung 8-1: Einflußfaktoren der Produktivität nach (Balzert 1998, S 13)

Zur Bewertung der hier erarbeiteten Methode zur Entwicklung störungstoleranter Steuerungen wird im folgenden untersucht, in wie fern die verschiedenen Faktoren zur Produktivitätssteigerung beeinflußt werden. Da die erarbeitete Methode in erster Linie das Softwareprodukt und den Softwareprozeß betrifft, beschränken sich die Untersuchungen auf Produkt- und Prozeßeinflußfaktoren (Abbildung 8-1/ Faktoren 1 bis 13). Da jedoch in der Literatur bei einigen Produktfaktoren widersprüchliche Angaben hinsichtlich einer möglichen Produktivitätssteigerung oder –minderung vorliegen (Abbildung 8-1/ Faktoren 11, 12 und 13), beschränken sich die folgenden

Untersuchungen auf jene Faktoren, zu denen es eine eindeutige Aussage gibt (Abbildung 8-1/ Faktoren 1 bis 10):

Produkteinflüsse

- **Produktkomplexität** (Abbildung 8-1/ Faktor 1): Nach Balzert steigt die Produktivität des Entwicklungsprozesses mit abnehmender Produktkomplexität. Die hier entwickelte Methode enthält zahlreiche Aspekte zur Reduzierung dieser. So ist ein bewährter Ansatz zur Reduzierung der Komplexität die Modularisierung des Softwaresystems, d.h. die strukturelle Gliederung der Software in autarke Bausteine, die über klare Schnittstellen externe Bezüge definieren. Das in Kapitel 5 vorgestellte Referenzkonzept, das als Gestaltungsrichtlinie für störungstolerante Steuerungen dient, ist ein modulares Konzept mit klarer Schnittstellendefinition zwischen den Modulen. Hier wird die Steuerungssoftware in einzelne Bausteine aufgeteilt, zwischen denen Client/Server-Beziehungen herrschen. Ein weiterer Ansatz zur Reduzierung der Komplexität ist die Erhöhung der Transparenz. Dazu werden im Referenzkonzept die Steuerungsbausteine in Steuerungsbereiche und Störungsbehandlungsbereiche mit klaren und transparenten Softwarestrukturen gegliedert, die leicht mittels graphischer Beschreibungstechniken beschrieben werden können. Außerdem wird bei der hier erarbeiteten Methode die Störungsbehandlungsfunktionalität von Anfang an berücksichtigt. Nachträgliche Erweiterungen, die meist zur Erhöhung der Komplexität führen, werden hierdurch vermieden. Zudem setzt das Referenzkonzept zur Störungsbehandlung nur einfache Algorithmen ein und verzichtet auf komplexe Verfahren wie Inferenzverfahren oder Fuzzy-Regeln. All diese Punkte tragen wesentlich dazu bei, die Komplexität störungstoleranter Steuerungen wesentlich zu reduzieren, was wiederum die Entwicklungsproduktivität erhöht.
- **Speicherbeschränkung** (Abbildung 8-1/ Faktor 2): Da die Speicherbeschränkung implementierungsabhängig ist und da das Referenzkonzept eine Gestaltungsrichtlinie auf Entwurfsebene ist, hat die entwickelte Methode keinen Einfluß auf diesen Faktor.
- **Laufzeitanforderungen** (Abbildung 8-1/ Faktor 3): Laut *Maxwell u.a. (1996)* sinkt mit zunehmender Laufzeitanforderung die Produktivität des Entwicklungsprozesses. Die Laufzeitanforderungen, die an eine störungstolerante Steuerung gestellt werden, sind zu Beginn der Arbeit in Kapitel 3.2 beschrieben. Hier sind insbesondere die schnellen Reaktionszeiten bei der Störungserkennung von großer Bedeutung. Mit dem hier definierten Referenzkonzept werden diese Anforderungen nicht verändert. Somit hat die erarbeitete Methode keinen Einfluß auf die Laufzeitanforderungen.

Prozeßeinflüsse

- **Stabile Anforderungen** (Abbildung 8-1/ Faktor 4)**:** Eine unzureichende frühzeitige Spezifikation von Softwareanforderungen provoziert Softwarefehler und verstärkt die Notwendigkeit, nachträglich Änderungen in der Software vornehmen zu müssen. Nach *Broy & Stauner (1999)* zeigen die Statistiken, daß über 50% der auftretenden Probleme (Fehler) in heutigen ausgelieferten Softwaresystemen auf eine unzureichende Anforderungsspezifikation zurückzuführen sind. Dies wirkt sich einerseits negativ auf die Effektivität der Softwareentwicklung aus. Andererseits wird dadurch auch die Komplexität der Software wesentlich erhöht. Im Vergleich zum gegenwärtigen, in der Steuerungstechnik gängigen Vorgehen, die Störungsbehandlungsfunktionalität erst nachträglich in die Steuerungsapplikation mit einzubinden, fordert das im Rahmen dieser Arbeit erstellte methodische Vorgehen zur Entwicklung störungstoleranter Steuerungen die frühzeitige Einbeziehung der Entwicklung der Störungsbehandlungsfunktionalität in den gesamten Entwicklungsprozeß. Somit liegt nicht nur eine detaillierte Spezifikation der Anforderungen an die Steuerungsfunktionalität vor, sondern auch eine genaue Beschreibung des geforderten Störungsbehandlungsverhaltens. Die frühzeitige Spezifikation der Anforderungen führt zu stabilen Anforderungen und erhöht somit die Produktivität des Entwicklungsprozesses.
- **Einsatz von Methoden (Beschreibungstechniken) und CASE-Tools** (Abbildung 8-1/ Faktoren 5 und 6): Durch den langfristigen Einsatz von Softwaremethoden und CASE-Werkzeugen kann die Produktivität des Entwicklungsprozesses wesentlich erhöht werden *(Balzert 1998, S14 ff.)*. Die hier erarbeitete Methode zur Entwicklung störungstoleranter Steuerungen ist auf den verstärkten Einsatz methodischer Beschreibungstechniken, wie z.B. Zustandsgraphen und Entscheidungstabellen, ausgerichtet. Des weiteren fordert die erarbeitete Methode den Einsatz leistungsfähiger graphischer Entwicklungswerkzeuge. Hierdurch werden die Anforderungen wesentlich strukturierter spezifiziert und die Funktionalität bzw. der Aufbau der Steuerungssoftware klarer beschrieben.
- **Wiederverwendbarkeit** (Abbildung 8-1/ Faktor 7)**:** Durch die Wiederverwendung bestehender Ergebnisse kann bekanntlich die Produktivität des Entwicklungsvorgangs wesentlich erhöht und die Qualität verbessert werden. Solche wiederverwendbaren Ergebnisse können im allgemeinen Projekterfahrungen, bewährte Softwarearchitekturen, aber auch fertige Softwarebausteine sein. In der hier entwickelten Methode wird die Wiederverwendbarkeit bestehender Ergebnisse in vielerlei Hinsicht gefördert. So dient z.B. das erarbeitete methodische Vorgehen als wiederverwendbarer Leitfaden zur effizienten Softwareentwicklung. Der wesentliche Wiederverwendungseffekt entsteht jedoch durch die modulare Softwarestruktur und die Definition der sogenannte Steuerungsklassen, die durch einfache Instantiierung in unterschiedlichen Anwendung erneut verwendet werden können.

- **Höhere Programmiersprachen** (Abbildung 8-1/ Faktor 8): Nach *Balzert (1998, S. 13)* hat der Einsatz höherer Programmiersprachen einen positiven Einfluß auf die Produktivität des Softwareentwicklungsprozesses. In der hier erarbeiteten Methode steht grundsätzlich offen, welche Programmiersprache zur Implementierung von störungstoleranten Steuerungsanwendungen eingesetzt wird. Dennoch wird der Einsatz höherer Programmiersprachen durch einige Punkte unterstützt. So ist einerseits das beschriebene Softwaremodell ein objektorientiertes Modell, was den Einsatz höherer OO-Programmiersprachen fördert, vorausgesetzt die entsprechende Steuerungsplattform unterstützt eine derartige Programmiersprache. Andererseits wird mit der Bereitstellung von Werkzeugen zur translativen Implementierung von störungstoleranten Steuerungen eine Entwicklungsumgebung bereitgestellt, die als Programmiersprache die Beschreibungstechnik des Softwaremodells verwendet.
- **Projektdauer** (Abbildung 8-1/ Faktor 9): Mit zunehmender Projektdauer sinkt die Produktivität des Entwicklungsprozesses. Die hier vorgestellte Methode zur Entwicklung störungstoleranter Steuerungen unterstützt die Reduzierung der Projektdauer und somit die Produktivitätssteigerung in zweierlei Hinsicht: Einerseits wird hier ein modularer Ansatz verfolgt, der eine gleichzeitige Entwicklung mehrerer autarker Softwaremodule mit definierten Schnittstellen ermöglicht. Andererseits wird hier die parallele Entwicklung von Steuerungs- und Störungsbehandlungsfunktionalität propagiert. Eine zur Ergänzung der Störungsbehandlungsfunktionalität nachträgliche Einarbeitung in die bereits erstellte Steuerungsfunktionalität, wie dies heutzutage der Regelfall ist, wird hier vermieden.
- **Zeitrestriktionen** (Abbildung 8-1/ Faktor 10). Nach *Balzert (1998, S. 17)* wirken sich sehr enge Termine negativ auf die Produktivität des Entwicklungsprozesses aus. Mit der hier vorgestellten Methode werden keine Zeitrestriktionen gemacht. Das methodische Vorgehen beschreibt lediglich die einzelnen Phasen der Entwicklung, die zu durchlaufen sind, ohne auf die eigentlich benötigte Zeit weiter einzugehen. Somit hat die hier erarbeitete Methode keine Auswirkung auf diesen Einflußfaktor.

Bei einer qualitativen Bewertung der hier erarbeiteten Methode im Zusammenhang mit den einzelnen produktivitätssteigernden Einflußfaktoren kann festgehalten werden, daß die Methode eine positiven Einfluß auf alle relevanten Faktoren hat. Dies gilt insbesondere für jene Faktoren, die die Produktivität im hohen Maße steigern, wie die Wiederverwendbarkeit, die Komplexitätsreduktion oder die Definition stabiler Anforderungen (siehe die Verhältniszahlen der Produktivitätssteigerung in Abbildung 8-1).

Zur quantitativen Bewertung dieser Produktivitätssteigerung soll hier ein einfaches Rechenbeispiel aufgezeigt werden: So nimmt nach *Wagner (1997, S. 5)* der

Störungsbehandlungsanteil heute bis zu 30% des Softwareumfangs ein. Vor diesem Hintergrund soll die hier entwickelte Methode als ergiebig bewertet werden, wenn durch die Anwendung dieser eine Kostensenkung von mindestes 30% pro entwickelter Softwareeinheit erzielt werden kann.

Somit sollte

$$\frac{K_{SWE-n}}{K_{SWE-o}} \leq 0.7$$

K_{SWE-n} = Kosten pro Softwareeinheit mit neuer Methode
K_{SWE-o} = Kosten pro Softwareeinheit ohne neue Methode

(Gl. 8-1)

sein.

Für das Rechenbeispiel wird als Softwareeinheit jene Arbeitsleistung betrachtet, die herkömmlicherweise in einer Stunde Softwareentwicklung geleistet werden kann. Bei einem angenommenen Stundensatz eines durchschnittlichen Softwareentwicklers von 80 DM/h liegt somit K_{SWE-o} bei DM 80,- pro Softwareeinheit.

Bei der Anwendung der im Rahmen der Arbeit entwickelten Methode wird einerseits eine Produktivitätssteigerung erzielt, gleichzeitig fallen aber Investitionskosten an, die in der Berechnung berücksichtigt werden müssen. Zur Berechnung der Produktivitätssteigerung wird hier zunächst der Durchschnittswert der Verhältniszahlen aller beeinflußten Faktoren gebildet. Dabei können natürlich nur jene Faktoren herangezogen werden, die Verhältniszahlen aufweisen. Im Einzelnen sind dies die Faktoren „Produktkomplexität", „stabile Anforderungen", „Methoden", „CASE-Tools" und Wiederverwendung" (Siehe Abbildung 8-1/ Faktoren 1, 4, 5, 6 und 7). Für diese Faktoren soll im betrachteten Beispiel eine nur 50 %-ige Erfüllung angenommen werden. Dadurch ergeben sich folgende Berechnungen:

$$\alpha_{100\%} = \frac{2{,}36 + 1{,}78 + 1{,}92 + 1{,}65 + 4{,}0}{5} = 2{,}34$$

$\alpha_{100\%}$ = ∅ Verhältniszahl der Produktivitätssteigerung bei 100%-iger Erfüllung aller Faktoren

(Gl. 8-2)

$$\alpha_{50\%} = \alpha_{100\%} - \frac{\alpha_{100\%} - 1}{2} = 1{,}67$$

$\alpha_{50\%}$ = ∅ Verhältniszahl der Produktivitätssteigerung bei 50%-iger Erfüllung aller Faktoren

(Gl. 8-3)

Zur Berechnung der anfallenden Investitionskosten werden Schulungskosten in Höhe von ca. DM 8.000,- pro Jahr sowie Kosten für CASE-Tools und Entwicklungslizenzen in Hohe von ca. DM 15.000,- pro Jahr angenommen. Zur Berechnung der Entwicklungskosten pro Softwareeinheit soll sich der Abschreibungszeitraum auf ein Jahr beschränken. In diesem Zeitraum kann ein Softwareentwickler eine Arbeitsleistung von 1760* Stunden erbringen. Durch die erzielbare Produktivitäts-

* 220 reguläre Arbeitstage pro Jahr a 8 Stunden

steigerung können jedoch mehr als 1760 Softwareeinheiten in einem Jahr entwickelt werden. Zur Berechnung der neuen Kosten pro Softwareeinheit ergibt sich folgende Gleichung:

$$K_{SWE-n} = \frac{Lohnkosten_{pro\ Jahr} + Investkosten_{pro\ Jahr}}{\alpha_{50\%} * Arbeitsleitung_{pro\ Jahr}} \qquad \textit{(Gl. 8-4)}$$

Unter Verwendung der obigen Zahlen liegen die Kosten pro Softwareeinheit bei:

$$K_{SWE-n} = \frac{80 * 1760 + (8000 + 15000)}{1{,}67 * 1760} = 55{,}7 \ \left[DM / Einheit\right] \qquad \textit{(Gl. 8-5)}$$

Bei einem Vergleich beider Kostenzahlen pro Softwareeinheit entsteht ein Verhältnis (K_{SWE-n} / K_{SWE-o}) von 0,69, das knapp unter 0,7 liegt. Daraus wird abgeleitet, daß der benötigte Aufwand zur Entwicklung störungstoleranter Steuerungen mit Hilfe der hier entwickelten Methode gut kompensiert werden kann.

9 Zusammenfassung und Ausblick

Der verstärkte Wettbewerb der letzten Jahre hat - bedingt durch die Zunahme potentieller Anbieter bei gleichzeitiger Sättigung der Märkte - dazu beigetragen, daß die flexible Automatisierung in der Produktion immer mehr an Bedeutung gewinnt. Auf diese Entwicklung werden vermehrt hochflexible Produktionssysteme eingesetzt. Diese Produktionssysteme sind jedoch oftmals durch eine unzureichende technische Verfügbarkeit gekennzeichnet, die primär auf die Komplexität dieser Systeme und ihre fehlende Störungstoleranz zurückzuführen ist.

Zur Erhöhung der technischen Verfügbarkeit in Produktionssystemen gibt es bereits zahlreiche steuerungstechnische Ansätze, die sich mit einer rechnergestützten Störungsbehandlung beschäftigen. Da die meisten technisch bedingten Stillstände auf Störungen in der Maschinen- und Anlagenperipherie zurückzuführen sind, zeichnet sich der maschinennahe Bereich durch ein besonderes Potential für störungsbehandelnde Steuerungen aus.

Zur rechnergeführten Störungsbehandlung existieren bereits unterschiedliche Ansätze. Die meisten eingesetzten Systeme unterstützen jedoch nur die Störungserkennung und Störungslokalisierung. Nur selten erfolgt hier eine umfassende Störungsbehandlung, die auch eine Störungsbehebung einschließlich des Wiederanlaufs des normalen Betriebs beinhaltet. Gerade hier liegt aber ein nicht unwesentliches Einsparungspotential von technisch bedingten Stillstandszeiten. Zwar sind die heutigen Steuerungen ausreichend leistungsfähig, um zusätzliche Funktionen zur Störungsbehebung samt Wiederanlauf zu unterstützen. Dies wird auch durch vereinzelte Steuerungsansätze aus der Forschung bestätigt. Dennoch werden derartige Ansätze industriell nur selten eingesetzt, da sie sich für den Maschinenhersteller wirtschaftlich kaum lohnen. So sind einerseits die bestehenden Steuerungsansätze viel zu komplex. Andererseits fehlen Methoden und Werkzeuge, die die aufwendige Entwicklung vereinfachen und effektiver gestalten.

Ziel dieser Arbeit war es deshalb, eine werkzeugunterstützte Methode zur effizienten Entwicklung von störungstoleranten Steuerungen, die alle Phasen der Störungsbehandlung unterstützt, zu erarbeiten. Diese Methode sollte zum einen ein vereinfachtes funktionales Referenzkonzept enthalten, das als Gestaltungsrichtlinie für störungstolerante Steuerungen dient. Andererseits sollte die Methode eine werkzeugunterstützte Vorgehensweise zur effizienten Entwicklung von störungstoleranten Steuerungsanwendungen beschreiben.

Um dieses Ziel zu erreichen, wurde im Rahmen dieser Arbeit zunächst eine detaillierte Analyse der Anforderungen an die hier zu erarbeitende Entwicklungsmethode durchgeführt. Dazu wurden die Anforderungen an das Referenzkonzept und an das methodische Vorgehen einzeln untersucht. Basierend auf den Anforderungen an das Referenzkonzept wurden anschließend zahlreiche

bestehende Ansätze zur rechnergeführten Störungsbehandlung hinsichtlich ihrer Eignung als Referenzkonzept analysiert und bewertet.

Aufgrund der Defizite bestehender Ansätze wurde hier ein neues, vereinfachtes Steuerungskonzept vorgestellt, das als Referenzkonzept für störungstolerante Steuerungen dient. Dieses Referenzkonzept berücksichtigt alle Phasen der Störungsbehandlung und unterstützt den Bediener des Produktionssystems bei der Bewältigung von Störungen auf Maschinenebene. Dabei beschränkt sich diese Unterstützung nicht nur auf die Bereitstellung einfacher Störungsmeldungen, sondern sieht vielmehr die aktive Führung des Bedieners und die Nutzung seiner flexiblen Fertigkeit bei der kompletten Störungsbehandlung vor. Dieses modular aufgebaute Referezkonzept kann vollständig anhand zustandsorientierter Beschreibungstechniken spezifiziert werden und verzichtet im Zusammenhang mit der Störungsbehandlungsfunktionalität auf die Verwendung komplexer Algorithmen der künstlichen Intelligenz, wie dies in vielen bestehenden Störungsbehandlungsansätzen der Fall ist.

Des weiteren wurde in dieser Arbeit ein methodisches Vorgehen zur Entwicklung störungstoleranter Steuerungsanwendungen, die auf dem Referenzkonzept basieren, beschrieben. Dieses methodische Vorgehen enthält unter anderem eine Beschreibungstechnik, die auf der UML basiert und für alle Phasen der Entwicklung verwendet werden kann. Zusätzlich beinhaltet das methodische Vorgehen eine detaillierte Vorgehensweise, die aufzeigt, wie die Beschreibungstechnik in verschiedenen Phasen und Schritten zur systematischen Entwicklung störungstoleranter Anwendungen verwendet werden kann. Wesentlich an dieser Vorgehensweise ist die parallele Entwicklung der Steuerungs- und Störungsbehandlungsfunktionalität, die nicht nur eine detaillierte Spezifikation der Anforderungen an die Steuerungsfunktionalität, sondern auch eine genaue Beschreibung des gewünschten Störungsbehandlungsverhaltens erfordert. Nachträgliche Softwareergänzungen, die die Softwarekomplexität weiter erhöhen, werden dadurch ausgeschlossen.

Zur Implementierung störungstoleranter Steuerungsanwendungen wurde hier außerdem ein Werkzeug entwickelt, das auf Basis von graphischen Beschreibungstechniken die einfache Programmierung von störungstoleranten Maschinensteuerungen ermöglicht. Am Beispiel eines Werkzeugwechselaggregats eines Bearbeitungszentrums wurden die hier erarbeiteten Ergebnisse validiert.

In der vorliegenden Arbeit wurden alle für die störungstoleranten Steuerungsanwendungen erforderlichen Daten durch den methodisch vorgehenden Entwickler erfaßt und spezifiziert. In zukünftigen Arbeiten müßte untersucht werden, inwiefern CASE-Tools und Informationssysteme, die detaillierte Mechanikmodelle von Produktionssysteme enthalten, zur automatischen Generierung von Steuerungs- und Störungsbehandlungssoftware herangezogen werden können. Ein derartiges Vorgehen würde einerseits die Entwicklung störungstoleranter Steuerungen weiter

vereinfachen. Andererseits könnte dadurch eine stärkere Integration der Entwicklung von Mechanik, Steuerungssoftware und Störungsbehandlungssoftware erfolgen, was letztendlich die Gesamtentwicklung von störungstoleranten Produktionssystemen verkürzen würde.

10 Literatur

Abel 1990

Abel, D.: Petri-Netze für Ingenieure – Modellbildung und Analyse diskret gesteuerter Systeme. Berlin: Springer 1990.

Anders 1998

Anders, C.: Adaptierbares Diagnosesystem bei Trasferstraßen. Berlin: Springer 1998 (isw Forschung und Praxis 126).

Awad u. a. 1996

Awad, M; Kuusela, J.; Ziegler, J.: Object Oriented Technologie for Real Time Systems – A Practical Approach Using OMT and Fusion. Upper Saddle River, Prentice Hall 1996.

Balzert 1996

Balzert, H.: Lehrbuch der Software-Technik – Software-Entwicklung. Heidelberg: Spektrum 1996.

Balzert 1998

Balzert, H.: Lehrbuch der Software-Technik – Software – Management, Software – Qualitätssicherung, Unternehmens-modellierung. Heidelberg: Spektrum 1998.

Barth u. a. 1988

Barth, W. u. a.: Steuerungen in der Automatisierungstechnik. Berlin: VBE Verlag Technik 1988.

Birkel 1995

Birkel, G.: Aufwandsminimierter Wissenserwerb für die Diagnose in flexiblen Produktionszellen. Berlin: Springer 1996. (iwb Forschungs-berichte 84)

Boehm 1987

Boehm, B. W.: Improving Software Productivity. In: IEEE Computer. (1997) 9, S. 43–57.

Booch 1994

Booch, G.: Object oriented analysis and design with applications, 2.nd ed. Redwood City: Benjamin/ Cummings 1994.

Booch & Rumbaugh 1995

Booch, G.; Rumbaugh, J.: Unified Methode – Version 0.8. Santa Clara: Rational Software Cooperation 1995.

Booch u. a. 1997

Booch, G.; Rumbaugh, J.; Jacobsen, I.: Unified Modeling Language User Guide. Reading, Massachusetts: Addison-Wesley Longman 1998.

Braun 1999

Braun, W.: Speicherprogrammierbare Steuerungen in der Praxis. Programmiersprachen von STEP7. Vieweg Verlag 1999.

Broy & Schmidt 1999

Broy, M.; Schmidt, J. W.: Zur Diskussion gestellt – Informatik: Grundlagenwissenschaft oder Ingenieurdisziplin. Informatik-Spektrum 22 (1999), S. 206 – 209.

Broy & Stauner 1999

Broy, M.; Stauner, T.: Requirements Engineering für eingebettete Systeme. Informationstechnik und Technische Informatik (it+ti) (1999) 2, S. 7 – 11.

DeMarco & Lister 1987

DeMarco, T.; Lister T.: Peopleware. New York: Dorset Haus Publishing Co. 1987.

DIN 19226 Teil 5 1994

DIN 19226, Teil 5: Leittechnik; Regelungstechnik und Steuerungstechnik – Funktionelle Begriffe. Berlin: Beuth Verlag 1994.

DIN 19237 1980

DIN 19237: Messen, Steuern, Regeln; Steuerungstechnik, Begriffe. Berlin: Beuth Verlag 1980.

DIN 25424 Teil 1 1981

DIN 25424, Teil 1: Fehlerbaumanalyse: Methoden und Bildzeichen. Berlin: Beuth-Verlag 1981.

DIN 25424 Teil 2 1990

DIN 25424, Teil 2: Fehlerbaumanalyse: Handrechenverfahren zur Auswertung eines Fehlerbaumes: Beuth-Verlag 1990.

DIN 31051 1985

DIN 31051: Instandhaltung – Begriffe und Maßnahmen. Berlin: Beuth-Verlag 1985.

DIN 40041 1990

DIN 40041: Zuverlässigkeit – Begriffe. Berlin: Beuth-Verlag 1990.

DIN 55350 Teil 11 1995

DIN 55350, Teil 11: Begriffe des Qualitätsmanagements und Statistik – Begriffe des Qualitätsmanagements. Berlin: Beuth-Verlag 1987.

DIN 66001 1983

DIN 66001: Sinnbilder und ihre Anwendung. Berlin: Beuth-Verlag 1983.

Diesch u. a. 1997

Diesch, R.; Graf, U.; Valkyser, B.; Weiner, M.: Organisationsmodule für offene Steuerungen. ZWF CIM 92 9/97. München: Carl Hanser Verlag 1997.

Diehl 1992

Diehl, G.: Steuerungsperipheres Diagnosesystem für Fertigungseinrichtungen auf Basis überwachungs-gerechter Komponenten. Berlin: Springer 1992. (isw Forschung und Praxis 89)

Douglass 1998

Douglass, B. P.: Real-Time UML - Developing Efficient Objets for Embedded Systems. Reading, Massachusetts: Addison-Wesley 1998.

Ehrlenspiel 1988

Ehrlenspiel, K.: Umdruck zur Vorlesung; Konstruktionslehre I. Grundlagen und Methodenbaukasten zum funktionsgerechten Konstruieren. TU-München: 1988.

Eißler 1983

Eißler, W.: Automatisierte Überwachungsverfahren für Fertigungseinrichtungen mit speicherprogrammierbaren Steuerungen. Berlin: Springer 1983. (IPA Forschung und Praxis 68)

Enderle 1989

Enderle, W.: Verfügbarkeitssteigerung automatisierter Montagesysteme durch selbständige Behebung prozeßbedingter Störungen. Karlsruhe: Schnelldruck Ernst Grässer 1989.

Fähnrich 1990

Fähnrich K.-P.: Ein system zur wissensbasierten Diagnose an CNC-Werkzeugmaschinen durch den Maschinenbediener Berlin: Springer 1990. (IPA Forschungs und Praxis 148)

Fleckenstein 1987

Fleckenstein, J.: Zustandsgraphen für SPS – Graphikunterstützte Programmierung und steuerungsunabhängige Darstellung. Berlin: Springer 1987. (isw Forschung und Praxis 63)

Freyermuth 1993

Freyermuth, B.: Wissensbasierte Fehlerdiagnose am Beispiel eines Industrieroboters. TH-Darmstadt: VDI Verlag 1993 (VDI Forschungsberichte, Reihe 8: Meß-, Steuerungs- und Regelungstechnik Nr. 315).

Frick 1995

Frick, A.: Der Software-Entwicklungsprozeß – Ganzheitliche Sicht. Grundlagen zu Entwicklungs-Prozeß-Modellen. München: Carl Hanser Verlag 1995.

Gamma u. a.1995

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.; Design Patterns – Elements of Reusable Object-Oriented Software. Reading Massachusetts: Addison Wesley 1995.

Grady 1992

Grady, B. R.: Practical Software Metrics for Project Management and Process Improvement. Englewood Cliffs: Prentice Hall 1992.

Grimm 1987

Grimm, W.: Diagnosesysteme für steuerungsperiphere Fehler an Fertigungseinrichtungen. Berlin: Springer 1987.

Grötsch & Seubert 1997

Grötsch, E.; Seubert, L.: SPS II – Speicherprogrammierbare Steuerungen. Bd. II, Programmbeispiele und Produkte. München: Oldenbourg Verlag 1997.

Glas 1993

Glas, H.: Standardisierter Aufbau anwendungsspezifischer Zellenrechnersoftware. Berlin: Springer 1993. (iwb Forschungsberichte 61)

Harel 1987

Harel, D.: State Charts: A Visual Formalism for Complex Systems. In: Science of Computer Programming (1987) 8, S. 231 ff.

Härdtner 1992

Härdtner, G. M.: Wissensstrukturierung in Expertensystemen für Fertigungseinrichtungen. Berlin: Springer 1992. (isw Forschung und Praxis 93)

He 1993

He, B. E. X: Modellgestützte Fehlererkennung mittels Parameterschätzung zur wissensbasierten Fehlerdiagnose an einem Vorschubantrieb. TH Darmstadt: VDI Verlag 1993. (VDI Forschungsberichte, Reihe 8: Meß-, Steuerungs- und Regelungstechnik Nr. 354)

Helml 1992

Helml, H. J.: Ein Verfahren zur on-line Fehlererkennung und Diagnose. Berlin: Springer Verlag 1992. (iwb Forschungsberichte 53)

Hofmann 1990

Hofmann, P.: Fehlerbehandlung in flexiblen Fertigungssystemen – Einführung für Hersteller und Anwender. München: Oldenbourg 1990.

Hopcraft & Ullman 1979

Hopcraft, J. E.; Ullman, J. D.: Introduction to automata theory, languages and computiation. Reading: Addison-Wesley 1979.

IEC 1131 Teil 3 1994

IEC 1131, Teil 3: Standard for programmable Controllers. Part 3: Programming Languages. 1994.

Isermann 1994

Isermann, R. (Hrsg.): Überwachung und Fehlerdiagnose. Moderne Methoden und ihre Anwendung bei technischen Systemen. Düsseldorf: VDI-Verlag 1994.

ISO TC 184 1986

ISO TC 184/SC5/WG1 Document N51 (Version 1.1): The Ottawa Report on Reference Models for Manufacturing Standards. Genf: 1986.

Jacobsen 1992

Jacobsen, I.; Christerson, M.; Jonsson, P.; Övergaard, G.: Object Oriented Software Engineering. A Use Case Driven Approach. Workingham: Addison Wesley 1992.

John & Tiegelkamp 1995

John, K. H.; Tiegelkamp, M.: SPS-Programmierung mit IEC 1131-3. Berlin: Springer 1995.

Kallfass 1999

Kallfass, H.: Verfügbarkeit in der mechanischen Fertigung. In: Seminarbericht (1999) 43: Produktivität und Verfügbarkeit. Hrsg. G. Reinhart /J. Milberg. München: Herbert Utz Verlag Wissenschaft 1999.

Kief 1997

Kief, H. B.: NC/CNC Handbuch '97/98. München: Carl Hanser Verlag 1998, S. 405 ff.

Kiratli 1989

Kiratli, G.: Konzept und Realisierung eines wissensbasierten Systems zur Diagnose und Bedienerunterstützung bei komplexen Fertigungseinrichtungen. Aachen: Trans-Aix-Press 1989.

Klinker 1995

Klinker, W.: atp – Marktanalyse „SPS-basierte Leitsysteme“, Teil 2: Tabellarische Übersicht. atp – Automatisierungstechnik Praxis. München: Oldenbourg Verlag 37 (1995) 12, S. 20–30.

Koch 1996

Koch, M. R.: Autonome Fertigungszellen – Gestaltung, Steuerung und integrierte Störungsbehandlung. Berlin: Springer 1996. (iwb Forschungsberichte 98)

Koch & Köhne 1995

Koch, M. R.; Köhne, T.: Autonomie – Leitbild mit Perspektive. ZWF Zeitschrift für Wirtschaftlichen Betrieb 90 (1995) 4, S. 165–167.

König & Ketteler 1994

König, W.; Ketteler G.: Prozeßüberwachung beim Messerkopfstirnfräsen durch Auswertung von Acoustic Emission-Signals. VDI-Z136 (1994) 1, S. 92–97.

Kühne 1985

Kühne, L.: Entwicklung eines universellen Überwachungs- und Diagnosesystems für Fertigungseinrichtungen: RWTH-Aachen: Fotodruck J. Mainz 1985.

Martin & Odell 1992

Martin, J.; Odell, J.: Object-Oriented Analysis & Design. Englewood Cliffs: Prentice-Hall 1992.

Maxwell u. a. 1996

Maxwell, K. D.; Wassenhove, L. V.; Dutta, S.: Software Development Productivity of European Space, Military, and Industrial Applications. In: IEEE Transactions on Software Engineering. 10/1996, S. 706–718.

Milberg 1997

Milberg, J.: Produktion – Eine Treibende Kraft für unsere Volkswirtschaft. In: Reinhart, G. (Hrsg.); Milberg, J. (Hrsg.): Mit

Schwung zum Aufschwung – Information, Inspiration, Information. Münchener Kolloquium 1997. Landsberg/ Lech: Verlag Moderne Industrie 1997, S. 17–40.

Milberg & Ebner 1994

Milberg, J.; Ebner, K.: Verfügbarkeit von Werkzeugmaschinen. Abschlußbericht der AiF-Forschungsstudie 8649. Frankfurt: VDW, 1994.

Moßig & Stäble 1995

Moßig, K.: Stäble, M.: Steuerungssymthese mit kontrollierten Free-Choice Petri-Netzen zur Prozeßbeschreibung. at – Automatisierungstechnik 43 (1995) 11, S. 506 ff.

Nordmann 1994

Nordmann, K.: Werkzeugüberwachung mit neuen Sensortechniken. VDI-Z Integrierte Produktion, Spezial- Werkzeuge 2 (1994), S. 42–44.

Oesterreich 1997

Oesterreich, B: Objektorientierte Softwareentwicklung mit der Unified Modeling Language. München: Oldenbourg Verlag 1997.

Petri 1962

Petri, C. A.: Kommunikation von Automaten, Universität Bonn 1962. (Schriften des rhein.-westfäl. Instituts für instrumentelle Mathematik an der Universität Bonn, Nr. 2)

Pfeifer & Richter 1993

Pfeifer, T., Richter, M. M.: Diagnose von technischen Systemen – Grundlagen, Methoden und Perspektiven der technischen Diagnose. Wiesbaden: Deutscher Universitätsverlag 1993.

Pfob 1998

Pfob, E.: Intelligente Konfiguration von Transferstraßen. In: Seminarbericht (1998) 37: Wettbewerbsfaktor Verfügbarkeit. Reinhart, G. (Hrsg.); Milberg, J. (Hrsg.). München: Herbert Utz Verlag Wissenschaft 1998, S. 32–43 .

PLCopen 1994

n. n.; Zeitschrift „PLCopening“ der PLCopen. Zaltbommel, Niederlande 1994.

Prtischow 1996

Pritschow, G.: Auf dem Weg zur herstellerübergreifenden „offenen Steuerung“ – eine Bestandsaufnahme. In: Komponenten und

Schnittstellen für offene Steuerungssysteme. München: Carl Hanser Verlag 1996, S. 7–27.

Puppe 1987

Puppe, F.: Diagnostisches Problemlösen mit Expertensystemen, Informatik-Fachberichte Nr. 148; Subreihe Künstliche Intelligenz. München: Springer 1987.

Rambaugh 1991

Rambaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorenson, W.: Object Oriented Modelling and Design. Englewood Cliffs: Prentice-Hall 1991.

Reinhart 1994

Reinhart, G.: Wettbewerbsfähige Produktion – Voraussetzungen für eine strategische Entscheidung. In: Milberg, J. Reinhart, G. (Hrsg).: Unsere Stärken stärken – Der Weg zu Wettbewerbsfähigkeit und Standortsicherung. Münchener Kolloquium 1994. Landsberg/Lech: Verlag Moderne Industrie 1994.

Reinhart u. a. 1996

Reinhart, G.; Lindemann, U.; Heizel, J.: Qualitätsmanagement – Ein Kurs für Studium und Praxis. Berlin: Springer 1996.

Reinhart u. a. 1998

Reinhart, G.; Ansorge, D.; Mauderer, M.; Sabbah, A.: Software in der Produktion. In: ZWF 93 (1998) 6, S. 282–285.

Reinhart u. a. 1999

Reinhart, G.; Meier, H., Sabbah A.: Störungstolerante Steuerungen. In: ZWF 94 (1999) 9, S. 486–489.

Reisig 1985

Reisig, W.: Systementwurf mit Netzen. Berlin: Springer 1985.

Reuschenbach 1992

Reuschenbach, W.: Entwicklung und Einsatz eines universellen Stördatenerfassungssystems mit wissensbasierter Diagnose für Produktionseinrichtungen. RWTH-Aachen: Fotodruck J. Mainz 1992.

Sayegh 1997

Sayegh, A.: CORBA Standard, Spezifikation, Entwicklung. Köln: O'Reilly Verlag 1997.

Schmidt 1991

Schmidt, M.: Konzeption und Einsatzplanung flexibler automatisierter Montagesysteme. Berlin: Springer 1991. (iwb Forschungsberichte 41)

Schneider 1994

Schneider, J.: Fehlerreaktion mit speicherprogrammierbaren Steuerungen – Ein Beitrag zur Fehlertoleranz. Berlin: Springer 1994. (isw Forschung und Praxis 99)

Schnell 1997

Schnell, R.: Induktive Näherungsschalter mit Überwachungsfunktionalität. In: Seminarbericht (1998) 29: Installationstechnik an Werkzeugmaschinen – Abschlußseminar. München: Herbert Utz Verlag Wissenschaft 1997, S. 64–76.

Schönecker 1992

Schönecker, W.: Integrierte Diagnose in Produktionszellen. Berlin: Springer 1992. (iwb Forschungsberichte 45)

Schwager 1983

Schwager, J.: Diagnose steuerungsexterner Fehler an Fertigungseinrichtungen. Berlin: Springer 1983. (isw Forschung und Praxis 48)

Seifert 1992

Seifert, H. J.: Modellgestützte Diagnose komplexer Produktionssysteme – Ein Beitrag zur Erhöhung der Verfügbarkeit kapitalintensiver Fertigungsanlagen. Bochum: Lehrstuhl für Produktionssysteme und Prozeßleittechnik der Ruhr Universität Bochum 1992.

Selic 1994

Selic, B.; Gullenkson, G.; Ward, P. T.: Real-Time Object Oriented Modelling. New York: John Wiley & Sons 1994.

Siemens 1997

Siemens, Siemens Nixdorf: Bedien- und Beobachtungs-system: „Ease of use" – mit allen Vorteilen der Windows-Welt. Software@Siemens, September (1) 1997.

SPS-Magazin 1997

n. n.: PC oder SPS: zwei Wege für ein Ziel. SPS-Magazin (1997) 8, S. 74–77.

Start-Magazin 1998

n.n.; „OPC Bonus Supplement". In: Start-Magazin (1998) 7/8.

Versteegen & Versteegen 1998
Versteegen, C.; Versteegen, G.: UML inside – CASE-Markt: Stand der Dinge. In: iX (1998) 9, S.85–89.

VDI 4008 Blatt 2 1986
VDI 4008, Blatt 2: Boolsches Modell. Düsseldorf: VDI-Verlag 1986.

Vossloh 1988
Vossloh, M.: Modellgestützte Früherkennung und wissensgestützte Diagnose von Fehlern an Werkzeugmaschinen beispielhaft dargestellt an Drehmaschinen. München: Carl Hanser Verlag 1988. (ITW Darmstädter Forschungsberichte für Konstruktion und Fertigung)

Wagner 1997
Wagner, M.: Fehlertolerante Steuerung maschinennaher Abläufe. Berlin: Springer 1997 (iwb Forschungsberichte 106).

Weck 1995
Weck, M.: Werkzeugmaschinen – Fertigungssysteme. Band 3.1 – Automatisierung und Steuerungstechnik 1. Düsseldorf: VDI-Verlag 1995.

Wiedmann 1993
Wiedmann, H.: Objektorientierte Wissensrepräsentation für die modellbasierte Diagnose an Fertigungseinrichtungen. (isw Forschung und Praxis 94)

Zender 1994
Zender, P.: Sicherheitsgerechte SPS in Anlagen mit Gefährdungspotential. atp – Automatisierungstechnik Praxis 36 (1994) 12, S. 27–34.