Planung und Optimierung von prioritätsbasierten Steuergerätenetzen für Fahrzeuge

Von der Fakultät Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von
Matthias Stümpfle
geb. in Ludwigsburg

Hauptberichter: Prof. Dr.-Ing. Dr. h. c. mult P. J. Kühn
Mitberichter: Prof. Dr.-Ing. P. Göhner
Tag der mündlichen Prüfung: 8.2.1999

Institut für Nachrichtenvermittlung und Datenverarbeitung
der Universität Stuttgart
1999

Summary

Electric and electronic components are becoming more and more important in vehicle systems. Functionalities that were based on mechanical components are consequently replaced by electronic components. Additionally, these components ("electronic control units") are increasingly networked via communication systems. These distributed communication systems, many of them implemented using priority based serial bus systems, allow for additional features and services in the vehicles.

The main objective of this work is to develop and provide a method that guarantees the timing behavior (the worst case response time) of services in such a distributed priority system.

Chapter 1 - Introduction

After a short introduction to vehicle network systems this chapter gives a brief overview of the expected impact and market relevance of in-vehicle communication systems. Furthermore, the basic modeling concept of communication requirements - periodic messages - is stated in here.

The relevance for this work stems from the requirement, that many of these new services have to react in real-time, i.e. they have to terminate before a given deadline.

Chapter 2 - Electronic Control Unit Networks in Vehicles

This chapter provides a general overview of the relevant communication concepts and technologies for ECU networks. These include architectures and requirements for in-vehicle networks - in comparison to the ISO-Open Systems Interchange communications reference model - requirements for operating systems and existing communication protocols and standards (including CAN, J1850 and D2B). The chapter closes with an outlook on future hierarchical backbone based high-speed vehicle communication system architectures.

Chapter 3 - Real-Time Priority Systems

Chapter 3 introduces the term real-time and provides a classification of scheduling strategies that can be used for real-time systems. After explaining the periodic modeling idea of system requests it provides an overview of priority based systems and highlights the difference between preemptive and non-preemptive priority systems.

This chapter explains in detail the basic problems and the existing analytical methods for single-processor priority based real-time systems and on finding their worst-case response time as the critical measure for their real-time capabilities. These methods form the basis for the later extension to the networked and distributed systems described in chapter 5.

Chapter 4 - Basics of System Optimization

A basic idea in this work is to use optimization technologies in order to find the appropriate priorities in a given system. Therefore, this chapter provides an overview of the classical optimization techniques as they are known in Operations Research (i.e. linear and non-linear programming). This includes the basic concept of "cost" or "quality function" that is representative of a system. This function then is optimized by modifying its parameters and checking, if the resulting systems better meets the established requirements.

The focus of this chapter is on the so called "nature analogue methods" as they appear to be most relevant for the highly complex NP-hard problem that is to be solved in this work. In this context the concept of chromosomes, alleles and genes and the basic mechanisms of natural reproduction (mutation, cross-over) and selection are introduced.

Chapter 5 - Modeling and Analysis of Vehicle Networks based on Real-Time Priority Systems

A particularity of ECU networks is, that services are provided in collaboration of multiple parts that work together: A service typically consists of two tasks running on different ECUs that are connected via a serial bus system. In this chapter these systems are modeled in a consistent way that treats messages on the bus systems like tasks that are running on an ECU. To show this commonality the term activity is used for both, tasks and messages. Services then become a given sequence of activities running on specific ECUs and bus systems.

After a formal description of these systems this chapter provides the derivation of the necessary algorithms to compute the worst-case response times and therefore extends the single processor system algorithms from chapter 3 to multiprocessor/distributed algorithms. Two such algorithms are presented with the first one providing the exact worst-case response time for distributed services and the second one with an estimation of the worst-case response time. Due to its complexity the first algorithm is only feasible for small systems whereas the second is easily computable and can be used for rather large systems (100 services).

Chapter 6 - Assignment and Optimization of Priorities in Distributed Real-Time Systems

The analytical results from chapter 5 provide an analytical model and computational method to find out if a given priority distribution leads to a system with all services meeting their deadlines. A problem arises, if this is not the case. As seen in the previous chapters, priorities are the only modifiable parameters in such a system. The question therefore is: is it possible to find another priority distribution that allows the system to fulfil the desired timing requirements? The basic idea to solve this problem is to apply optimization techniques.

The most important task to solve in order to apply an optimization technique, is to establish a significant cost or quality function. This was done by a least laxity approach for the service worst-case response times of the distributed system. Additionally, the quality function was normalized to guarantee that the "quality" of a non-functioning system (i.e. at least one service misses its deadline) is always worse than the quality of any working system.

Apart from the quality function a coding scheme for the system had to be found. The system was coded in chromosomes, with each chromosome representing a processor or bus in the system. Processes and messages become the genes on the chromosome and the priorities are the alleles. Furthermore, the basic optimization method cycle is presented here.

The method then is applied to a reference scenario from the SAE (Society of Automotive Engineers) that consists of seven ECUs that are connected to a serial bus system. This system allows for more than 10^176 possibilities to assign priorities. Nevertheless, the presented method was able to find working systems within only a couple of CPU hours, whereas linear search could not find a single solution within 7 CPU days! Based on the SAE scenario different optimization algorithms (Simulated Annealing, Threshold Accepting and Evolution Strategies) were tested with the most effective being the Evolution Strategies in combination with a heuristic starting solution.

The downside of this approach is, that it cannot be guaranteed that a working system can be found or ultimately exists. Furthermore, it is not possible to distinguish between local and global optimums. On the other hand, if a solution exists, this method provides a way to solve this NP-hard problem in polynomial time.

Chapter 7 - Tool Supported Planning

The variety of different methods and techniques that were developed and evaluated in this theses are also implemented in a software tool that is described in this chapter. A modular, object oriented approach (Booch) was taken to model the system and make it easily extendable.

The implementation was done in about 85 C++-classes (15.000 LOC). Furthermore, the compiler tools lex and yacc were used to parse system description files. In these files the system can be described in an easy to use ASCII-notation. As the implemented tool offers many interfaces, it is easy to integrate more tools like e.g. an automatic code generator.

Appendix

The appendix provides several proofs for the analytical parts of the worst-case calculation of the distributed system. Furthermore, a brief introduction into an event triggered simulation is given. This simulation was used to verify the analytical results in this thesis.

Inhaltsverzeichnis

Abkürzungen

ABUS	Automobile Bitserielle Universal Schnittstelle
Ack	Acknowledgement
API	Application Programming Interface (auch: Application Program Interface)
ASN.1	Abstract Syntax Notation.One
ATM	Asynchroner Transfermodus, Asynchronous Transfer Mode
BS	Betriebssystem
CAN	Controller Area Network
CDMA	Code Division Multiple Access
CL, CO	Connectionless, Connection Oriented
CRC	Cyclic Redundancy Code
CS	Circuit Switching
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
D2B	Digital Data Bus, Digital Domestic Bus
DAB	Digital Audio Broadcast
(D)DM	(Distributed-)Deadline-Monotonic-Algorithmus
DL	Data Link (Layer)
DLC	Data Length Code
DPA-Matrix	Dienst-Prozessor-Aktivität-Matrix
DQDB	Distributed Queue Dual Bus
(D)RM	(Distributed-)Rate-Monotonic-Algorithmus
ECU	Electronic Control Unit (Steuergerät)
EMV	Elektromagnetische Verträglichkeit
EOD	End of Data
EOF, EOM	End of Frame, End of Message
FDDI	Fibre Distributed Data Interface
FDM	Frequency Division Multiplexing
GSM	Global System for Mobile Communication (ehem. Groupe Spéciale Mobile)
GPS	Global Positioning System
GY	Gateway

HC	Hamilton Circuit
IDE	Identifier Extension
IFD, IFS	Inter Frame Delimiter, Inter Frame Space
IP	Internet Protocol
IrDA	Infrared Data Association
ISDN	Integrated Services Digital Network
ISO	International Standardization Organisation
LAN	Local Area Network
LLC	Logical Link Control
MAC	Media Access Control
NRZ(I)	Non Return to Zero (Inverted)
N	Network Layer
NP	Nichtdeterministisch Polynomial
nup	(System mit) nichtunterbrechenden Prioritäten
OSI	Open Systems Interconnection
PCP	Priority Ceiling Protocol
PDA	Personal Digital Assistant
PIP	Priority Inheritance Protocol
PS	Packet Switching
PWM	Pulsweitenmodulation, Pulse Width Modulation
RAM	Random Access Memory
ROM	Read Only Memory
RM	Rate-Monotonic-Algorithmus
RTR	Remote Transmission Request
SAE	Society of Automotive Engineers
SAT	Satisfiability-Problem
SCP	Standard Corporate Protocol
SDM	Space Division Multiplexing
SG	Steuergerät (auch ECU)
SLIO	Serial Linked Input Output
SOF, SOM	Start of Frame, Start of Message

SRR	Substitute Remote Request
TCP	Transport Control Protocol
TDM	Time Division Multiplexing
TSP	Travelling Salesman Problem
UDP	User Datagram Protocol
up	(System mit) unterbrechenden Prioritäten
UTP	Unshielded Twisted Pair (ungeschirmte, verdrillte Zweidrahtleitung)
TTP	Time Triggered Protocol
VAN	Vehicle Area Network
WDM	Wavelength Division Multiplexing
ZE	Zeiteinheit
ZSG	Zentrales Steuergerät

Formelzeichen

a	Ausführungsposition einer Aktivität innerhalb eines Dienstes. $1 \leq a \leq a_{max}$, d.h. $a = 1$ ist die erste Aktivität des Dienstes
a_{max}	Letzte Ausführungsposition einer Aktivität innerhalb eines Dienstes
A_s	Aktivität innerhalb eines Dienstes *s*
$A_{s,\,k}$	Aktivität-Darstellung in der DPA-Matrix: *s* gibt den zugehörigen Dienst an und *k* den zugehörenden Prozessor der Aktivität
B	Blockierungsfaktor
C	Computation Time, Bearbeitungszeit
C^*	normierte Bearbeitungszeit, $[C^*] = ticks$
D	Deadline, Frist
D_s	Deadline, Frist eines Dienstes *s*
e_k	Zahl der Aktivitäten auf einem Prozessor *k*
i	Index, entspricht der Priorität der zugehörigen Task/Aktivität
G_s	Menge der Aktivitäten eines Dienstes *s*
k	Komponente eines Steuergerätenetzes. Kann sowohl Bus als auch Prozessor sein
K	Menge der Komponenten *k* eines Systems
l_k	Leistungsfähigkeit einer Komponente *k* (Prozessor oder Bus) in *tick/s*
L_s	Laxity einer Aktivität/eines Dienstes *s*. Die Differenz zwischen Frist und Antwortzeit
$L_{max,\,s}$	Maximale (theoretische) Laxity einer Aktivität/eines Dienstes *s*. Die Differenz zwischen Deadline und Bearbeitungszeit
λ	Anzahl der Nachkommen (einer Population)
μ	Anzahl der Eltern (einer Population)
m	Anzahl der Dienste in einem System
M	Menge der Parametervektoren *S* eines Systems
n	Anzahl der Prozessoren in einem System
(N)	Allgemeine Schicht im ISO-OSI-Basisreferenzmodell
N	Menge der Prozessoren eines Systems
O	Offset

Π	Population, Menge von Individuen zu einem bestimmten Zeitpunkt
p	statische Priorität einer Aktivität A
P	Busy Period, Aktivitätszeit
q	Zählindex für die Anforderungen einer Aktivität
Q	Qualitätsfunktion (auch Ziel- oder Fitnessfunktion). Auch Variable, die den Qualitätswert angibt
r	Worst-case-Antwortzeit
r_s	Worst-case-Antwortzeit eines Dienstes s
R	Antwortzeit (auch Durchlaufzeit)
ρ	Auslastungsgrad, Utilization eines Betriebsmittels
$\Re$	Abhängig vom Typ des Prioritätensystems entweder die Aktivitätszeit (k: up) oder die Wartezeit (k:nup)
S	Systemzustandsvektor, $S \in M$
s	Dienst (Transaktion), der durch das Steuergerätenetz erbracht wird, $s \in X$. Außerdem gilt: $1 \leq s \leq m$
s	Systemparameter, Element eines Systemzustandsvektors
t	allgemeine Zeitvariable
T	Ankunftsabstand zwischen zwei Anforderungen
T_i, $T(A_{s,k})$	Ankunftsabstand zwischen zwei Anforderungen der Aktivität i
T_S	Ankunftsabstand zwischen zwei Anforderungen eines Dienstes s
U	Zufallsvariable, gleichverteilt im Intervall [0,1]
V	Verbindungsmatrix eines Steuergerätenetzes. Gibt an, welche Komponenten k des Netzes miteinander verbunden sind
W	Wartezeit
X	Menge der Dienste S (Transaktionen) eines Systems
niederpriore Task(s)	Task(s), die, relativ zu einer betrachteten Task i, eine niedrigere Priorität haben
höherpriore Task (s)	Task(s), die, relativ zu einer betrachteten Task i, eine höhere Priorität haben

Kapitel 1

Einleitung

1.1 Elektronische Systeme in Fahrzeugen

Der Stellenwert von elektrischen und elektronischen Komponenten in einem Fahrzeug ist seit geraumer Zeit mit einem stetigen Wachstumsfaktor belegt. Viele der bisher vorhandenen Funktionen, die auf mechanischen Lösungen basieren, werden in immer stärkerem Maße durch elektrische Funktionen ergänzt oder gar ersetzt. Dies dient einerseits der Erhöhung von Fahrsicherheit und Fahrkomfort. Andererseits ergeben sich weitere positive Effekte: z.B. im Bereich des Umweltschutzes, in welchem durch effizientere Motorenregelung Emissionen reduziert werden können und vor allem auch im Bereich der Montagefreundlichkeit sowie der anschließenden Wartbarkeit der Produkte, womit eine allgemeine Kostenreduktion erzielt wird.

Außer den bereits vorhandenen, traditionellen Funktionen lassen sich mit Hilfe der Elektronik und der damit zusammenhängenden Kommunikationstechnik ganz neue Funktionsqualitäten realisieren. Komfortdienste, so z.B. das gemeinsame Schließen aller Türen, Fenster und des Schiebedachs, deren Realisierungen bisher an dem zu hohen Verkabelungs- und Steuerungsaufwand scheiterten, werden durch Kommunikationstechnik möglich. Auch die Telematikdienste, wie beispielsweise Autopilotsysteme zur Zielfindung oder die Ferndiagnose, bei der ein liegengebliebenes Fahrzeug Kontakt mit einer Werkstatt aufnehmen kann, gehören zu dieser neuen Kategorie von Diensten. Die durchdringende Einführung dieser Dienste steht unmittelbar bevor und wird einen neuen Boom sowohl für die Diensteanbieter, die nicht mehr notwendigerweise die Fahrzeughersteller selbst sind, als auch für die Hersteller der notwendigen Komponenten bringen.

Wirtschaftsforscher prognostizieren für die in Fahrzeugen installierten Datennetze, welche die Grundlage für die Kommunikation zwischen den Komponenten darstellen, ein exponentielles Wachstum [14]: 1992 waren weltweit 1,53 Mio. Fahrzeugnetze installiert, 1996 werden es über 5,31 Mio. sein und für das Jahr 2000 erwartet man 16,8 Mio Netzinstallationen. Deshalb wird in diesem Zusammenhang oft vom kommenden „Zeitalter der Fahrzeugkommunikation“ gesprochen [8]. Die mit Hilfe dieser Netze möglichen neuen Funktionen und Dienste gewinnen überdies unter dem Hintergrund der Diversifizierung, d.h. des Erzeugens von Unterschieden zur Marktkonkurrenz, immer mehr an Bedeutung. In einer Phase, in der die Fahrzeuge, bedingt durch die physikalischen Erkenntnisse im Bereich des Karosserie- und Motorenbaus, immer ähnlicher werden, müssen neue Gebiete gefunden werden, in denen sich die Produkte voneinander unterscheiden können. Somit wird ein immer größerer Beitrag der Wertschöpfung

aus den Funktionen der Elektrik und Elektronik mit der dafür notwendigen Software gewonnen.

Vielen der neuen Funktionen und Dienste ist gemein, daß sie durch die Zusammenarbeit mehrerer Komponenten erbracht werden. Die Gründe dafür sind unterschiedlicher Natur: teilweise liegen die notwendigen externen Daten nur an bestimmten Stellen im Fahrzeug vor (z.B. Raddrehzahlmessung). Andere Ursachen können in der logischen Konzeption und Architektur des Systems liegen (z.B. Client-Server-Konzept mit einem exponierten Zentralrechner). Eine solche Zusammenarbeit setzt voraus, daß die beteiligten Komponenten miteinander kommunizieren können. Dies wird in neueren Fahrzeugen dadurch erreicht, daß die Steuergeräte, die diese Funktionen oder Teile einer Funktion realisieren, über geeignete Kommunikationssysteme miteinander zu lose gekoppelten Systemen verknüpft werden. Dabei kommen zur Zeit hauptsächlich serielle, prioritätsgesteuerte Bussysteme zum Einsatz. Durch die Verwendung der Bussysteme läßt sich sowohl bei der Herstellung der Fahrzeuge (Kabelmenge, Montagezeit) als auch beim Betrieb des Fahrzeugs (geringeres Gewicht, leichtere Wartung und Erweiterbarkeit des Systems) eine immense Kostenersparnis erzielen.

Ein Steuergerät, welches als Knoten innerhalb eines derartigen Kommunikationsnetzes arbeitet, beinhaltet typischerweise einen Mikrocontroller, d.h. einen Einchip-Computer, dessen Software die geforderte Steuerungs- oder Regelungsfunktion erbringt. Zur Aufnahme von Umwelteinflüssen, bzw. zur aktiven Reaktion sind an den Mikrocontroller entsprechende Peripheriebausteine (Sensoren und Aktoren) angeschlossen.

Bedingt durch den Einsatz dieser Kommunikationssysteme in Fahrzeugen stehen zwei Anforderungen konkurrierend an zentraler Stelle: Da die Herstellung von Fahrzeugen in den Bereich der Massenfertigung fällt, ist die Einsparung von Kosten einerseits bei der Entwicklung und Fertigung, andererseits ebenso bei der Wartung von eminenter Wichtigkeit. Bei großen Stückzahlen machen sich bereits kleinste Einsparpotentiale stark bemerkbar. Demgegenüber steht bei vielen der durch das System zu erbringenden Dienste die Anforderung der Echtzeitfähigkeit (s.u.). Es ist leicht einzusehen, daß diese Anforderungen miteinander konkurrieren, weil die Echtzeitfähigkeit durch leistungsfähigere und damit teure Komponenten eher erzielbar ist, als durch die angestrebten, kostengünstigen Bauteile. Dies unterscheidet die hier betrachteten Systeme auch deutlich von beispielsweise großen Vermittlungssystemen der Telekommunikation, die der Investitionsgüterherstellung zuzuordnen sind, oder auch, im Bereich der Transportsysteme, von Flugzeugen, die ebenfalls in geringeren Stückzahlen gefertigt werden und für die deshalb ganz andere Kostenrandbedingungen gelten.

1.2 Charakteristika von Anwendungen und Diensten in Fahrzeugkommunikationsnetzen

Will man die Architektur eines Kommunikationssystems bewerten oder planen, dann müssen die Anforderungen der Anwendungen bekannt sein, welche dieses Kommunikationsnetz verwenden wollen. Die Anwendungen, die mit Hilfe der Steuergerätenetze realisiert werden sollen, weisen im Bezug auf ihr zeitliches Verhalten Unterschiede auf. Betrachtet man die Gesamtvernetzung eines modernen Fahrzeugs [156, 130, 126], so lassen sich die folgenden Grundanwendungen erkennen:

- Anwendungen, die zyklisch bestimmte Anforderungen erfüllen müssen. Hierzu gehören z.B. Teile der Motorsteuerung mit ihren Regelkreisen, bei denen die Drehzahl in bestimmten Zeitabständen über Sensoren bestimmt und anschließend an andere Geräte weitergegeben werden muß. Dabei setzen die physikalischen Vorgänge den zeitlichen

Rahmen für die Periodizität der Regelkreise. Ein Zuspätkommen kann nicht toleriert werden, da sonst die Funktionsfähigkeit des Systems in Gefahr gerät.

- Weitere Anwendungen und damit verbundene Ereignisse sind von nichtzyklischer, aperiodischer Natur. Treten diese jedoch mit einer bestimmten Häufigkeit auf, so können sie durch einen periodischen Vorgang angenähert werden. Hierunter lassen sich z.B. Statusmeldungen, wie „Blinker links an“ oder „Bremslicht aktiviert“, einordnen.
- Außerdem gibt es Ereignisse, die noch seltener sind als die aperiodischen: sogenannte sporadische Ereignisse. Wird beispielsweise der Airbag eines Fahrzeugs ausgelöst, so kann man davon ausgehen, daß die Zeit für die Wiederauslösung dieses Airbags gegen unendlich geht, da dieses Ereignis für das System nur einmal eintritt. Auf der anderen Seite muß allerdings das Einhalten einer zeitlichen Frist für das Auslösen des Airbags, nach dem Erkennen eines Unfalls, gewährleistet werden.
- Schließlich gibt es Anwendungen, die keinen engen zeitlichen Anforderungen unterliegen. Zumindest sind die Anforderungen diesbezüglich deutlich schwächer als bei den drei zuerst genannten Fällen. Hierunter fallen Anwendungen, wie z.B. die Bereitstellung der Fahrerinformation (Tankanzeige, Außentemperatur etc.).

Viele Anwendungen, die im ersten Augenblick als zeitunkritisch erscheinen, z.B. die Innenraum-Komfortfunktionen, müssen jedoch bei näherer Betrachtung meist ebenfalls sicherheitskritische Anforderungen erfüllen. Als Beispiel sei ein elektrischer Fensterheber genannt, der sofort auf ein Stopp-Signal reagieren muß, wenn entsprechende Sensoren ein zu großes Drehmoment des Motors gemessen haben, das von einem eingeklemmten Körper herrührt.

Außer den zeitlichen Anforderungen, die für das korrekte Funktionieren des Systems verantwortlich sind, existieren ebenso Anforderungen bezüglich der Datensicherheit (security), der Zuverlässigkeit oder Ausfallsicherheit (reliability) und der Systemsicherheit (safety) solcher Systeme. Diese Aspekte werden allerdings in dieser Arbeit nicht weiter in Betracht gezogen.

1.3 Übersicht über die Arbeit

Die vorliegende Arbeit befaßt sich mit der Analyse und der Planung von Echtzeit-Fahrzeugkommunikationssystemen auf der Basis von Prioritätensystemen. Dabei wird ein besonderes Augenmerk auf die zentrale Anforderung an ein derartiges System gelegt: die Einhaltung und die Gewährleistung der Echtzeiteigenschaften. Für die Planung dieser Systeme müssen deshalb effiziente Algorithmen bereitgestellt werden, die ein Einhalten der Anforderungen sicherstellen. Dabei wird sich zeigen, daß das Kommunikationssystem nicht isoliert, sondern vielmehr im Kontext mit den Diensten und Anwendungen gesehen werden muß, die diese Kommunikation verwenden.

Als Ziele der Arbeit ergeben sich die folgenden Punkte:

- Die Darstellung einer allgemeinen Architektur für Fahrzeugkommunikationssysteme,
- die Modellierung eines solchen Systems und die Beschreibung der einzuhaltenden Randbedingungen,
- die Präsentation einer Methode, welche die Anforderungen mit den Randbedingungen in Einklang bringt,
- der Nachweis der Funktionsfähigkeit der Methode und
- die Umsetzung der gewonnenen Verfahren und Methoden in ein Planungswerkzeug.

Der für das Verständnis der Arbeit notwendige Rahmen wird in Form allgemeiner Grundlagen in Kapitel 2 dargestellt. Hier werden Kommunikationsprinzipien und -modelle vorgestellt, die für den Betrieb solcher Fahrzeugnetze von Bedeutung sind. Wo nötig, werden die speziellen Anforderungen aufgezeigt, die für ein solches System gelten. Dies geschieht insbesondere für die Bereiche offene Systemarchitektur, Übertragungskanal, Medienzugriff sowie Software-Strukturen für höhere Schichten. Im Anschluß daran folgt eine kurze Darstellung von Betriebssystemmechanismen, welche in Kapitel 3 weiter vertieft werden. Schließlich wird ein Überblick über existierende Protokolle und Standards für Kommunikationssysteme in Fahrzeugen gegeben. Mit einem Ausblick auf künftige automotive Netzarchitekturen schließt das Kapitel.

Das 3. Kapitel befaßt sich mit der Darstellung existierender Entwurfs- und Analysemethoden für Echtzeitprioritätensysteme, die auf einem Prozessor basieren. Hierzu werden der Begriff der Echtzeitfähigkeit sowie eine Klassifikation von Scheduling-Strategien, mit einem Schwerpunkt auf den Prioritätensystemen, präsentiert. Außerdem werden die Anforderungen und Dienste eines solchen Echtzeitprioritätensystems formal beschrieben. Der Hauptteil dieses Kapitels ist den Analysemethoden für solche Echtzeitsysteme gewidmet: Nach der Darstellung bekannter Analyseverfahren erfolgt eine schrittweise Entwicklung und Herleitung des Algorithmus‘ zur Berechnung der Worst-case-Antwortzeit von Aktivitäten auf einem Prozessor. Als offensichtliches Ergebnis kann hier schon festgehalten werden, daß ausschließlich die vergebenen Prioritäten das zeitliche Verhalten der Aktivitäten festlegen. Der letzte Abschnitt des Kapitels beschäftigt sich mit der Bestimmung der Ausführungszeit von Aktivitäten.

Im nachfolgenden Kapitel 4 erscheint eine Übersicht über die Grundlagen der Systemoptimierung, wie sie im Hinblick auf die Prioritätenvergabemethoden notwendig sind. Dazu wird, nach einer Beschreibung erreichbarer Optimierungsziele, eine Taxonomie von Optimierungsverfahren vorgenommen. Mit der nachfolgenden Einführung in die Komplexitätsbetrachtung und der Klassifikation von sog. „schwierigen“ Problemen (Theorie der *NP*-vollständigen Probleme) reduziert sich die Zahl der verwendbaren Optimierungsverfahren für das vorliegende Problem. Den Schwerpunkt der hierzu vorgestellten Verfahren bilden Optimierungsansätze, die der Natur entnommen sind, so z.B. das simulierte Ausglühen (Simulated Annealing), das Toleranzschwellenverfahren (Threshold Accepting) und die Evolutionsstrategien.

Kapitel 5 bildet die Fortsetzung zu Kapitel 3, in dem die dort gefundenen Beziehungen nunmehr auf verteilte Systeme erweitert werden. Zunächst wird die Strukturierung des Steuergerätenetzes durch ein entsprechendes Modellierungsverfahren vorgenommen. Mittels dieses Modells kann das System der nachfolgend präsentierten Analyse zugänglich gemacht werden. Im Rahmen der Analyse wird ein Algorithmus entwickelt, der die exakte Bestimmung der Worst-case-Antwortzeit der betrachteten Dienste erlaubt. Der hohe rechentechnische Aufwand für diesen Algorithmus bei der Analyse größerer Systeme macht allerdings die Suche nach einer Alternative nötig. Dies erklärt die Notwendigkeit für die Präsentation eines zweiten, approximativen Algorithmus, der als Ergebnis zwar nur eine obere Schranke des Worst-cases liefert, diese jedoch in vertretbarer Zeit berechnet.

In Kapitel 6 werden Möglichkeiten untersucht, auf welche Weise die Prioritäten in einem verteilten System vergeben werden müssen, damit die Echtzeitanforderungen eingehalten werden. Nach der Vorstellung zweier einfacher heuristischer Verfahren kommen die aus Kapitel 4 bekannten Optimierungsmethoden zur Prioritätenvergabe zum Einsatz. Um diese Methoden anwenden zu können, wird zuerst eine Qualitätsfunktion benötigt, die eine Systemqualität beschreibt, welche als Maß für die Echtzeitfähigkeit des Systems verwendet wird. Daraufhin erfolgt eine nähere Untersuchung unterschiedlicher Optimierungsmöglichkeiten sowie eine Diskussion der grundsätzlichen Einsatzfähigkeit der Verfahren für das Prioritätenvergabepro-

blem. Zur Erzielung aussagekräftiger Ergebnisse wird ein großes Referenzsystem, dessen Spezifikation geeignet ergänzt wurde, mit Hilfe der entwickelten Methode bearbeitet.

Die präsentierten Methoden und Verfahren wurden in einem Planungswerkzeug prototypisch implementiert. In Kapitel 7 wird die grundlegende, objektorientierte Architektur des Werkzeugs vorgestellt und sein Einsatz exemplarisch demonstriert. Mit einer Bewertung der Einsatzfähigkeit des Werkzeugs schließt dieses Kapitel.

Die Arbeit endet in Kapitel 8 mit einer Zusammenfassung der wichtigsten Ergebnisse und gibt dort einen Ausblick auf zukünftige Erweiterungen der behandelten Ansätze.

Abschließend soll auf folgendes Zitat von Conway et. al. aus dem Buch „Theory of Scheduling“ [25] aufmerksam gemacht werden:

> *„The job-shop problem is a fascinating challange. Although it is easy to state, and to visualize what is required, it is extremely difficult to make any progress whatever toward a solution. Many proficient people have considered the problem, and all have come away essentially empty-handed. Since this frustration is not reported in the literature, the problem continues to attract investigators, who just cannot believe that a problem so simply structured can be so difficult, until they have tried it.*“
>
> Conway, Maxwell, Miller, 1967

Trotz dieser Warnung wird das Problem der Echtzeitplanung in einem verteilten System in der vorliegenden Arbeit angegangen. Dies scheint um so lohnenswerter, als eine gefundene Lösung große Vorteile bei der Planung und Entwicklung von verteilten Echtzeitprioritätensystemen in Fahrzeugen verspricht. Viele Einsatzgebiete für derartige Systeme, insbesondere im sicherheitskritischen Bereich, werden durch solch eine Lösung überhaupt erst möglich.

Kapitel 2

Steuergerätenetze in Fahrzeugen

Ziel dieses Kapitels ist es, einen allgemeinen Überblick über Steuergerätenetze für Fahrzeuge und die darin enthaltenen Kommunikationssysteme zu geben. Dazu werden die typischen Merkmale eines solchen Netzes sowie die speziellen Anforderungen aufgezeigt, die im automotiven Bereich herrschen.

Nach einer Übersicht über die für solche offenen, verteilten Systeme notwendigen Kommunikationsmechanismen werden auch Betriebssystemaspekte angesprochen, die für die Betrachtungen in den späteren Kapiteln eine Rolle spielen. Schließlich werden Realisierungen für den automotiven Bereich vorgestellt, wobei der Schwerpunkt auf den – später intensiver betrachteten – seriellen Bussen mit bitweiser Arbitrierung liegt. Abschließend sollen zukünftige Netzarchitekturen, die auch die Anbindung an externe Netze vorsehen, angesprochen werden.

2.1 Architektur und Merkmale von Kommunikationssystemen für Fahrzeuge

Bild 2-1 zeigt ein typisches Fahrzeugnetz, das aus zwei Bussystemen besteht. Die beiden gezeigten Bussysteme, Motorbus und Innenraumbus, werden über das Zentralsteuergerät (ZSG) miteinander verbunden. Exemplarisch ist der Aufbau zweier Steuergeräte gezeigt: Einmal die Software-Sicht links und im rechten Steuergerät die verwendeten Hardware-Komponenten. Zur Strukturierung des Systems bieten sich diese Sichten auf die in den Netzen verwendeten Steuergeräte an: Die Hardware-Architektur repräsentiert die in einem Steuergerät enthaltenen Komponenten. Hier findet man heute einfache 8 bit- oder 16 bit-Mikrocontroller wieder. Außerdem werden zur Abwicklung des Kommunikationsprotokolls Buscontroller-Bausteine verwendet. An die Schnittstellen des Mikrocontrollers werden die Sensoren und Aktoren angeschlossen, die zum einen die für das System notwendigen Werte erfassen und zum anderen über die Aktoren entsprechende Reaktionen veranlassen.

Die Software-Architektur klärt das Zusammenwirken der Anwendungsprogramme auf der Basis eines Betriebssystems unter der Zuhilfenahme von Kommunikationsdiensten, die ein Kommunikationssubsystem leistet, welches Teil des Betriebssystems sein kann.

Das dargestellte Bild zeigt außerdem ein Beispiel für die Kommunikationsanforderungen durch Dienste. Zur Veranschaulichung soll ein Zentralverriegelungsdienst realisiert werden. Hierzu nimmt das Zentralsteuergerät den Impuls für die Verriegelung über einen Sensor (Schalter) auf und gibt diesen über das Kommunikationsmedium „Innenraumbus“ an die Steu-

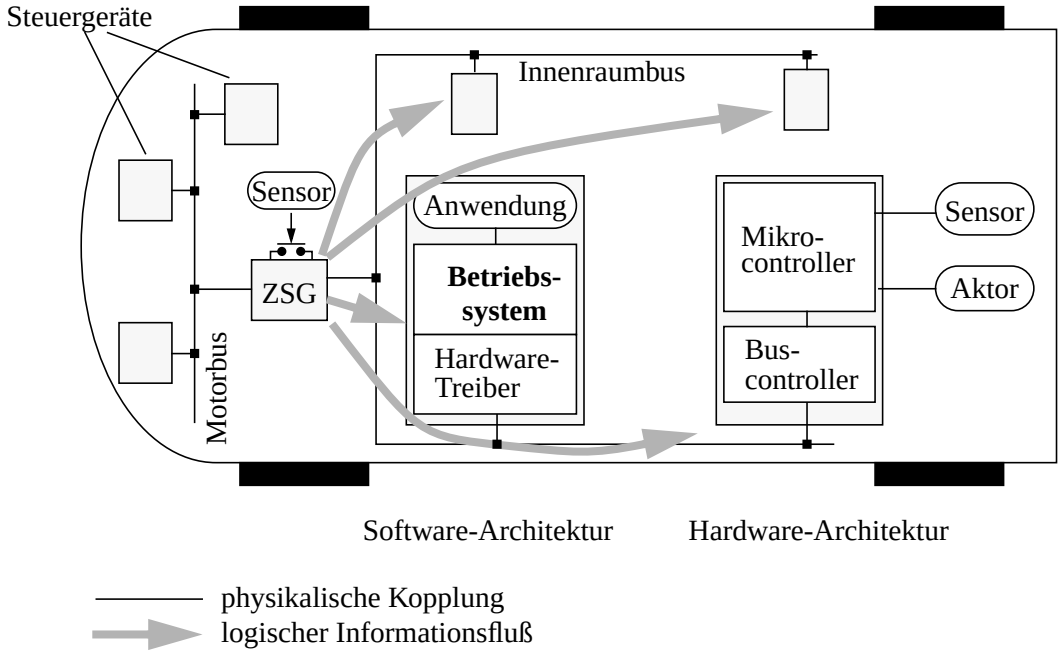


Bild 2-1: Fahrzeugsteuergerätenetz

ergeräte weiter, die sich an den Türen befinden. Hier werden schließlich, als letztes Glied in dieser Ablaufkette, die Aktoren (Verriegelungsmotor) zum Verschließen der Türen betätigt.

In den folgenden Unterkapiteln werden allgemeine Merkmale von Kommunikationssystemen und ergänzend dazu die speziellen Merkmale von Fahrzeug-Kommunikationssystemen dargestellt. Nach einer kurzen Erläuterung der allgemeinen Prinzipien für Kommunikationssysteme werden außerdem, entsprechend des ISO-OSI-Schichtenmodells, die einzelnen Funktionen derartiger Systeme beschrieben.

2.1.1 Architekturanforderungen an ein offenes Kommunikationssystem

Das Steuergerätenetz eines Fahrzeugs wird durch eine Menge unterschiedlicher Steuergeräte gebildet. Verschiedene Zulieferer erhalten vom Fahrzeughersteller die Spezifikation für ein bestimmtes Steuergerät und liefern dieses in der Regel zum direkten Einbau. Sollen diese Geräte miteinander kommunizieren können, so ist eine offene, verteilte Kommunikationsarchitektur, die Schnittstellen zur Verfügung stellt, welche jedem Lieferanten bekannt sind, unumgänglich. Einen möglichen Leitfaden für derartige Schnittstellen bietet das OSI-Schichtenmodell.

2.1.1.1 Das Schichtenmodell für offene, verteilte Systeme

Die Verbindung von Computersystemen unterschiedlicher Herkunft ist an das Vorhandensein offener Schnittstellen gebunden, die eine Zusammenarbeit möglich machen. Über diese Schnittstellen wird festgelegt, welche Funktionalitäten bereitzustellen sind, damit z.B. eine Verbindung aufgebaut werden und ein Informationsaustausch stattfinden kann. Um dies im Bereich der Kommunikation von Computersystemen zu realisieren, wurde von der Internationalen Organisation für Standardisierung (ISO) das Basisreferenzmodell für die Kommunika-

tion offener Systeme (*Open System Interconnection Reference Model*) definiert, das in einer sieben Schichten umfassenden Beschreibung die Zusammenarbeit zwischen Kommunikationssystemen festlegt. Dieses Modell liegt seit 1984 als internationaler Standard vor [70]. Es handelt sich bei diesem Modell allerdings weniger um eine Implementierungsanleitung, als vielmehr um die Benennung der für die Kommunikation notwendigen Funktionen und deren Einteilung in die genannten Schichten.

Bild 2-2 zeigt die von der ISO vorgeschlagene Schichtung des Referenzmodells. Aufgrund von detaillierteren Beschreibungen bestimmter Funktionsgruppen und der Erhebung proprietärer Firmenlösungen zu Standards innerhalb einzelner Schichten wurde es nötig, einige der Schichten feiner zu gliedern. Die Abbildung zeigt die für den Bereich der lokalen Netze wichtige Unterteilung der Sicherungsschicht (Data Link Layer, DL) in die zwei Unterschichten 2a, Medienzugriffskontrolle (Media Access Control, MAC) und 2b, logische Sicherungsschicht (Logical Link Control, LLC). Das dem Modell zugrundeliegende Schichtungsprinzip sieht vor, daß eine Schicht (N), der Dienstnutzer (service user), auf den Funktionen und Diensten der darunterliegenden Schicht (N-1), dem Diensterbringer (service provider), aufbaut. Objektorientiert modelliert ausgedrückt „erbt“ der *Service User* vom *Service Provider* dessen Funktionalität. Der Service User fügt sodann dem System „Kommunikation“, dem beide angehören, neue schichtspezifische Funktionen hinzu. Dem Anwender stehen die Funktionen eines Providers in Form einer Anwendungsschnittstelle (Application Programming Interface, API) zur Verfügung, d.h. in einer programmtechnischen Lösung kann der Anwender eine wohldefinierte Menge von Funktionen und Methoden verwenden, um seine kommunikationstechnischen Aufgaben zu lösen. Für weitere Ausführungen zu den Funktionen und Protokollen des Referenzmodells siehe [96, 70, 145].

Betrachtet man die einzelnen Schichten von unten nach oben, so kann man zwei funktionell unterschiedliche Gruppen unterscheiden: Die Schichten 1-4 werden zum sogenannten „Transportsystem“ zusammengefaßt. In diesen Schichten sind die Funktionen für die Einrichtung

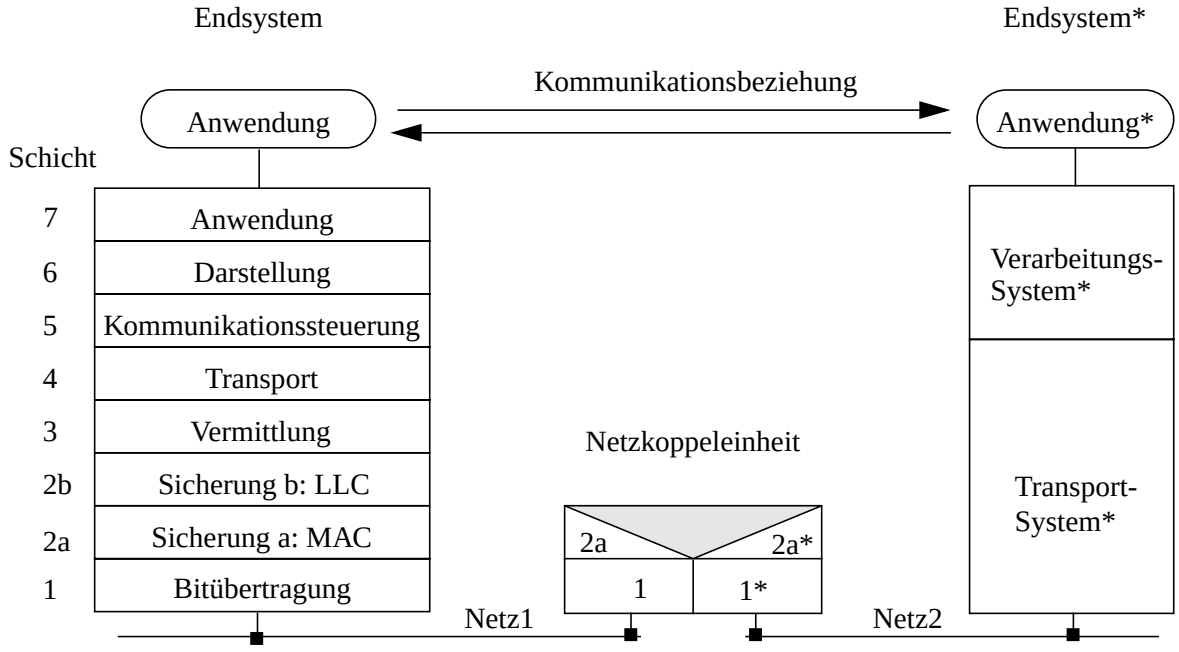


Bild 2-2: Das Basisreferenzmodell, illustriert an zwei Netzknoten, die auf der Sicherungsschicht über einen Knoten mit Netzkoppeleigenschaften miteinander verbunden sind.

einer sicheren Verbindung zum Datenaustausch zwischen Endsystemen spezifiziert. Die Schichten 5-7 bilden das „Verarbeitungssystem“ und sind mehr anwendungsorientiert, d.h. hier finden sich Funktionen, die es den Applikationen erleichtern, die Kommunikationseinrichtung effizient zu verwenden. Im einzelnen werden den Schichten die folgenden Funktionen zugeordnet:

- Schicht 1, die Bitübertragungsschicht (physical layer), legt die physikalischen Eigenschaften des Übertragungskanals der Verbindung fest. Hier werden Signalpegel, die Repräsentation eines Bits auf dem Medium sowie die Dimensionierung von Steckverbindern vereinheitlicht. Typische Standards, welche die Bitcodierung auf dieser Schicht festlegen, sind z.B. die RS-232-Schnittstelle und die Manchester-Bitcodierung.
- Schicht 2, die Sicherungsschicht (data link layer), ist in zwei Teile aufgetrennt: Die technologieabhängige Medienzugriffssteuerung ist für die Vergabe des Mediums an die konkurrierenden Kommunikationsteilnehmer verantwortlich, während die logische Sicherungsschicht Funktionen zur Übermittlung von Datenblöcken bereitstellt. Hier werden Fehler erkannt und eine Synchronisation für den Übertragungsabschnitt durchgeführt.
- Schicht 3, die Vermittlungsschicht (network layer), sorgt dafür, daß Knoten über mehrere Teilnetze hinweg verbunden werden. Dazu sind hier die Funktionen der Verkehrslenkung und, damit verbunden, der Auswertung von Netzadressen vorgesehen.
- Schicht 4, die Transportschicht (transport layer), ermöglicht die Ende-zu-Ende-Kommunikation über mehrere Teilsysteme hinweg, d.h. die kommunizierenden Partner befinden sich jeweils im Endsystem. Die dazwischenliegenden Transfersysteme sind für sie transparent.
- Schicht 5, die Kommunikationssteuerungsschicht (session layer), erlaubt eine Dialogsteuerung mit der Bereitstellung von Synchronisationspunkten.
- Schicht 6, die Darstellungsschicht (presentation layer), legt das Datenformat und den Aufbau von Datenstrukturen fest. Ein Standard, der dies ermöglicht, ist die Beschreibungssprache ASN.1 [71].
- Schicht 7, die Anwendungsschicht (application layer), bildet die eigentliche Schnittstelle zu den Applikationen, die Kommunikationsdienste in Anspruch nehmen wollen. Hier sind unterstützende Funktionen, wie z.B. die Übertragung vollständiger Dateien oder Mail-Dienste angesiedelt.

Man spricht in diesem Zusammenhang, bei der Komposition der Schichten zu einem Kommunikationssystem, auch von einem Protokollstapel (protocol stack). Für die Realisierung einer funktionierenden Kommunikationsarchitektur müssen aber nicht alle Funktionen einer Schicht, oder gar alle Schichten selbst, vorhanden sein. Dies kann u.U. sogar unnötigen Overhead und damit eine Vergrößerung der Bearbeitungszeit verursachen. Bei lokalen Netzen bestehen typische Architekturen aus einer Kombination der Schichten 1, 2 und 7, wobei die Schicht 7 im wesentlichen die Funktionen der Schicht 2 bereitstellt.

Viele aktuelle Protocol Stacks (z.B. die TCP/IP Protocol Suite) kommen mit weniger als diesen sieben Schichten aus. In der Regel werden in modernen Systemen nur Funktionen umgesetzt, die den OSI-Schichten 1-4 und 7 entsprechen. Dies führte dazu, daß Forderungen nach einer Reform des OSI-Modells immer lauter wurden [146], in der z.B. die Schichten fünf und sechs mangels sinnvollen Einsatzes nicht mehr aufgeführt werden.

2.1.1.2 Netzstrukturen, Netzkopplung und Verkehrslenkung

Durch die *Netzstruktur* wird die Verknüpfung von Systemkomponenten, wie z.B. den Steuergeräten untereinander, zu einer funktionalen Einheit beschrieben. Dabei werden die Knoten des entstehenden Netzgraphen durch die Steuergeräte und die Kanten des Graphen durch die Verkabelung zwischen diesen Komponenten repräsentiert. Die zu einem solchen Netzgraphen verknüpften Knoten eines Kommunikationssystems lassen sich grundsätzlich in vier verschiedene Topologien einordnen (siehe Bild 2-3): Bus, Stern, Ring und Vollvermaschung.

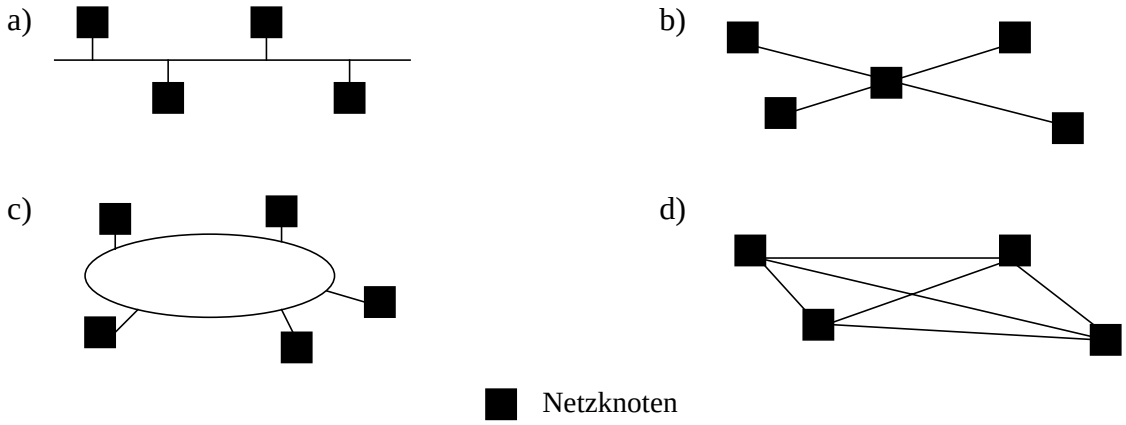


Bild 2-3: Topologien lokaler Netze. Bussysteme (a) und Ringsysteme (c) sind die typischen Topologien für LAN. Sternsysteme (b) entsprechen einem Bussystem mit einer minimalen Buslänge. Die Topologie ist meist durch die Protokollspezifikation vorgegeben, da das verwendete Medienzugriffsverfahren in der Regel topologieabhängig ist. Vollvermaschte Systeme (d) benötigen die maximale Kabellänge.

Bus-, Stern- und Ringsysteme haben für die betrachteten Fahrzeugsysteme den höchsten Stellenwert. Vollvermaschte Systeme scheiden aufgrund des hohen Verkabelungsaufwands aus. Betrachtet man die Evolution der Fahrzeugverkabelung, dann sollen gerade die vollvermaschten Systeme durch die anderen Topologien ersetzt werden. Kombiniert man die Grundstrukturen, so lassen sich sehr komplexe Netze aufbauen, die auch eine hierarchische Gliederung des Systems zulassen (s.u.).

Netzkopplung, d.h. die Verbindung verschiedener Netze, kann in zwei Bereiche unterteilt werden: zum einen die Ankopplung externer Netze an die Fahrzeugnetze und zum anderen die Ankopplung interner Netze untereinander, z.B. um hierarchische Strukturen aufzubauen. Die Netzkoppeleinheiten müssen dabei u.U. zwischen verschiedenen Protokollen der Schicht 2 vermitteln (Bridge-Funktionalität) und sie aufeinander abbilden. Dies ermöglicht, außer der Kopplung unterschiedlicher Netze, auch eine Begrenzung der Stationenzahl und damit verbunden eine Begrenzung der Last auf einem Teilnetz. Wird dasselbe Protokoll auf Schicht 2 verwendet, dann kann durch sog. Repeater eine Signalregeneration durchgeführt und damit das Netz in seiner Ausdehnung vergrößert werden. Aufgrund der begrenzten Längen im Fahrzeug ist dort der Einsatz von Repeatern aber eher unwahrscheinlich.

Wird ein Knoten als Netzkoppeleinheit betrieben, dann soll er entscheiden können, welche Information von einem Teilsegment in das andere geleitet wird. Es findet damit eine Form der

Verkehrslenkung (routing) statt. Für diese Funktion muß eine Bridge Information über die an den Teilsegmenten angeschlossenen Knoten haben, die sie entweder selbstlernend erhält oder, was im automotiven Bereich aus Kostengründen wahrscheinlicher ist, durch Vorgabe der Konfiguration kennt. Zu den möglichen Formen der Verkehrslenkung auf Schicht 2 siehe [16]. Netzkopplungseinheiten für höhere Schichten existieren noch für die Schicht 3 in Form von *Routern* und für die Schichten 4-7 in Form von *Gateways*. Diese Formen sind allerdings eher für Weitverkehrsnetze als für Fahrzeugnetze interessant. Entsprechende Literatur findet sich bei [133, 53].

Die Verkehrslenkung hat das Ziel, den „günstigsten“ Weg von einer Verkehrsquelle zu einer Verkehrssenke zu wählen. Dies setzt natürlich das Vorhandensein von Alternativwegen voraus, was aufgrund der Kostenoptimierung in Fahrzeugsystemen selten der Fall sein wird; hier finden sich in der Regel baumartige, hierarchische Strukturen, die keine Alternativrouten zulassen.

Durch eine Hierarchiebildung ist es möglich, ein leistungsstarkes, d.h. mit hoher Bandbreite versehenes, System als „Backbone“ zu verwenden und im Teilnehmeranschlußbereich mit Systemen zu arbeiten, die eine geringere Bandbreite aufweisen und damit kostengünstiger sind. Bild 2-4 zeigt eine derartige Anordnung mit einem schnellen Ringsystem auf der obersten Hierarchiestufe, daran angekoppelt, auf der mittleren Hierarchieebene, ein Bussystem und, im Teilnehmeranschlußbereich, eine Sterntopologie. Die notwendige Bandbreite nimmt dabei von oben nach unten ab. Knoten, die in zwei Topologien wirksam sind, arbeiten als Gateways (GY).

Eine derartige Architektur läßt sich außerdem auch leicht in Bereiche mit unterschiedlichen Sicherheitsanforderungen unterteilen. (Im Bild dienen zur Unterscheidung dieser Bereiche die

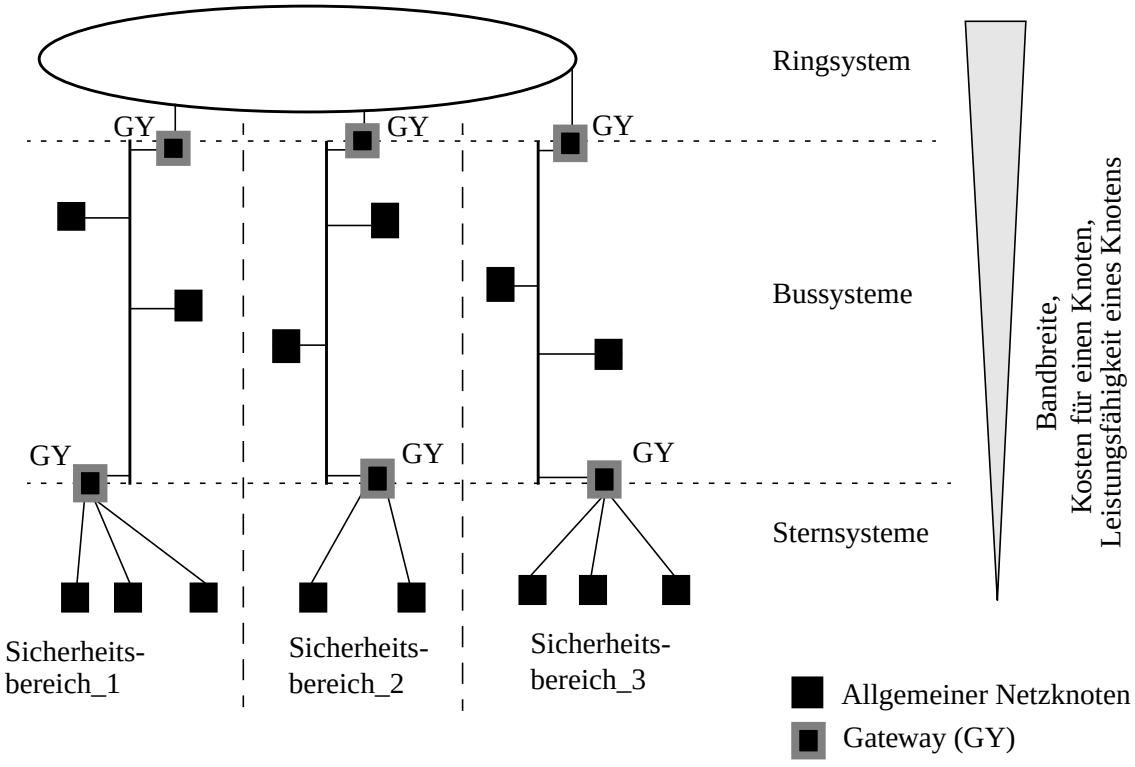


Bild 2-4: Beispiel eines hierarchischen Systems mit drei Stufen

senkrechten, gestrichelten Linien.) Hohe Sicherheitsanforderungen („security“) bestehen an solchen Stellen, an denen Daten unter keinen Umständen von außen manipuliert werden dürfen. Hierunter fällt z.B. das Motormanagement oder zukünftige Drive-by-wire-Systeme. Die zugehörigen Übertragungsleitungen werden deshalb einerseits mechanisch schwer zugänglich und andererseits durch entsprechend sichere Gateways von den anderen Teilnetzen nur für autorisierte Anwender erreichbar gemacht.

Außer einer Topologiebetrachtung kann für ein derartiges System auch eine Betrachtung der logischen Architektur erfolgen. Auf das dargestellte System läßt sich z.B. einfach ein Client/Server-System abbilden. Dabei arbeiten die Knoten der beiden unteren Ebenen als Clients und lassen aufwendigere Berechnungen von den Servern des Ringsystems bearbeiten.

2.1.1.3 Adressierung, Verbindungskonzepte und Vermittlungsverfahren

Adressen werden verwendet, um Kommunikationsteilnehmer, aber auch Netzressourcen zu lokalisieren. Adressen müssen dabei von Namen unterschieden werden, die eine höhere Abstraktion darstellen. Namen bezeichnen eine bestimmte Ressource und die Adresse gibt Aufschluß darüber, wo sich diese Ressource befindet. (Die Analogie „Fernsprechteilnehmer entspricht Name“ und „Telefonnummer entspricht Adresse“ mag dies verdeutlichen.) Die bereits vorgestellte Verkehrslenkung sorgt dann dafür, daß die transportierte Information zum richtigen Ort gelangt.

Aus der Fülle der Adressierungsarten [146, 96] sollen hier nur zwei Kategorien kurz aufgeführt werden, da sie für die betrachteten lokalen Netze von Interesse sind: die Stationsadressierung und die Botschaftsadressierung.

- Bei der *Stationsadressierung* erhält ein Knoten im Netz eine eindeutige Adresse, unter der er identifiziert werden kann, bzw. Botschaften zu ihm gelenkt werden können. Dies ist die „traditionelle“, auch im OSI-Modell beschriebene Methode, die in den meisten Netzen Verwendung findet. Ein Datenrahmen enthält in seinem Kopf dann z.B. die Adresse des Zielknotens und die Adresse des Absenders.
- Bei der *Botschaftsadressierung* wird eine Botschaft durch eine Kennung (Identifier) eindeutig gekennzeichnet. Knoten, die diese Botschaft empfangen, müssen anhand dieses Identifiers entscheiden, was mit der Botschaft geschehen soll (ignorieren, übernehmen, an ein anderes Teilnetz weitergeben, etc.). Der Vorteil dieser Adressierungsart besteht darin, daß 1:n-Beziehungen sehr einfach realisiert werden können und daß zusätzliche Geräte in ein bestehendes Netz ohne Änderung der Software der anderen Teilnehmer (zumindest, wenn die neue Station nur „mithört“) eingefügt werden können. Über die Lokalität einer Station, d.h. dem Ort, an dem sie sich befindet, kann bei dieser Adressierungsart zunächst nichts erfahren werden.

Kommunikationsbeziehungen lassen sich in zwei grundsätzlich verschiedene *Verbindungskonzepte* untergliedern. Die verbindungsorientierte Kommunikation (connection oriented, CO) braucht vor dem Austausch der Nutzinformation einen expliziten Verbindungsaufbau, in dem u.a. die Dienstgüteparameter ausgehandelt werden können. Ist die Beziehung beendet und sind alle Daten ausgetauscht, so wird die Verbindung wieder explizit in einer Verbindungsabbauphase beendet. Der Vorgang des Telefonierens, mit Teilnehmerwahl, Signalisierung, Abheben und Gespräch bis hin zum Auflegen durch einen Teilnehmer verdeutlichen diese Phasen. Eine verbindungslose Kommunikation (connectionless, CL) kommt ohne die Phasen des Verbin-

dungsauf- und -abbaus aus. Sie dient der Einzelblockkommunikation und stellt keine weiteren Protokollfunktionen zur Verfügung. Hier kann als Analogie die (gelbe) Paketpost dienen.

Bei den *Vermittlungsverfahren*, die zeitweilig eine Nachrichtenverbindung zwischen wechselnden Kommunikationspartnern herstellen, unterscheidet man zwei Grundversionen: die Paketvermittlung (packet switching, PS) und die Durchschaltevermittlung (circuit switching, CS) [119].

- Die Basis der *Paketvermittlung* ist die Aufteilung der zu übertragenden Information in einzelne Blöcke oder Pakete. Einem Informationsblock wird dann, außer den zu übertragenden (Teil-) Nutzdaten, auch Adreßinformation mitgegeben, so daß jeder Knoten, der den Informationsblock empfängt, entscheiden kann, ob er der Adressat dieses Informationsblocks ist, oder ob er den Informationsblock geeignet weitergeben muß. Es wird also im jeweiligen Knoten über die Weitervermittlung entschieden. Ist eine Teilstrecke belegt, so muß der Informationsblock im aktuellen Knoten zwischengespeichert werden.
- Die *Durchschaltevermittlung* wird bisher in den meisten Fällen zur Realisierung von Echtzeitdiensten verwendet; klassischerweise zur Übertragung von Sprache im Telefonsystem. Dabei wird den beteiligten Kommunikationsteilnehmern für die Dauer der Verbindung ein unmittelbarer Übertragungsweg zur Verfügung gestellt, unabhängig davon, ob Information übertragen wird oder nicht. Die Durchschaltevermittlung arbeitet immer verbindungsorientiert, so daß dem Dienst ein Kanal mit fester Übertragungsrate exklusiv zur Verfügung steht.

Kommunikationssysteme bilden immer eine Kombination der angesprochenen Verbindungskonzepte und Vermittlungsverfahren. So existieren etwa verbindungsorientierte Paketvermittlungsverfahren wie ATM oder X.25 und ebenso verbindungslose Paketvermittlungssysteme, wie sie hauptsächlich in den in dieser Arbeit betrachteten LANs oder auch beim User Datagram Protocol (UDP) der TCP/IP-Protokollfamilie vorkommen. Schließlich gibt es noch verbindungsorientierte Systeme mit Durchschaltevermittlung, wie z.B. das ISDN. Verbindungslose Systeme mit Durchschaltevermittlung existieren nicht.

2.1.2 Der physikalische Übertragungskanal

Der physikalische Übertragungskanal legt laut Basisreferenzmodell die elektrischen und mechanischen Eigenschaften des Kommunikationssystems, d.h. z.B. auch die Beschaffenheit von Steckverbindern, fest. Außerdem wird hier beschrieben, mittels welcher Codierung einzelne Bits auf dem Medium repräsentiert werden.

2.1.2.1 Merkmale und Anforderungen

Der physikalische Übertragungskanal in Fahrzeugen ist wesentlich von drei Randbedingungen geprägt:

- Möglichst niedrige Kosten für das Kabelmaterial: einfaches Material, wie beispielsweise nichtgeschirmte verdrillte Zweidrahtleitungen (unshielded twisted pair, UTP) und kurze Längen.
- Einfache Montage der Leitungen mit einer Minimalzahl an fehler- und störungsträchtigen Steckverbindungen.
- Hohe elektromagnetische Verträglichkeit (EMV). Die EMV ist in zweierlei Hinsicht zu

berücksichtigen. Zum einen kann es zu Störungen, bedingt durch das Abstrahlen der Kommunikationsleitungen selbst, kommen, zum anderen können externe Einflüsse störend auf das Übertragungsmedium wirken und so Fehlfunktionen verursachen.

Das Ziel ist somit die Erreichung maximaler Ausfall- und Störsicherheit bei minimalem Preis.

2.1.2.2 Die Bitcodierung

Bei den meisten Protokollen findet auf Schicht 1 eine Codierung der logischen Bits in physikalische Signale statt, d.h. in die Signale, die tatsächlich mit bestimmten physikalischen Eigenschaften über das Medium übertragen werden. Insbesondere bei den in Abschnitt 2.3.1 vorgestellten Verfahren finden die folgenden Codierungsarten oft Verwendung: das „Non Return to Zero Inverted“ (NRZI)-Verfahren, die Manchester-Codierung sowie die Pulsweitenmodulation PWM. Bild 2-5 gibt einen Überblick über die Codierungsarten.

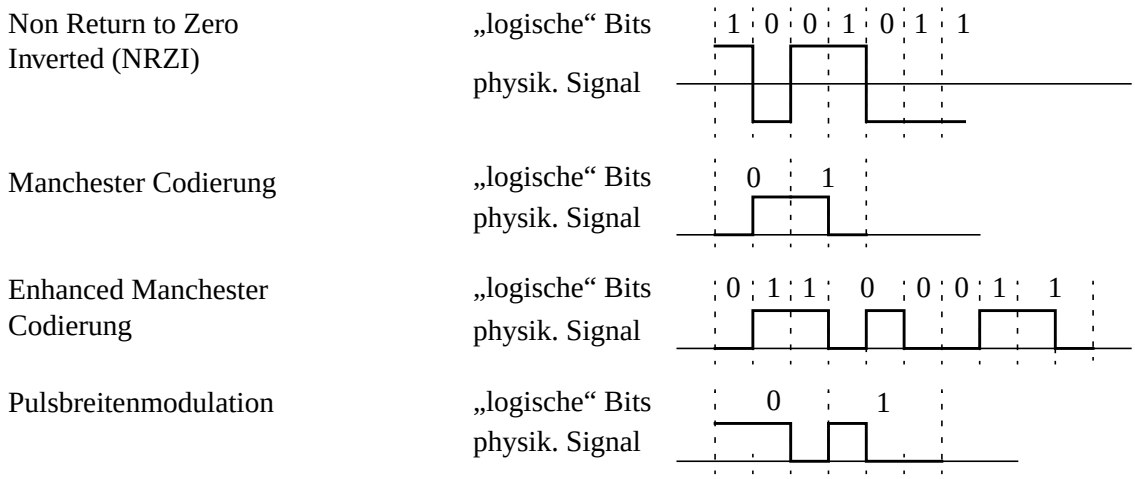


Bild 2-5: Codierungsarten der Bitübertragungsschicht

NRZI-codierte logische Bits werden so in physikalische Signale umgesetzt, daß bei jeder logischen „0“ ein Flankenwechsel stattfindet bzw. bei einer „1“ der Signalpegel gehalten wird. Die nichtinvertierte Version (NRZ) arbeitet genau umgekehrt. Bei der Manchester-Codierung werden je zwei physikalische Signalzustände für die Codierung eines logischen Bits verwendet. Die Enhanced-Manchester-Codierung optimiert das Verhältnis zwischen logischen Bits und physikalischen Signalzuständen dahingehend, daß nur jedes vierte Bit Manchester-codiert wird. Die Schrittgeschwindigkeit der Manchester-Codierung ist um 100% höher als die der NRZ-Codierung. Bei der Pulsbreitenmodulation (PWM) wird jedes logische Bit in drei physikalische Signale codiert, wobei das erste und das dritte physikalische Signal immer gleich sind. Die Schrittgeschwindigkeit ist dreimal so hoch wie bei den NRZ-Verfahren.

2.1.2.3 Verwendete physikalische Medien

Für die Verwendung im Fahrzeug kommt eine Reihe von physikalischen Medien in Frage, die hier kurz vorgestellt werden sollen.

Verdrillte Zweidrahtleitungen (twisted pair) werden heute bevorzugt zur Verknüpfung der Komponenten und zur Realisierung einer Bustopologie eingesetzt. Sie bestehen in der Regel

aus zwei gegenseitig verdrillten, isolierten Kupferleitungen. Dank dieser Verdrillung heben sich die in zwei benachbarten „Schleifen“ von äußeren Feldern induzierten Störspannungen praktisch vollständig auf. Damit diese Störunterdrückung groß genug wird, darf die Verdrillung nicht zu weitmaschig sein; ca. 30 Maschen pro Meter sind eine gängige Größe. Gegen kapazitive Störungen sind diese Kabel meist noch durch eine zusätzliche Abschirmung geschützt. Die Dämpfung spielt bei den für Fahrzeuge notwendigen Kabellängen keine so große Rolle.

Koaxialkabel, welche aus einem Mittelleiter und einem äußeren Schutzleiter aufgebaut sind, die wiederum durch ein Dielektrikum getrennt sind, werden aufgrund des höheren Preises selten eingesetzt. Die Geräuschimmunität ist für diese Kabel, bedingt durch den Aufbau, deutlich besser als für die Zweidrahtleitung.

Lichtwellenleiter aus Kunststoff oder Glas haben keine EMV-Probleme, werden aber aufgrund der teuren Technik noch nicht so häufig eingesetzt. Der höhere Preis gegenüber der „twisted pair“-Lösung resultiert dabei nicht nur aus den Kabelpreisen, die mit Koaxialkabeln vergleichbar sind, sondern auch aus den Steckern und den opto-elektronischen Komponenten innerhalb der Kommunikationsknoten. Außerdem bietet die Verlegung auf engstem Raum noch Probleme, da die Radien bei der Verlegung der Kabel nicht beliebig klein sein können, ohne die Eigenschaften des Kabels nachteilig zu verändern. Durch die immer stärker fortschreitende Integration der Komponenten dürften die Kostennachteile der Lichtwellenleiter in Zukunft jedoch eine immer geringere Rolle spielen und deren Einsatz mehr und mehr in den Vordergrund rücken.

Kommunikation über *Infrarot*, d.h. statt einer leitungsgebundenen Übertragung wird moduliertes Licht im Infrarotbereich über die Luftschnittstelle übertragen, spielt im Moment auf dem Markt noch keine große Rolle. Derartige Systeme werden allerdings in zunehmendem Maße für die Vernetzung von Komponenten im Innenraum von Fahrzeugen von Interesse sein. Denkbar sind hier z.B. persönliche digitale Assistenten (PDA), mit deren Hilfe auch auf externe Netze zugegriffen werden kann. (Siehe dazu auch Abschnitt 2.4.)

2.1.3 Medienzuteilungs- und -zugriffsverfahren

Ressourcen für die Kommunikation stehen in der Regel nicht in beliebigem Umfang für alle Kommunikationsteilnehmer zur Verfügung, die diese anfordern. Für die Medienzuteilung müssen deshalb Verfahren verwendet werden, die den Zugang auf das Medium beispielsweise über der Zeit verteilen (Time Division Multiplexing, TDM) oder das Medium auf andere Weise aufteilen. Dies kann durch verschiedene Frequenzen oder Wellenlängen (Frequency Division Multiplexing, FDM, und Wavelength Division Multiplexing, WDM) geschehen, oder indem verschiedene Codes zur Codierung der Information verwendet werden (Code Division Multiplexing, CDM). Die vierte Form ist der Raummultiplex (Space Division Multiplexing, SDM) wie er z.B. bei modernen Richtantennen zur Mehrfachnutzung von Frequenzen durch räumliche Trennung der Sendebereiche angewendet oder durch parallele Verlegung mehrerer Leitungen realisiert wird. Allerdings ist es gerade diese letzte Multiplexform die, aufgrund der hohen Kosten für die Verkabelung, durch andere Multiplex-Methoden ersetzt werden soll.

Für die in dieser Arbeit betrachteten automotiven Systeme sind zur Zeit lediglich die TDM-Systeme von Interesse, da die anderen Systeme u.a. einen deutlich höheren Schaltungsaufwand benötigen und deshalb aus Kostengründen ausscheiden. Zeitmultiplex kann durch unterschiedliche Medienzugriffsverfahren, z.B. im direkten Wettbewerb (contention) erreicht werden. Eine weitere Möglichkeit ist die vorberechnete oder zur Laufzeit erfolgte Zuteilung von Zeit-

abschnitten (time slots). Einige dieser Verfahren, insbesondere die Zugangsregelung mittels bitweiser Arbitrierung über Prioritäten, werden in Abschnitt 2.3.1 und ihre analytische Betrachtung in Kapitel 3 detailliert vorgestellt.

2.1.4 Architekturmerkmale und Realisierungen höherer Schichten

Sollen offene Systeme entworfen werden, so müssen insbesondere die höheren Schichten des Kommunikationsprotokollstapels und die darauf aufsetzenden Applikationen entsprechende Eigenschaften aufweisen. Hierbei zeigt sich auch ein langsam voranschreitender Umbruch in der Realisierung der Applikationen. Wo früher proprietäre Lösungen, meist nicht nach strukturierten oder gar objektorientierten Methoden, implementiert wurden, setzen sich heute zu Recht immer mehr diese zukunftsträchtigen Paradigmen durch. Erste Vorschläge, die in diese Richtung zielten, wurden dabei bereits in der PROMETHEUS[1]-Arbeitsgruppe PRO-COM unternommen (vgl. [43] und [103]).

2.1.4.1 Anforderungen

Die Anforderungen an die höheren Schichten sind im wesentlichen wiederum vom automotiven Umfeld getrieben:

- kostengünstige Lösungen
- skalierbare Systeme, die auf unterschiedlichsten Hardware-Plattformen einsetzbar sind
- weitgehende Transparenz der Kommunikationsmechanismen für den Applikationsprogrammierer (der nicht zum Kommunikationsexperten werden soll)
- Unterstützung aller gängigen (Hardware-)Schnittstellen nach außen
- Unterstützung unterschiedlicher logischer Betriebsmodelle (z.B. Client/Server-Betrieb)

Diese Anforderungen seien hier der Vollständigkeit wegen aufgeführt, können jedoch im Rahmen dieser Arbeit nur kurz anhand eines Realisierungsvorschlags erörtert werden. Einzig die Eigenschaften eines Kommunikationssubsystems sollen intensiver betrachtet werden, da dieses Einfluß auf die, noch zu untersuchenden, zeitlichen Eigenschaften des Steuergerätenetzes hat.

2.1.4.2 Realisierung höherer Schichtfunktionen für automotive Systeme

Führt man die Wiederverwertbarkeit von Software als weitere Anforderung auf (auch wenn diese aufgrund des notwendigen Entwicklungsaufwands in Relation zu den produzierten Stückzahlen im Moment noch keine große Rolle spielt), dann machen Architekturen Sinn, wie sie aus anderen Bereichen der Kommunikationstechnik bereits bekannt sind [145, 96]. Ein grundlegendes Modell für ein solches System zeigt Bild 2-6. Das Kommunikationssystem ist dabei Teil des Betriebssystems und bietet seine Dienste über ein API den Applikationen an. Diese verwenden die Kommunikationsdienste innerhalb ihres Funktionsablaufs, ohne sich dabei mit Fragen der Verkehrslenkung, der Vermittlungsverfahren oder der Adressierung auseinandersetzen zu müssen. Auch Synchronisationseigenschaften und die Garantie der Datenkonsistenz können so von den Anwendungen ferngehalten werden. Einzig die Festlegung

1. **Pro**gra**m**m für ein **e**uropäisches **T**ransportsystem mit **h**öchster **E**ffizienz und bisher **u**nerreichter **S**icherheit

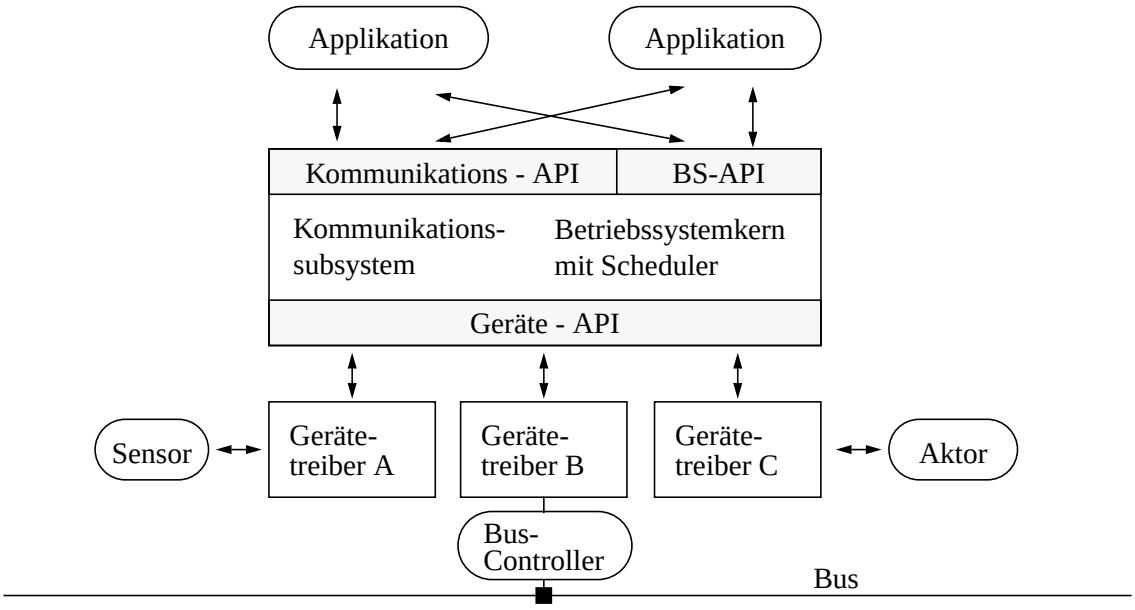


Bild 2-6: Allgemeine Software-Systemarchitektur für Steuergeräte

bestimmter Dienstgüteparameter für eine Kommunikationsverbindung könnte in den Aufgabenbereich einer Applikation fallen, sofern Verbindungen dynamisch aufgebaut werden können. Allerdings ist der Großteil der in einem automotiven System anfallenden Kommunikationsanforderungen bereits vor der Inbetriebnahme des Systems bekannt. Abhängig von der Ausstattung des Systems kennt man die Informationsquellen und -senken und kann deshalb die zur Realisierung der Kommunikation notwendigen Software-Strukturen bereits vorher bereitstellen und sie, wiederum kostengünstig, in den nichtflüchtigen ROM-Speicher eines Steuergeräts legen.

Auch für die angeschlossenen Geräte (Sensoren und Aktoren) benötigt man wiederum eine Schnittstelle des Betriebssystems für die zur Ansteuerung dieser Geräte verwendeten Geräte-Treiber. Damit erreicht man an dieser Stelle die Flexibilität, beliebige Geräte verwenden und in das System integrieren zu können, wenn der entsprechende Treiber bereitgestellt wird.

Verwendet man einen objektorientierten Ansatz zur Realisierung des Kommunikationssubsystems [144], so läßt es sich mit Hilfe von grundlegenden „atomaren“ Funktionsblöcken, die im Sinne der Objektorientierung von einer gemeinsamen Basis abgeleitet werden, dekomponieren. Ermittelt man diese Funktionsblöcke anhand der in einem derartigen System anfallenden Kommunikationstypen, dann werden im wesentlichen die folgenden Grundfunktionen (hier als „Manager“ bezeichnet) benötigt:

- ein Speicher-Manager, der, je nach Konfiguration, einen oder mehrere Pufferplätze für Nachrichten zur Verfügung stellt,
- ein Fork-Manager, der einen Nachrichtenkanal auf mehrere Kanäle aufspalten kann und
- ein Paket-Manager, der mehrere Nachrichtenpakete auf einen Transportkanal multiplexen und am Bestimmungsort wieder demultiplexen kann.

Instanzen dieser Manager werden über eine gemeinsame Schnittstelle, die aus der objektorientierten Modellierung resultiert, wie Bausteine zu den gewünschten Verbindungsarten 1:1 (unicast), 1:m (group cast) und m:n mit entsprechenden Speichereigenschaften zusammengesetzt.

2.2 Betriebssysteme

Betriebssysteme für Steuergeräte sind in der Regel Multitasking-fähige Echtzeitsysteme. Sie basieren deshalb oft auf den in dieser Arbeit betrachteten Prioritätenschedulern (siehe dazu Kapitel 3). Den Anwendungen des Systems werden dabei Prioritäten zugeordnet, anhand derer entschieden wird, welche Anforderung als nächste bearbeitet wird. Beispielhaft soll ein einfaches prioritätsgesteuertes Betriebssystem vorgestellt werden [147].

2.2.1 Anforderungen an ein automotives Betriebssystem

Die Anforderungen an das Betriebssystem sollen in drei Prioritätsgruppen eingeteilt werden:

- Anforderungen auf der Basis von externen und internen Programmunterbrechungen (Interrupts), denen die höchste Priorität zugeordnet wird. Hierunter fallen Unterbrechungen durch Zeitgeber (timer) oder Meldungen der Buscontroller beim Empfang einer neuen Botschaft.
- Anforderungen des Kommunikationssubsystems inklusive der Bedienung der Gerätetreiber, wie sie im vorhergehenden Abschnitt vorgestellt wurden. Diesen Anforderungen wird eine mittlere Priorität zugewiesen.
- Anforderungen der Anwendungen (tasks), die die eigentlichen Programme darstellen, welche die niederen Prioritäten zugeteilt bekommen.

Typischerweise sind die Prioritäten für die beiden ersten Typen fest vorgegeben, während die Prioritäten für die Anwendungen von deren (zeitlichen) Randbedingungen abhängen.

2.2.2 Realisierung

Eine Software-Datenstruktur für einen zu implementierenden Scheduler kann als Vektor mit verketteten Listen realisiert werden. Dabei sind den einzelnen Listen feste, im System verfügbare Prioritäten zugeordnet. Die Bearbeitung der Anforderungen, die beim Eintreffen in die entsprechende Liste geschrieben werden, geschieht schließlich (wie in Bild 2-7 zu sehen) in der Reihenfolge der Prioritäten und innerhalb einer Prioritätsklasse in der Reihenfolge der Anforderungen in der Liste. Treten während der Abarbeitung von Anforderungen wieder solche mit höherer Priorität in das System ein, dann werden diese als nächste bearbeitet, sobald der Scheduler wieder aktiviert wird. Die Prioritätsstufe 1 wurde in der Darstellung ausgespart, da sie für den Scheduler reserviert ist. Wird der Scheduler allerdings als sogenannter „tick scheduler“ realisiert, so reicht die durch einen Zeitgeber-Interrupt initiierte Unterbrechungsbehandlung (Interrupt Service Routine) aus, um die höchstpriore Anforderung zu ermitteln und zu starten. Man muß somit keine spezielle Priorität für den Scheduler reservieren.

Die Möglichkeiten eines Betriebssystems hängen indirekt auch von dem zur Verfügung stehenden Speicher (RAM und ROM) ab, so daß viele moderne Betriebssysteme skalierbar ausgelegt sind. Ausgehend von der Grundfunktionalität des Schedulers, die auf jedem Zielsystem zur Verfügung steht, gibt es unterschiedliche funktionale Ausbaustufen für die einzelnen Zielplatt-

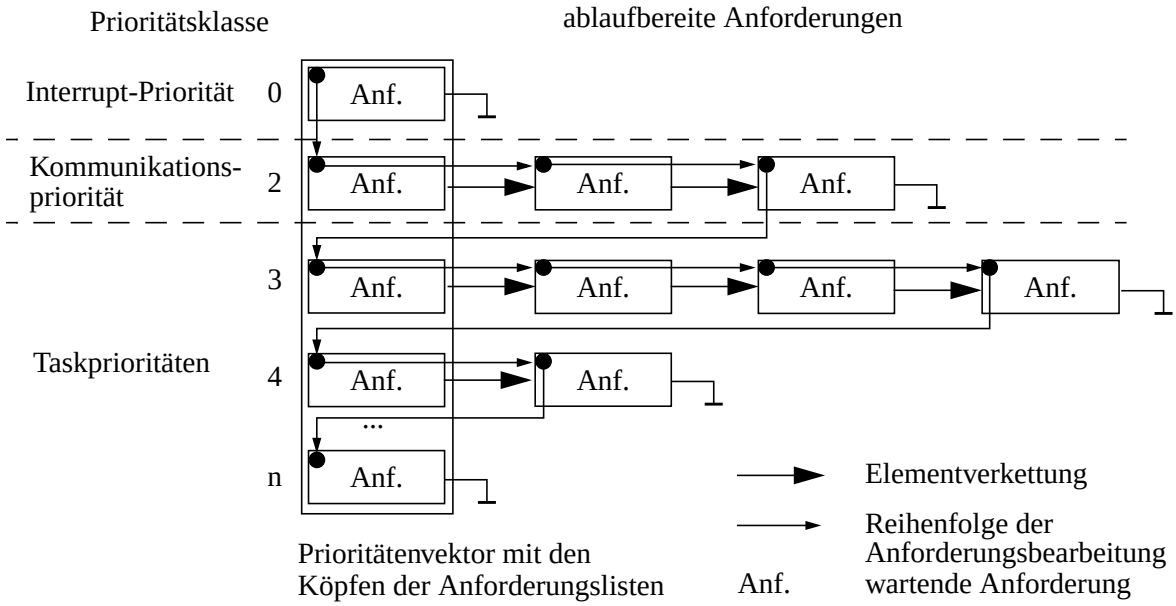


Bild 2-7: Datenstruktur der ablaufbereiten Anforderungen eines Prioritäten-Betriebssystems

formen. Eine dieser Ausbaustufen ist auch die Bereitstellung eines Kommunikationssubsystems, wie es in Abschnitt 2.1.4 vorgestellt wurde.

2.3 Protokolle und Standards

In diesem Abschnitt erfolgt die Darstellung von existierenden LAN-Konzepten, wie sie in Fahrzeugen Verwendung finden. Dazu werden zunächst die busbasierten Systeme mit prioritätsgesteuerter, bitweiser Arbitrierung betrachtet, da sie auch Gegenstand der nachfolgenden Analyse und Optimierung sein werden. Außerdem werden weitere LAN-Strukturen vorgestellt, wie sie für die Zukunft denkbar sind oder bereits in anderem Zusammenhang in Fahrzeugen eingesetzt werden.

2.3.1 Medienzugriffsverfahren mit bitweiser Arbitrierung

Die in der Kommunikationstechnik übliche Unterscheidung zwischen Hochgeschwindigkeitssystemen und Systemen mit geringerer Kapazität ordnet alle Busse mit bitweiser Arbitrierung den langsamen Systemen zu. Für den charakterisierenden a-Faktor[2] dieser Systeme gilt systembedingt immer $a < 1$. Die bitweise Arbitrierung verlangt, daß ein bestimmter Zustand des Mediums (bit) für alle Stationen gleichzeitig erkennbar ist. Damit ist auch die maximale Ausdehnung solcher Systeme, abhängig von der gewünschten Bandbreite, durch die Wellenausbreitungsgeschwindgkeit auf dem Medium festgelegt.

Die Mehrzahl der heute für automotive Anwendungen eingesetzten Bussysteme (CAN, VAN, SCP, ABUS) arbeitet mit Botschaftsadressierung und einem darauf aufbauenden prioritätsgesteuerten Medienzugriff. Dabei werden nicht die einzelnen Stationen (Knoten) eines Netzes adressiert, sondern jede Botschaft erhält eine eindeutige Kennung, den sog. „Identifier“. Befindet sich eine Botschaft auf dem gemeinsamen Medium, dann entscheidet jede Station durch

2. Der a-Faktor berechnet sich aus der Medienlänge l, der Übertragungsrate B, der relativen Ausbreitungsgeschwindigkeit v_m sowie der mittleren Paketlänge L [98].

Lesen des Identifiers darüber, ob sie sich für diese Botschaft interessiert oder nicht. Der Identifier, der sich am Anfang eines Datenrahmens befindet, wird zusätzlich zur Regelung des Buszugriffs verwendet. Grundlage dieses Verfahrens ist die Unterscheidung eines dominanten („d“) und eines rezessiven („r“) Zustands auf dem Übertragungskanal (Bus). Durch bitweise Arbitrierung erhält genau der Sender Zugang zum Medium, dessen Identifier die höchste Prio-

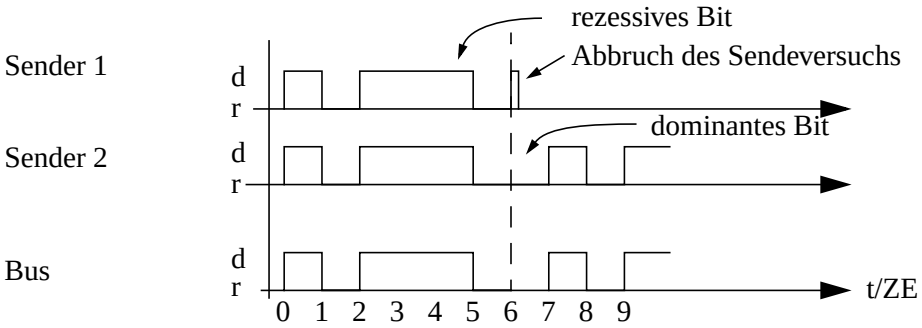


Bild 2-8: Bitweise Arbitrierung in einem dominant-rezessiven Kanal

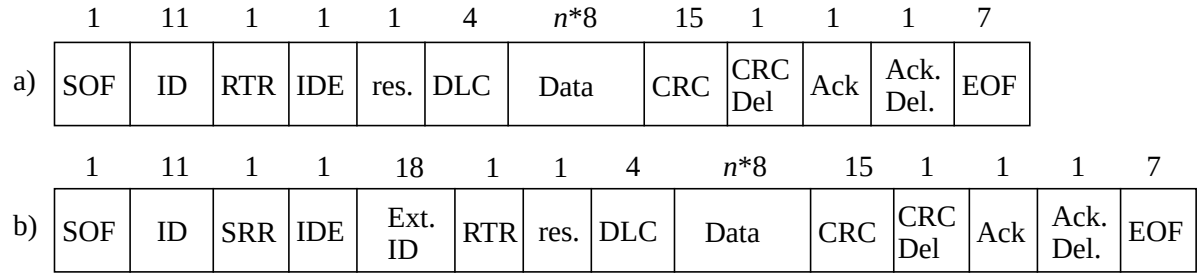
rität hat (siehe Bild 2-8): Zum Zeitpunkt $t = 0$ wollen zwei Sender den Bus belegen. Solange beide Sender das gleiche Signal senden, gibt es keine Probleme. Detektiert ein Sender ein „dominantes“ Bit auf dem Bus, während er ein „rezessives“ Bit senden will (Sender 1 zum Zeitpunkt $t = 6$), dann hat er die Arbitrierung verloren und zieht sich vom Medium zurück. Voraussetzung für das Funktionieren dieses Protokolls ist das ständige Überwachen des Mediums („Monitoring“). Dieser Protokolltyp wird mit CSMA/CA bezeichnet, wobei CA für „Collision Avoidance“ steht. An anderer Stelle [127] wird auch von „Collision Resolution“ gesprochen. Der Vorteil dieser Arbitrierungsmethode besteht darin, daß bei den zu einem bestimmten Zeitpunkt um den Medienzugriff konkurrierenden Botschaften diejenige mit der jeweils höchsten Priorität gewinnt und den Zugriff erlangt. Für ihre Übertragung geht keine (Warte-) Zeit verloren.

Bedingt durch die Botschaftspriorisierung kann jeder Identifier nur genau einmal vergeben werden, um Inkonsistenzen zu vermeiden. Bei allen Systemen kann eine Botschaft, wenn sie sich nach erfolgreicher Arbitrierung auf dem Medium befindet, nicht mehr unterbrochen werden. Entsprechend der in Kapitel 3 folgenden Klassifikation handelt es sich dabei um nichtunterbrechende Prioritätensysteme.

2.3.1.1 Controller Area Network (CAN)

Das CAN-Protokoll wurde 1985 in einer Kooperation der Firmen Bosch (Entwicklung) und Intel (Implementierung) erstellt [85] und liegt inzwischen als Norm vor [72]. Es erlaubt in seiner Standardform Identifier mit einer Länge von 11 bit. Zur Botschaftskennung und somit auch zur Arbitrierung stehen dadurch 2048 unterschiedliche Botschaften zur Verfügung. In der erweiterten Version können 29 bit große Identifier verwendet werden. Bild 2-9 zeigt den Aufbau der CAN-Rahmen.

Stellt ein Sender nach dem Übertragen des Identifiers noch Differenzen zwischen dem eigenen, gesendeten Signal und dem Signal auf dem Kanal fest, so wird dies als Fehler interpretiert und ein entsprechendes Fehlerflag gesendet, das aus sechs dominanten Bits besteht. Dies wird von den anderen Stationen ebenfalls als Fehlerfall entdeckt, da die sechs gleichen Bits die



SOF	Start of frame	res.	reserviertes Bit
ID	Identifier	Data	Nutzdatenfeld
RTR	Remote Transmission Request	CRC	cyclic redundancy check
IDE	ID Extension Switch	CRC Del.	CRC Delimiter
DLC	Data Length Code	Ack	Acknowledgement
SRR	Substitute Remote Requ.	Ack.Del.	Ack Delimiter
Ext. ID	ID Extension	EOF	End of Frame

Bild 2-9: CAN-Rahmen.a) Standard Format, b) Extended Format. Im Extended Format stehen 29 bit lange Identifier zur Verfügung. Die Zahlen über den Feldern geben deren Länge in bit an.

„Stopfbit“-Regel (bit stuffing) verletzen. Laut dieser Regel wird nach fünf gleichen Bits automatisch ein Bit mit umgekehrter Polarität in den Datenstrom eingefügt. Dies dient im Normalfall zur Erhaltung der Synchronisation der Stationen und wird außerdem, wie gezeigt, zur Kennzeichnung von Fehlerfällen ausgenutzt. Im Gegensatz zu den anderen vorgestellten Protokollen dieses Kapitels, führt CAN eine aktive Fehlersignalisierung durch. D.h., daß auch Stationen, die nicht Empfänger einer Botschaft sind, Fehlerrahmen senden, sofern sie einen Fehler erkannt haben. Damit wird die Botschaft zerstört und Dateninkonsistenzen können vermieden werden.

Außer der einfachen Versendung von Rahmen erlaubt CAN auch das Anfordern von Rahmen mittels „Remote Transmit Request“ (RTR). Dazu sendet eine Station einen Rahmen mit gesetztem RTR-Bit und ohne Nutzdaten. Die Antwort auf einen RTR-Rahmen trägt wiederum denselben Identifier wie die Anfrage. Dies bildet die einzige Ausnahme der eindeutigen Identifier.

Zur Erkennung von Bitfehlern (nach der Entfernung der Stopfbits) verfügt CAN über ein 15-bit CRC (zyklischer Binärcode), mit dem bis zu fünf Fehler in einer Botschaft und Fehlerbüschel bis zu einer Länge von fünfzehn Bit entdeckt werden können. Außerdem können ungerade Fehleranzahlen erkannt werden.

Die Codierung der logischen Bits in physikalische Signale erfolgt per NRZ-Codierung. Dadurch wird die Schrittgeschwindigkeit auf dem Medium um maximal 20% größer, wenn das Bitstuffing mitberücksichtigt wird. Über das zu verwendende physikalische Medium macht der Standard keine Angaben. Typische Realsierungen verwenden verdrillte Zweidrahtleitungen. Auch optische Übertragungskanäle mit Kunststoff-Lichtwellenleitern wurden bereits realisiert [101].

Das CAN-Protokoll hat sich inzwischen als Industriestandard durchgesetzt. Seine Anwendungsfelder sind nicht auf den automotiven Bereich beschränkt geblieben, sondern auch die Automatisierungstechnik, die Gebäudeleittechnik und die Medizintechnik setzen CAN auf-

grund seiner hohen Übertragungssicherheit ein. Zudem tragen die niedrigen Kosten (< 10 USD pro Knoten bei 8-bit Mikrocontrollern [88]) zum Erfolg dieses Protokolls bei.

2.3.1.2 Vehicle Area Network (VAN)

Das VAN-Protokoll [73] wurde in Anlehnung an CAN von den staatlichen, französischen Automobilfirmen entwickelt. Hier stehen Identifier mit zwölf Bit zur Verfügung. Allerdings endet die Arbitrierung nicht am Ende des Identifiers, sondern wird bis einschließlich des CRC-Feldes fortgesetzt. Werden Abweichungen zwischen gesendetem Signal sowie auf dem Bus beobachtetem Signal festgestellt, dann zieht sich die Station zurück. Eine aktive Fehlermeldung wie bei CAN findet nicht statt. Bild 2-10 zeigt den Aufbau eines VAN-Rahmens.

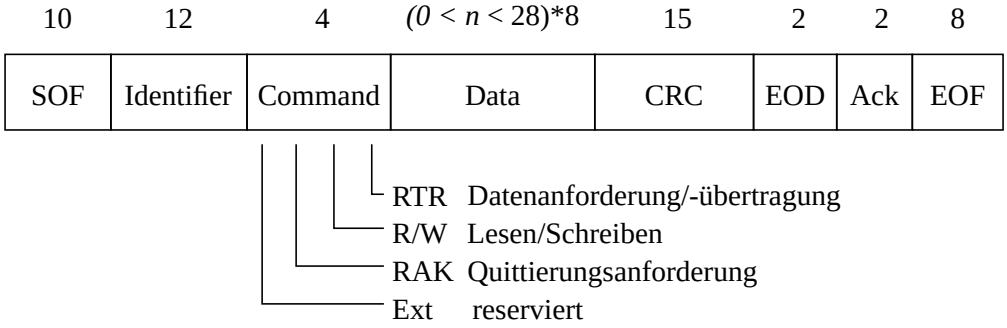


Bild 2-10: VAN-Rahmen. Die Ziffern über den Feldern geben die Ausdehnung in bit an.

Das VAN-Protokoll ermöglicht eine aktive Bestätigung des erfolgreich empfangenen Rahmens, noch während der Rahmen gesendet wird (inframe response). Dazu vertauschen Sender und Empfänger kurzzeitig für die Zeit von zwei Bit (Acknowledge) die Rollen. Diese quittierte Übertragung kann natürlich nur bei Punkt-zu-Punkt-Verbindungen sinnvoll eingesetzt werden.

Wie bei CAN gibt es, außer den normalen Datenrahmen, einen „Remote Transmit Request“, auf den als Antwort ein „Response“-Rahmen gesendet wird. Kann ein Knoten die per RTR angeforderten Daten sofort senden, so kann dies während des RTR bereits geschehen (inframe response). Dazu wird das rezessive Bit der anfordernden Station überschrieben, so daß diese sich vom Bus zurückzieht und schließlich die Daten übertragen werden können.

Die Leistungsfähigkeit des CRC entspricht weitgehend der des CAN. Zur Empfängersynchronisierung verwendet VAN eine „Enhanced Manchester“-Bitcodierung, womit eine um 25 % höhere Schrittgeschwindigkeit auf dem Medium notwendig wird. Jedes vierte logische Bit wird dabei Manchester-codiert, so daß die nötige Flankenhäufigkeit im Signal sichergestellt ist.

2.3.1.3 Standard Corporate Protocol (J1850)

Das J1850-Protokoll ist in [74] spezifiziert, weist aber in dieser Spezifikation einige Lücken auf. Deshalb soll das von Ford entwickelte Standard Corporate Protocol (SCP) [38] betrachtet werden, das auf J1850 basiert. Zur Priorisierung stehen 24 Bit, eingeteilt in drei logische Felder, zur Verfügung. In den ersten acht Bit, die wiederum logisch in zwei Nibble unterteilt sind, kann im ersten Nibble eine allgemeine Priorität („Prio“) vergeben werden. Danach folgen vier Bit („Type“), in denen der Rahmentyp spezifiziert wird. Im zweiten Feld wird das Ziel der

Nachricht festgelegt. Im letzten Feld wird schließlich die Senderadresse abgelegt. Damit ist die Zahl der Stationen in einem SCP-Netz logisch auf 256 begrenzt. Da die Arbitrierung, vergleichbar mit dem VAN-Protokoll, über den gesamten Rahmen geht, kann die Priorisierung/ Adressierung wie bei den bereits vorgestellten Protokollen betrachtet werden. Bild 2-11 zeigt

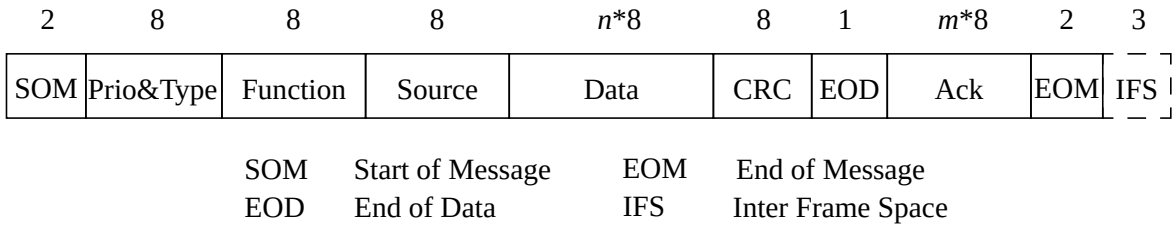


Bild 2-11: SCP-Rahmen

den Aufbau eines SCP-Rahmens. Für die Werte von m und n gilt: $0 \le n \le 7$, $1 \le m \le 7$ und $m + n \le 8$, so daß im maximalen Fall sieben Datenbytes verschickt werden können.

Die Fehlererkennungsmechanismen bei SCP entsprechen weitgehend jenen bei VAN, auch SCP erlaubt keine aktive Fehlersignalisierung. Allerdings kann der CRC des SCP-Protokolls keine Paritätsüberprüfung durchführen und Büschelfehler nur bis zu einer Länge von 8 bit erkennen.

SCP verwendet zur Codierung der logischen Bits in physikalische Signale die Pulsbreitenmodulation (PWM). Dabei wird jedes logische in drei physikalische Bitsignale umgewandelt, so daß eine um den Faktor drei höhere Schrittgeschwindigkeit notwendig ist. Zwischen zwei SCP-Rahmen muß eine 3-bit lange Pause (Inter Frame Space) gemacht werden.

2.3.1.4 Automobile Bitserielle Universal Schnittstelle (ABUS)

Der von VW eingesetzte ABUS [12] wird inzwischen nicht mehr weiterentwickelt und soll hier nur der Vollständigkeit wegen aufgeführt werden. Das Format eines Rahmens erlaubt die Übertragung zweier Datenbytes. Zur Arbitrierung steht ein 12 bit langer Identifier zur Verfügung, der außerdem eine logische Unterscheidung zwischen Steuer- und Datennachrichten über die Priorität des ersten Bits des Identifiers ermöglicht. Wie aus Bild 2-12 zu sehen ist, exi-

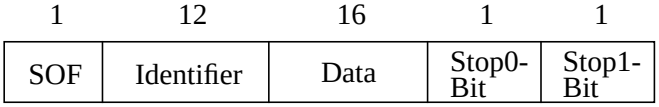


Bild 2-12: ABUS-Rahmen

stieren im ABUS-Protokoll nur rudimentäre Fehlersicherungsmechanismen. Tritt bei der Übertragung ein Fehler auf, dann wird die Polarität der beiden Stop-Bits umgedreht, worauf der Sender der Botschaft die Übertragung wiederholt. Auf die Berechnung eines CRC wird gänzlich verzichtet.

2.3.1.5 Vergleich der Protokolle

Einige der wichtigen Leistungsmerkmale der vorgestellten Protokolle sollen hier festgehalten werden. Die maximale Ausdehnung der Netze hängt, aufgrund der bitweisen Arbitrierung, von

Merkmal	CAN	VAN	SCP	ABUS
Datenfeldlänge pro Rahmen in Byte	8	28	7	2
Datenrate in kbit/s	1000	250	46,1	500
Hamming-Distanz *h* des CRC	5	3	3	-
Bits zur Codierung versch. Botschaften	11/29	12+	4+	12

Tabelle 2-1: Merkmale der Busse mit bitweiser Arbitrierung

der verwendeten Datenrate ab. Typische Entfernungen sind dabei 40 m bei 1 Mbit/s und 1000 m bei 125 kbit/s (Werte für CAN). Das „+“-Zeichen in der Zeile Bits zur Codierung zeigt an, daß die Arbitrierung über das Identifier-Feld hinaus fortgeführt wird. Zur Charakterisierung der Einsatzmöglichkeiten wurden umfangreiche Untersuchungen über das Bitfehlerverhalten dieser Protokolle am Institut in Zusammenarbeit mit einem Hersteller durchgeführt [39, 23], bei denen CAN in bezug auf die Bitfehlerwahrscheinlichkeit als das sicherste dieser Protokolle abschnitt.

2.3.2 Weitere LAN-Konzepte und Protokolle für den automotiven Bereich

In den folgenden Unterabschnitten sollen noch weitere, im automotiven Bereich eingesetzte, Kommunikationskonzepte vorgestellt werden. Dabei reichen die Vorschläge bzw. Standards von einfachen Protokollen (DIGITbus) bis hin zu vollständigen Umgebungen (Mars-TTP).

2.3.2.1 DIGITbus

Der DIGITbus der Firma ITT Automotive [55] soll als Low-cost-Lösung einfache (binäre) Steuersignale von Sensoren liefern oder Aktoren ebenso ansprechen. Er soll damit den „Teilnehmeranschlußbereich“ z.B. eines CAN-Systems realisieren, bei dem es aus Kostengründen und aus Rechenleistungsgründen keinen Sinn machen würde, jeden Sensor oder Aktor direkt mit einem CAN-Knoten auszurüsten.

Das Prinzip des DIGITbus entspricht der Idee der SLIO-Bausteine (Serial Linked Input Output), die intelligente Sensor- oder Aktorknoten darstellen und ohne teuren Mikrocontroller auskommen. Beim DIGITbus wird eine Aufteilung in Daten- und Steuernetz vollzogen, wie sie ebenso an anderer Stelle proklamiert wird [127], indem über den DIGITbus lediglich Steuerimpulse verschickt werden und man somit auch mit geringer Bandbreite für das System auskommt.

2.3.2.2 Domestic Digital Bus (D2B)

Für dieses Protokoll [24], das auch unter dem Namen „Digital Data Bus“ bekannt ist und ursprünglich von Philips entwickelt wurde, wird als Topologie für die Verknüpfung der Netzknoten überraschenderweise ein Ring verwendet [61]. Als physikalisches Medium für den

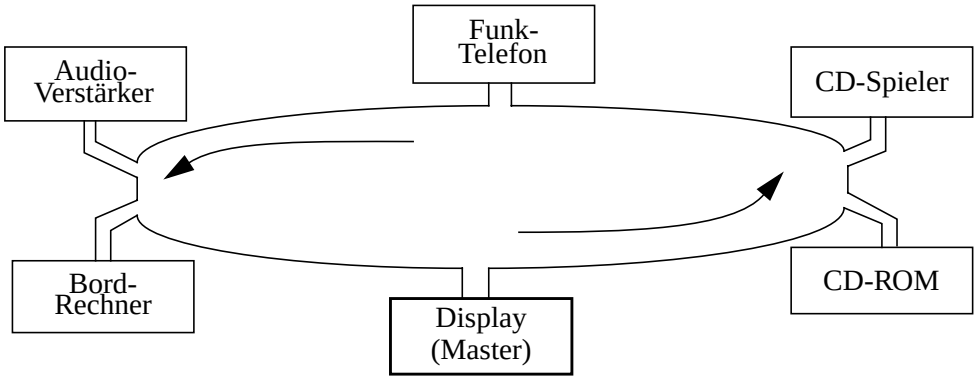


Bild 2-13: Typische D2B-Konfiguration

„Bus“ werden Kunststofflichtwellenleiter verwendet. Ziel des D2B ist die Vernetzung und Steuerung von Audio- und Multimediakomponenten (CD-Spieler, Telefon, Tuner, Audioverstärker etc.) im Fahrzeug. Dazu wurde eine Reihe von Befehlen zur Steuerung dieser Komponenten entwickelt und zwischen verschiedenen Herstellern der Geräte standardisiert.

Das Datenformat des Protokolls ist auf die Übertragung von Stereosignalen optimiert: Es erlaubt z.B. die Übertragung von drei 16-bit Stereokanälen in Echtzeit bei einer Übertragungsbandbreite von 4,9 Mbit/s. Dazu kann jeweils ein Teilrahmen für drei linke oder drei rechte Stereokanäle verwendet werden. Dies ist aber nicht zwingend nötig. Die Daten können viel-

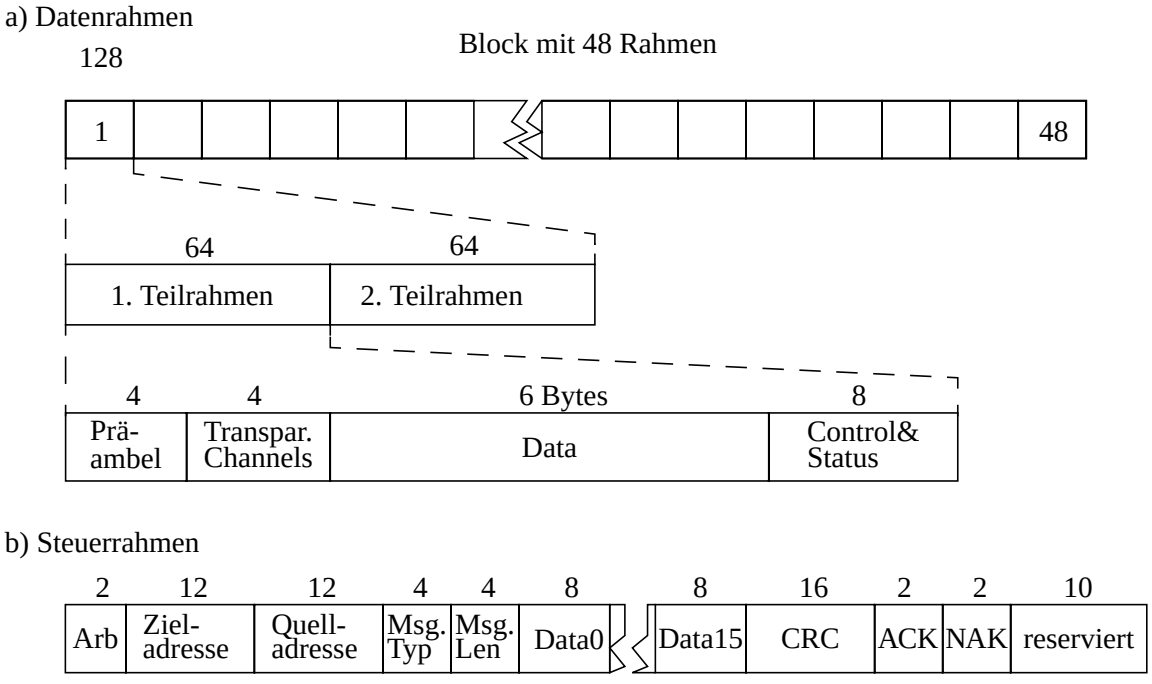


Bild 2-14: Rahmenformate des D2B. Die Ziffern über den Feldern geben die Feldlängen in bit an.

mehr frei in das Datensegment des Rahmens gelegt werden. Steuerinformation für die Komponenten wird über einen Block hinweg über mehrere Datenpakete verteilt übertragen. Jeweils 2 bit des Control- und Statusfeldes werden dazu verwendet. Bild 2-14 zeigt die Block- und Rahmenformate von D2B.

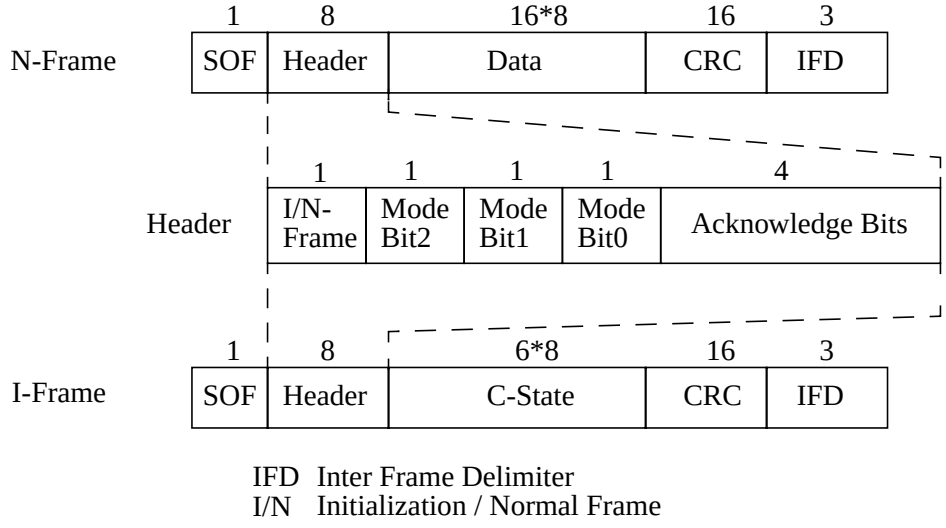
Über die transparenten Kanäle (Transparent Channels) können weitere unabhängige Kommunikationsverbindungen betrieben werden.

Der Medienzugriff wird mittels Reservierung bei der Master-Station realisiert, die anschließend den entsprechenden Rahmen zur Übertragung der Nutzdaten zuteilt.

2.3.2.3 Time Triggered Protocol (TTP)

Beim Time Triggered Protocol [92], das als Kommunikationsprotokoll für das fehlertolerante, echtzeitfähige Mars-System [91] entwickelt wurde, wird der Medienzugriff ebenfalls durch Zeitmultiplex erreicht. Es findet jedoch kein Wettbewerbsverfahren statt, sondern die Sendezeitpunkte werden vor Systemstart berechnet und das Wissen über diese Zeitpunkte steht jedem Kommunikationsteilnehmer zur Verfügung. Für einen synchronen Kommunikationsablauf ist ein globaler Systemtakt notwendig. Die Basis für eine Kommunikationsverbindung ist dabei die Annahme, daß sowohl Sender als auch Empfänger zu einem bestimmten Zeitpunkt davon ausgehen, daß sie eine gültige quasi durchschaltevermittelte Kommunikationsbeziehung haben. Im Controller, der das Protokoll abarbeitet, wird dazu der aktuelle Controller Zustand („C-State“) festgehalten. Der C-State enthält die globale Zeit, den Betriebsmodus und eine Gruppenzugehörigkeitsinformation (membership information). Diese Zugehörigkeitsinformation wird über ein Bitfeld codiert und jeder Station dabei ein bestimmtes Bit zugeordnet. Das Rahmenformat des TTP zeigt Bild 2-15.

Um eine höhere Fehlertoleranz des Systems zu erreichen, können einzelne Rahmen über getrennte, parallele Busse übertragen werden. Wird ein Rahmen empfangen, so erfolgt die Berechnung des CRC über die Felder des Headers, der Daten und über den im Controller gespeicherten lokalen C-State. Danach findet ein Vergleich des Ergebnisses mit dem empfan-



IFD Inter Frame Delimiter
I/N Initialization / Normal Frame

Bild 2-15: Rahmenformate des TTP. Ein Rahmen besteht aus drei Feldern: einem ein-Byte langen Header, einem Datenfeld mit bis zu 16 Byte und einem ein-Byte langen CRC.

genen CRC statt. Wird eine Differenz in einem der redundanten, empfangenen Rahmen festgestellt, dann wird der Knoten von der Liste der aktiven Teilnehmer gestrichen. Hat sich eine Station vom System zurückgezogen, kann sie später mit Hilfe von Initialisierungsrahmen versuchen, wieder integriert zu werden.

Um eine Garantie für die störungsfreie Funktion der Netze erzielen zu können, gehen Mars und TTP von einem Modell der logischen Kapselung der einzelnen Knoten aus. Damit soll vermieden werden, daß Verkehr der anderen Stationen durch eine fehlerhafte Station gestört wird, die unerwünschten Verkehr produziert („babbling idiot“), weil sie zu unerlaubten Zeitpunkten das Medium belegt. Das theoretische „fail-silent“-Modell, das dieser Eigenschaft zugrundegelegt wird, verlangt dazu, daß eine Station entweder korrekte oder gar keine Rahmen nach außen gibt.

Außer den Datenrahmen existieren überdies Initialisierungsrahmen (I-Message) für die Inbetriebnahme des Netzes. In deren Datenfeld wird der C-state des Senders transportiert. Die Berechnung des CRC beim Empfänger erfolgt hier auf „normale“ Weise. Über die I-Messages können auch Stationen nach der Reparatur wieder in das Netz aufgenommen werden. Dazu müssen aber I-Messages nicht nur beim Systemstart, sondern während der gesamten Laufzeit des Systems gesendet werden. Deshalb wird empfohlen, in bestimmten Zeitabständen I-Messages von verschiedenen Stationen generieren zu lassen. Über diese Notlösung, die einen fest reservierten Anteil der Bandbreite erfordert, läßt sich eine Art Netzmanagement realisieren und eine Synchronisation der Stationen erreichen. Dies stellt für die aktuellen Implementierungen noch große Probleme dar, da man momentan noch mit Bandbreiten von unter 20 kbit/s arbeitet! Ebenfalls schwierig ist die Integration von Quellen mit sehr unterschiedlichen zeitlichen Bedingungen, da die Zykluszeit nach der höchsten Frequenz ausgelegt werden muß und deshalb, wegen der notwendigen Reservierung für die niederfrequenten Messages, Bandbreite unnötig vergeben wird. Dies macht auch einen Hauptnachteil gegenüber ereignisgesteuerten Systemen aus.

TTP kann nicht isoliert, d.h. ohne eine Art Mars-System betrachtet werden, da nur hiermit die Anforderungen, die TTP an seine Umgebung stellt, erfüllt werden können. In diesem Kontext wird von *Firewalls* (Bild 2-16) gesprochen, d.h. Schnittstellen, an denen sich das jeweils auf

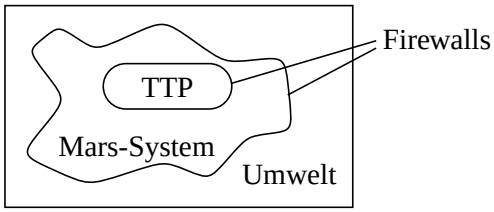


Bild 2-16: TTP-Firewalls

der anderen Seite befindliche System auf die korrekte Funktionsfähigkeit seines Partners verläßt, bzw. nur deshalb die eigene Funktionsfähigkeit garantiert werden kann.

Das Time Triggered Protocol ist ein Vorschlag der TU Wien mit dem Anspruch, eine Lösung für sicherheitskritische Echtzeitanwendungen zu sein. Daß dies mit den prioritätsgesteuerten Systemen nicht möglich sei [93], konnte allerdings noch nicht nachgewiesen werden.

Weitere Details zum Time Triggered Protocol und insbesondere eine kritische Auseinandersetzung mit seiner Leistungsfähigkeit können [75] entnommen werden.

2.3.3 Einteilung der Kommunikationsanforderungen anhand der SAE-Classes

Die Society of Automotive Engineers (SAE) hat zur Klassifikation und schließlich auch Standardisierung von Steuergerätenetzen und deren Kommunikationsanforderungen ein 3-Klassensystem aufgestellt, das die jeweiligen speziellen Anforderungen einer Klasse festhält und vergleichbar macht [130]. Diese Einteilung soll v.a. auch der Migration der konventionellen Verkabelungs- und Kommunikationstechnik hin zur Vernetzung mittels Bussystemen dienen. Die durch die Klassen beschriebenen Steuergerätenetze haben die folgenden Eigenschaften:

- Class A: Ein Steuergerätenetz, das einen Bus als Multiplexgrundlage verwendet, wodurch der Verkabelungsaufwand reduziert werden soll (Raummultiplex wird in einen Zeitmultiplex umgewandelt). Zielanwendungen sind langsame, zeitunkritische Schaltvorgänge.
- Class B: Wie A. Zusätzlich wird Gebrauch von der Multiplexeigenschaft des Busses gemacht, indem Sensorsignale von mehreren Steuergeräten gelesen werden und nicht, wie bisher, mehrfach über das Netz übertragen oder von verschiedenen Sensoren bereitgestellt werden. Steuergeräte dieser Netze existierten typischerweise auch schon in konventionell verkabelten Systemen.
- Class C: Das Netz muß hohe Datenraten auch mit Echtzeitanforderungen gewährleisten, um z.B. geschlossene Regelkreise realisieren zu können. Zu diesen Anwendungen gehören u.a. die Motorsteuerung und das ABS.

Die Anforderungen der Klassen bauen aufeinander auf, so daß eine Abwärtskompatibilität gewährleistet ist, d.h. mit einem Class C-Netz können auch die Anforderungen an ein Class A oder ein Class B-Netz erfüllt werden. Schließlich ist es, neben den technischen Anforderungen, insbesondere auch wieder der Kostenaspekt, welcher für die Wahl der Implementierung einer speziellen Klasse entscheidend wird.

Wendet man diese Klassifikation auf die Systeme an, die in dieser Arbeit untersucht und geplant werden, dann handelt es sich ausschließlich um Class C-Systeme. Ein ausführliches Beispiel dazu wird in Kapitel 6 betrachtet.

2.4 Zukünftige automotive Netzstrukturen

Das Datenaufkommen in den Fahrzeugen steigt von Jahr zu Jahr stetig, nicht zuletzt durch die immer stärker ins Fahrzeug drängenden Multimedia- und Telematikanwendungen. Spätestens bei der Übertragung von externen Videobildern in das Fahrzeug sind die bisher vorgestellten Kommunikationskonzepte nicht mehr ausreichend, da sie die geforderten Nutzdatenbandbreiten von z.B. mind. 1 Mbit/s für MPEG-codierte Bewegtbilder nicht bereitstellen können. Vor allem nicht, solange gleichzeitig Echtzeitanforderungen aufrechterhalten werden. Zur Lösung bieten sich zwei Methoden an: erstens, die gleichzeitige Verwendung mehrerer Systeme mit niederer Bandbreite oder zweitens, die Verwendung von High-Speed LAN. Letztere Variante wird trotz hoher Knotenpreise immer interessanter, da sie außerdem eine einfachere Ankopplung an externe Hochgeschwindigkeitssysteme erwarten läßt. Damit ist auch das zweite Thema zukünftiger Netzstrukturen motiviert: die Anbindung an externe Netze.

2.4.1 Lokale Hochgeschwindigkeitsnetze

Zur Erreichung hoher Bandbreiten lassen sich mehrere preisgünstige, existierende Systeme mit niederer Bandbreite parallel betreiben. Dies hat außer der erhöhten Verfügbarkeit den Vorteil, daß unterschiedlich sicherheitskritische Daten (siehe Abschnitt 2.1.1.2) physikalisch voneinander getrennt werden können. Die Kopplung dieser Systeme erfolgt mit Steuergeräten, die eine Gateway-Funktionalität übernehmen, d.h. in der Regel mit zwei Buscontrollern ausgestattet sind. Eine entsprechende Task leistet schließlich die Vermittlungsfunktion zwischen den beiden Bussen. Bild 2-1 zeigte bereits eine derartige Anordnung.

Existieren große Geschwindigkeitsunterschiede zwischen den Teilnetzen, so können diese nur mittels dynamischem Pufferspeicher ausgeglichen werden. Dieser ist in der Regel teuer und scheidet deshalb für Massenprodukte aus.

Die offensichtlichen Nachteile dieser Lösung können an zwei Punkten festgehalten werden: zum einen steigt wiederum der Verkabelungsaufwand (wenn zu einem Steuergerät zwei Busanschlüsse gelegt werden müssen), zum anderen müssen zwei oder mehr Systeme aufeinander abgestimmt und angepaßt werden. Dies ist in der Regel jedoch nur dann einfach möglich, sofern die Koppeleinheiten sehr leistungsfähig und damit teuer sind [156].

Komponenten heutiger lokaler Hochgeschwindigkeitsnetze, wie z.B. FDDI oder auch ATM (insbesondere die Controller, die das Protokoll umsetzen), sind im Moment für den Einsatz in Fahrzeugen noch zu teuer (Kosten liegen im Bereich von mehreren hundert DM). Allerdings wird man längerfristig, wegen der steigenden Bandbreitenanforderung, nicht ohne die Verwendung dieser Medien auskommen[3]. Dabei ist es nicht zwingend notwendig, daß das gesamte Fahrzeug mit Hochgeschwindigkeitskomponenten ausgerüstet wird. Vielmehr sind heterogene hierarchische Systeme sinnvoll, die auf die jeweiligen Anforderungen abgestimmt sind. Bereits heute ist eine Vielzahl von Protokollen auf dem Markt, die an bestimmte Anforderungen angepaßt sind (beispielhaft seien hier der CAN-Bus im Bereich der schnellen Echtzeitdatenübertragung und D2B-Ringe für die Übertragung von Audiosignalen genannt) und die gemeinsam in Fahrzeugen zur Anwendung kommen.

2.4.2 Die Anbindung an externe Netze

Die Anbindung externer Netze kann auf einfache Weise durch entsprechende Gateways erfolgen, solange ähnliche Protokolle in den Netzen verwendet werden. Sie wird allerdings deutlich komplexer, sobald sehr unterschiedliche Protokolle (unterschiedliche Adressierung und Bandbreite) zu einem gemeinsamen System integriert werden sollen [95]. Hier existieren bereits Realisierungen, welche einerseits den digitalen Mobilfunk (GSM) einbinden [115] und andererseits Lösungen, welche Kommunikationskanäle, z.B. der Inmarsat-Satelliten, ausnutzen [157]. Die Anforderungen des Marktes werden in diesem Bereich dazu beitragen festzulegen, welche Protokolle und Standards sich tatsächlich weiter durchsetzen werden.

Betrachtet man die rasante Entwicklung der für Dienste notwendigen Bandbreite (mit dem Ziel der Video-Übertragung ins Fahrzeug), so werden die momentan verfügbaren Standards bald nicht mehr ausreichen, da die damit zur Verfügung stehende Bandbreite (9,6 kbit/s bei GSM/ DCS 1800), auch mit den in GSM Phase 2 möglichen parallelen Kanälen, für die zukünftigen Multimedia-Anwendungen nicht ausreichen werden, ganz abgesehen von der Überlastung der

3. Eine Übersicht über die existierenden Standards lokaler Hochgeschwindigkeitsnetze (High Speed LAN), insbesondere der Token-basierten Systeme, gibt [148].

einzelnen Funkzellen. Hier müssen noch Standards und Protokolle gefunden werden, die eine ausreichend hohe Bandbreite auch in bewegten Fahrzeugen sicherstellen können. Eine weitere aktuelle Entwicklung entschärft diese Problematik etwas, wirft jedoch neue Probleme auf. Die durch Laufzeitumgebungen, wie z.B. Java, möglich gewordene Verlagerung der Rechenleistung in die Benutzersysteme und die damit zusammenhängende Reduzierung der zu übertragenden Daten, führt zu geringeren Bandbreiten gegenüber herkömmlichen Systemen. Allerdings geht dies zu Lasten der Benutzersysteme, die aufgrund der erhöhten Rechenleistung teurer werden. Welche dieser Lösungen und Wege (hohe Bandbreite bei der Übertragung oder leistungsfähige Endsysteme) sich durchsetzen wird, hängt nicht zuletzt von den Kosten für die Datenübertragung ab.

Bild 2-17 zeigt ein mögliches Szenario eines heterogenen Fahrzeugnetzes. Gesteuert über ein Zentral-Steuergerät (ZSG) mit Gateway-Funktion lassen sich damit Dienste erstellen, die über die Grenzen eines (Teil-)Segments hinausgehen. Beispielsweise kann zur Erstellung eines Telematik-Dienstes Navigationsinformation, die über das GPS-Steuergerät empfangen wurde, mit einem Routenvorschlag kombiniert werden, der über den GSM-Kanal (oder den Hochgeschwindigkeitskanal, wenn z.B. detaillierte Karten gewünscht werden) bei einem externen Dienstleister angefordert wurde. Das Resultat der Anfrage und die anschließende Zielführung können dem Fahrer über das Video-Display angezeigt und über Audio-Komponenten akustisch mitgeteilt werden. Ergänzt wird das Szenario durch den Anschluß von PDAs per Infrarot-Schnittstelle (IRDA), die den Mitfahrenden den Zugang zu den externen Netzen bieten.

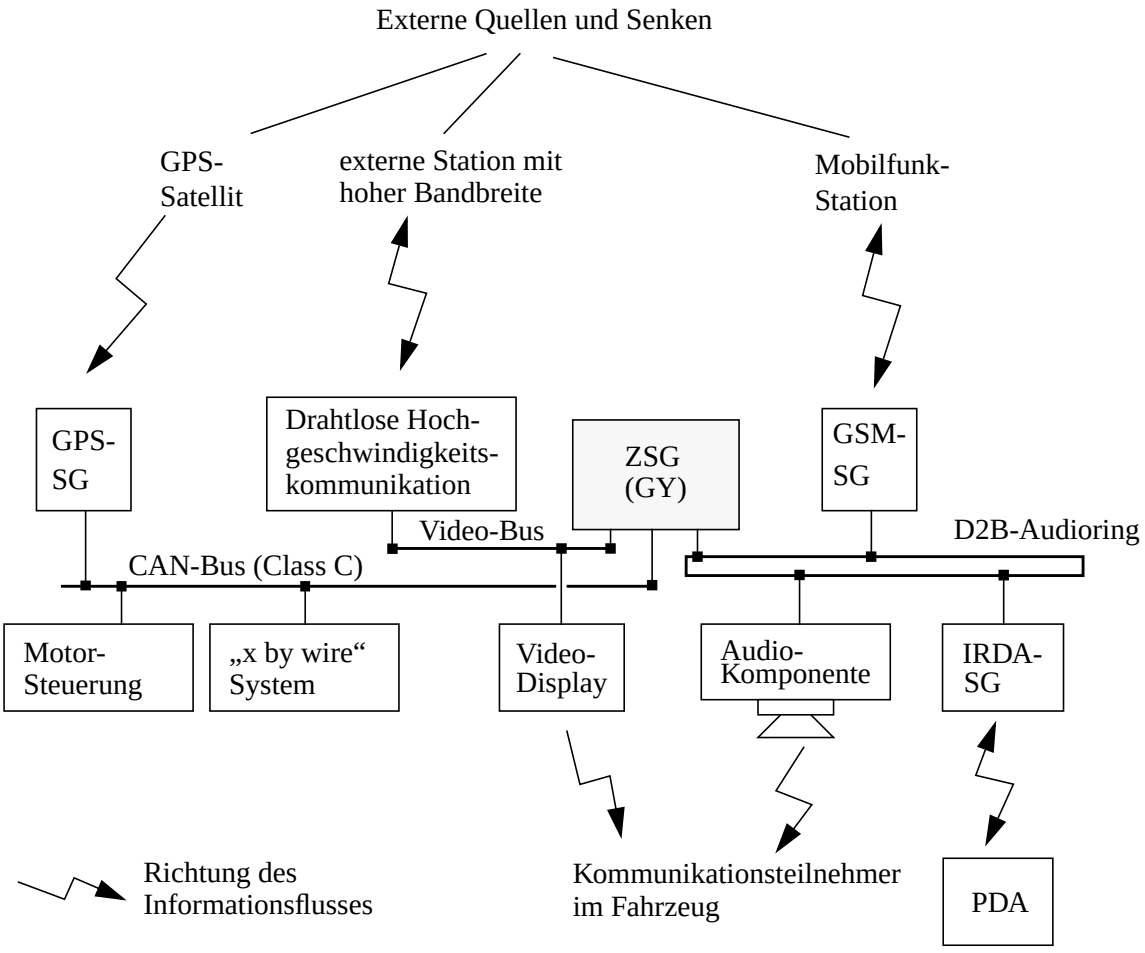


Bild 2-17: Heterogenes automotives Netz mit netzübergreifenden Diensten

Kapitel 3

Echtzeitprioritätensysteme

Die in automotiven Systemen eingesetzten Betriebssysteme arbeiten in der Regel mit einer prioritätsgesteuerten Scheduling-Strategie, um eine den Anforderungen entsprechende Vergabe der Betriebsmittel gewährleisten zu können. Da Echtzeitanforderungen hierbei eine zentrale Rolle spielen, werden zunächst die Eigenschaften von Echtzeitsystemen dargestellt. Daran schließt sich eine Taxonomie der unterschiedlichen Scheduling-Strategien an.[1]

Im weiteren Verlauf des Kapitels werden die beiden Grundformen prioritätsgesteuerter Scheduler, unterbrechende und nichtunterbrechende Systeme, vorgestellt. Weiterhin werden Algorithmen eingeführt, mit deren Hilfe eine sinnvolle Vergabe der Prioritäten auf einem Prozessor vollzogen werden kann. Methoden für den Entwurf und die Entwurfsbewertung von Echtzeitsystemen mit statischen Prioritäten auf einem Prozessor bilden den Hauptteil dieses Kapitels. Hierzu erfolgt eine Vorstellung von Analysemethoden, mit denen die Ausführbarkeit (feasibility oder schedulability) eines erstellten Rechenplans (schedule) nachweisbar wird und insbesondere die maximale Antwortzeit (worst-case response time) für einen Prozeß (task) berechnet werden kann.

3.1 Echtzeitsysteme

Wie bereits in Kapitel 1 erwähnt, müssen viele der Anforderungen und Dienste eines automotiven Systems in Echtzeit erfüllt werden. Echtzeitbetrieb ist definiert als

> „Der Betrieb eines ... Systems, ... dessen Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zufälligen Zeitspanne oder zu vorherbestimmten Zeitpunkten anfallen“ [31].

Die Schwierigkeit für Echtzeitsysteme liegt darin, daß außer der Rechtzeitigkeitsforderung, also dem Bereitstellen eines Ergebnisses vor einem Fertigstellungstermin oder einer Frist (*deadline*), zusätzlich eine Gleichzeitigkeitsforderung besteht [104]. Unter der Gleichzeitigkeit versteht man die Tatsache, daß auf mehrere gleichzeitig ablaufende Prozesse reagiert werden muß. Eine weitere Anforderung ist die Verläßlichkeit (Ausfallsicherheit) solcher Systeme, die hier allerdings nicht weiter in die Betrachtung einbezogen werden soll.

1. Man spricht bei den betrachteten Systemen auch häufig von sog. „eingebetteten Systemen“ (embedded systems), da sie in eine technische Umgebung integriert sind und keine eigene Benutzerschnittstelle haben.

Untersucht man die Rechtzeitigkeitsanforderung, so kann man zwei grundlegende Fälle unterscheiden:

- Harte Zeitanforderungen (hard real-time constraints), d.h. Verspätungen sind unter keinen Umständen zulässig, da die Ergebnisse dann entweder nutzlos oder gar schädlich sind [55]. Die harten Zeitanforderungen ergeben sich dabei normalerweise durch die physikalischen Gesetze der gesteuerten Prozesse.
- Weiche Zeitanforderungen (soft real-time constraints), bei denen als Konsequenz aus einer zunehmenden Verspätung der Ergebnisse die Kosten ständig steigen.

Beispielhaft für eine harte Echtzeitbedingung sei das Auslösen eines Sicherheitsstopps genannt, wenn beim Schließen eines Fahrzeugfensters ein Widerstand festgestellt wird. Gelingt es nicht, diese Information schnell genug zu bearbeiten und den Fenstermotor anzuhalten, so kann dies fatale Folgen haben. Demgegenüber handelt es sich bei der Zeitanforderung beim Aufschalten des Wähltons einer Telefonvermittlung lediglich um eine weiche Echtzeitbedingung. Für 90 % aller Fälle darf die Verzögerung nur 400 ms betragen. Wird diese Forderung jedoch nicht eingehalten und der Wählton kommt verspätet, bleibt das Gesamtsystem immer noch funktionsfähig.

3.2 Klassifikation von Scheduling-Strategien

Um eine Einordnung der Echtzeit-Prioritätensysteme zu ermöglichen, soll zunächst eine kurze Klassifikation von Scheduling-Strategien erfolgen. Abhängig vom Anwendungsfall und somit von den Ressourcen-Randbedingungen, wie z.B. vorhandene Rechen- oder Speicherkapazität, existiert eine Vielzahl von Scheduling-Strategien [22, 25, 143]. Das zugrundeliegende Scheduling-Problem der Betriebsmittelzuteilung kann durch das folgende Modell (Bild 3-1) beschrieben werden.

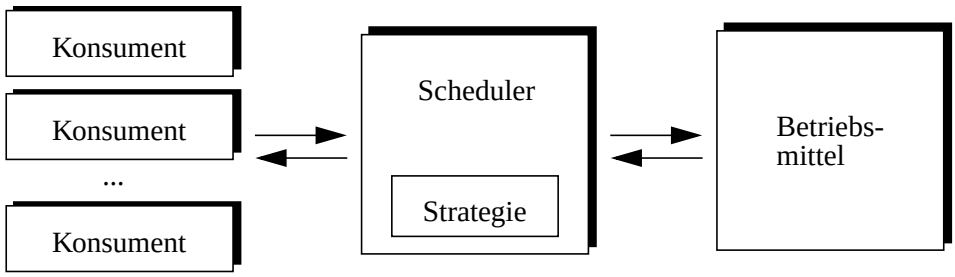


Bild 3-1: Grundlegendes Scheduling-Modell

Konsumenten benötigen zur Leistung einer bestimmten Aufgabe ein Betriebsmittel. Da dieses Betriebsmittel über dem Ablauf der Zeit mit anderen Konsumenten geteilt werden muß, kann ein einzelner Konsument keinen exklusiven Zugriff erhalten, sondern lediglich aufgrund bestimmter Regeln auf das Betriebsmittel zugreifen. Diese Regeln werden durch die Strategie des Schedulers umgesetzt. Abgebildet auf den Anwendungsfall eines automotiven Kommunikationsnetzes lassen sich die folgenden Komponenten zuordnen: Konsumenten sind zum einen die Tasks, die auf einzelnen Knoten ablaufen und zum anderen die Botschaften, die über das Kommunikationssystem ausgetauscht werden. Als Betriebsmittel, die geteilt werden müssen, existieren die CPU, Speicher im System und das gemeinsame Kommunikationsmedium.

Die folgende Taxonomie soll einige der grundlegenden Verfahren und Strategien in Zusammenhang bringen:

- Den grundlegendsten Unterschied zwischen Scheduling-Strategien bildet die Aufteilung in *lokale* und *globale* Strategien. Lokales Scheduling beschäftigt sich mit der Betriebsmittelvergabe auf einem Einprozessorsystem, während sich das globale Scheduling mit der Frage beschäftigt, *wo* in einem Mehrprozessorsystem eine Aufgabe sinnvoll bearbeitet werden soll (Lastverteilung). Im letzten Fall wird es letztendlich der lokalen Scheduling-Strategie des jeweiligen Prozessors überlassen, die Betriebsmittel zu vergeben.
- Ein weiteres Unterscheidungskriterium der Scheduling-Strategien ist der Zeitpunkt, zu dem der Schedule erstellt wird. Berechnet man den Schedule vor der Systemlaufzeit, so spricht man von *statischen* Systemen (static oder deterministic scheduling). Wird der Schedule während der Systemlaufzeit erstellt oder sich ändernden äußeren Bedingungen angepaßt, dann handelt es sich um eine *dynamische* Strategie (dynamic scheduling). Die Bedingungen an ein statisches System sind prinzipbedingt strenger als an ein dynamisches System. Im ersteren müssen alle Prozesse und ihre Anforderungen zu dem Zeitpunkt bekannt sein, an dem die Software für die beteiligten Knoten erstellt wird. Im Fall des dynamischen Scheduling sind diese Bedingungen nicht so streng, insbesondere ist kein großes a priori Wissen über die Ressourcenwünsche eines Prozesses notwendig. Allerdings müssen hier genügend Berechnungs- und Systemressourcen bereitstehen, um während der Laufzeit einen gültigen Schedule ermitteln zu können.
- Ein weiteres Unterscheidungsmerkmal besteht darin, ob die Systeme mit oder ohne *Prioritäten* arbeiten. Systeme ohne Prioritäten können wiederum nach verschiedenen Zuteilungsverfahren unterschieden werden. Die bekanntesten sind das „First in first out“-, das „Last in first out“- und das „Random“-Verfahren, die jeweils vorgeben, wie konkurrierende Anforderungen behandelt werden. Prioritätensysteme selbst lassen sich in unterbrechende und nichtunterbrechende Systeme unterteilen. Eine ausführliche Diskussion der Prioritätensysteme erfolgt in Abschnitt 3.4.

Oftmals werden in realen Systemen Mischformen dieser Grundstrategien implementiert. Insbesondere bei den in dieser Arbeit betrachteten verteilten Systemen trifft man gemischte Realisierungen an: Steuergeräte sind oft mit einem unterbrechenden Prioritäten-Scheduler ausgerüstet, während das verbindende Bussystem nur nichtunterbrechende Prioritäten zuläßt. Allerdings sind alle Prioritäten bereits vor der Systemlebenszeit vergeben. Man spricht in diesem Fall des statischen Prioritäten-Schedulings von „Fixed Priority Scheduling“.

Diskutiert man die Scheduling-Strategien aus der Sicht der Erreichbarkeit der Ziele, so können *optimale* und *suboptimale* Lösungen unterschieden werden. Kennt man alle Systemzustände a priori und handelt es sich um keine allzugroßen Systeme, dann kann ein, bezüglich eines bestimmten Kriteriums (minimale Prozeßgesamtlaufzeit, maximale Ausnutzung der Ressourcen oder maximaler Systemdurchsatz), optimaler Schedule erstellt werden. Können diese Kriterien, z.B. weil das System zu groß ist, nicht ohne weiteres berechnet werden, kann man mit suboptimalen Lösungen arbeiten. Hierbei werden wiederum zwei Kategorien unterschieden. Bei *heuristischen* Lösungen werden Regeln verwendet, die intuitiv als richtig erscheinen und normalerweise durch indirekte Kontrolle von Systemparametern eine Problemvereinfachung darstellen. Die zweite Kategorie sind *Näherungsverfahren*, bei denen eine Lösung akzeptiert wird, sofern sie die Randbedingungen erfüllt. Wichtige Punkte bei der Bewertung einer solchen Näherungslösung sind:

- die Zeit, die gebraucht wird, eine Lösung zu finden,
- die Fähigkeit, eine gefundene Lösung zu bewerten, und
- eine Möglichkeit, den Lösungsraum intelligent zu durchsuchen.

Die im letzten Punkt angesprochene intelligente Suche im Lösungsraum, die natürlich auch Auswirkungen auf die Suchzeit hat, wird in Kapitel 4 näher betrachtet. Tabelle 3-1 faßt die vorgestellten Kriterien nochmals kurz zusammen.

Kriterium	Strategie	Bemerkung
Betriebs-mittelver-gabe	- lokal - global	Betriebsmittelvergabe (Zeit) im Einprozessorsystem Betriebsmittelvergabe (Ort) im Mehrprozessorsystem
Zeitpunkt der Planung	- statisch - dynamisch	vor der Systemlebenszeit während der Systemlebenszeit
Abarbei-tungsver fahren	- Prioritäten - LiFo - FiFo - Random	Reihenfolge entsprechend der Priorität jüngste Anforderung als nächste älteste Anforderung als nächste willkürliche Anforderung als nächste
Lösungs-qualität	- optimal - suboptimal	es existiert keine bessere Lsg. bezügl. eines best. Kriteriums Erfüllt Randbedingungen, ist aber nicht optimal

Tabelle 3-1: Übersicht über Scheduling-Strategien

3.3 Modellierung der Anforderungen

Anforderungen, die ein eingebettetes System bearbeiten muß, zeigen meist ein charakteristisches zeitliches Verhalten. Betrachtet man bestimmte Regelkreise, so erkennt man, daß die dadurch erzeugten Anforderungen periodischen Charakter haben. So wird z.B. in einem Fahrzeug in bestimmten Zeitabständen die Motordrehzahl gemessen und diese anderen Steuergeräten zur Weiterverarbeitung übermittelt. Diese Periodizität ist die Grundlage für die später vorgestellten Analyse- und Prioritätszuteilungsverfahren. Allerdings sind nicht alle Anforderungen periodisch. Das Auslösen des Airbags, das durch ein Signal eines Crash-Sensors initiiert wird, ist ein hochgradig nichtperiodischer, sog. sporadischer Vorgang. Die Periodendauer geht hierbei gegen unendlich, da ein einmal ausgelöster Airbag nicht mehr in diesem System verwendet wird. Allerdings bleiben die harten Echtzeitanforderungen trotzdem bestehen, so daß man derartige Anforderungen ebenfalls mitberücksichtigen muß. Insgesamt werden drei Typen von Anforderungen unterschieden (siehe Bild 3-2):

- Periodische Anforderungen: Für sie kann eine feste Periodendauer T angegeben werden, die den Abstand zwischen zwei Anforderungen festlegt.
- Aperiodische Anforderungen: Für diesen Typ kann eine minimale Periodendauer angegeben werden, die nicht zwischen zwei Anforderungen unterschritten werden kann.
- Sporadische Anforderungen: Für sie können keine Aussagen über die Häufigkeit des Auftretens gemacht werden.

Die Grundlage der meisten Analyseverfahren ist die Annahme von periodischen Anforderungen. Um diese Analyseverfahren anwenden zu können, müssen sporadische und aperiodische Anforderungen auf periodische Anforderungen abgebildet werden [118]. Ein derartiges Verhalten ist z.B. durch die Bearbeitung mit „Polling Servern“ möglich [136, 143]. Diese Server sind Prozesse, die, selber periodisch, die Anforderungen in bestimmten Zeitabständen bearbei-

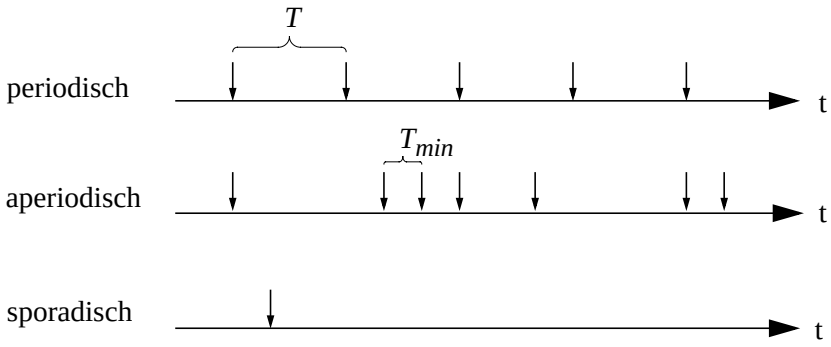


Bild 3-2: Typen von Anforderungen

ten. Allerdings kann die Antwortzeit für eine aperiodische Anforderung sehr hoch werden, wenn sie kurz nach dem Ablauf der pollenden Task in das System eintritt. Daß mit dieser Methode u.U. ein Großteil der Ressourcen verschwendet wird, ist offensichtlich, aber unvermeidbar.

Betrachtet man die beiden zuletzt genannten Anforderungstypen ohne die Verwendung eines zusätzlichen Servers näherungsweise als periodisch, indem ein minimaler Abstand T_{min} angenommen wird, dann werden einerseits sicherlich ebenfalls Ressourcen verschwendet, andererseits hat man aber die Voraussetzung geschaffen, um für diesen Fall die harten Echtzeitanforderungen garantieren zu können. Befinden sich außerdem Anforderungen mit weichen Echtzeitanforderungen im System, so können die für den „Notfall“ reservierten Ressourcen im Normalbetrieb problemlos diesen Anforderungen zur Verfügung gestellt werden.

Allgemein werden durch die Anforderungen an ein System Tasks gestartet. Diese Tasks werden durch die folgenden zeitlichen Merkmale beschrieben (vgl. Gantt-Diagramm in Bild 3-3):

- die Periodendauer *T*, die den Abstand zwischen zwei Anforderungen angibt,
- die Bearbeitungs- oder Bedienzeit (computation time) *C*, das ist die Zeit, in der die Task das benötigte Betriebsmittel (z.B. die CPU) zur Verfügung hat,
- die Antwortzeit (Durchlaufzeit, response time) *R*, die angibt, wieviel Zeit vom Eintritt der Anforderungen bis zum Ende der Bearbeitung verstreicht. Diese Zeit ist die Summe aus der eigenen Bearbeitungszeit und einer Wartezeit *W*, in der auf die Zuteilung des Betriebsmittels gewartet werden muß. Dieses Warten resultiert z.B. durch die Interferenz oder Beeinflussungszeit (interference) *I*. Das ist die durch andere Anforderungen verursachte Wartezeit.
- Schließlich existiert die Frist (deadline) *D*, die angibt, bis zu welchem Zeitpunkt die Anforderung bearbeitet sein muß, wenn Echtzeitforderungen bestehen.

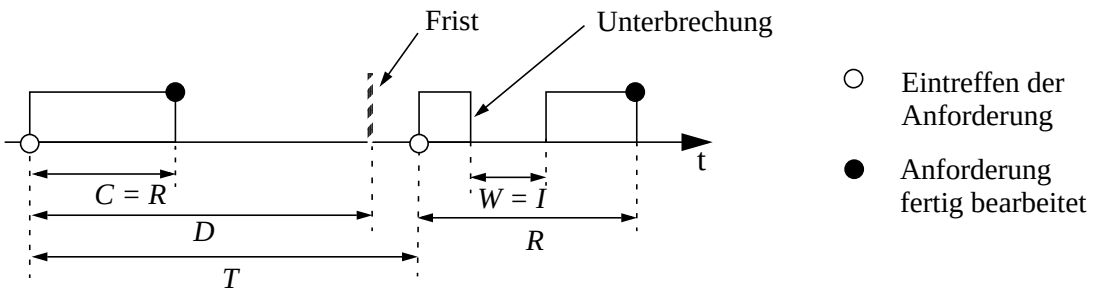


Bild 3-3: Zeitverhalten einer Task

Bei der ersten Anforderung ist die Anwortzeit gleich der Bearbeitungszeit, im zweiten Fall tritt eine Wartezeit durch Interference auf. Bearbeitungszeit und Wartezeit können natürlich aus einer beliebigen Zahl von Teilabschnitten zusammengesetzt sein.

3.4 Prioritätensysteme

Die in dieser Arbeit betrachteten Echtzeitsysteme basieren alle auf dem Prioritätenprinzip: ein Betriebsmittel wird genau dem anfordernden Prozeß zugeteilt, der im Moment der Zuteilungsentscheidung die höchste Priorität besitzt. Damit lassen sich Bedienstrategien realisieren, die Anforderungen aufgrund ihrer Wichtigkeit, Herkunft oder Dringlichkeit berücksichtigen. Die Zuteilungsentscheidung wird dabei durch eine Bedieneinheit *BE*, den Scheduler, realisiert, der in bestimmten Abständen aktiviert wird. Dies kann entweder in festgelegten Zeitabständen sein, oder wenn eine neue Anforderung eintrifft oder aber dann, wenn die Bearbeitung einer Anforderung beendet ist. Bei den hier betrachteten Systemen handelt es sich stets um statische Prioritäten, d.h. sie werden bereits vor der Systemlebenszeit vergeben und den Anforderungen zugeordnet. Außerdem ist die Bearbeitungszeit für eine Anforderung eine bekannte und konstante Größe, so daß ein deterministisches Bedienmodell (*D*) angenommen werden kann. Bild 3-4 zeigt die Modellierung eines Prioritätensystems in Form eines einfachen Warteschlangenmodells.

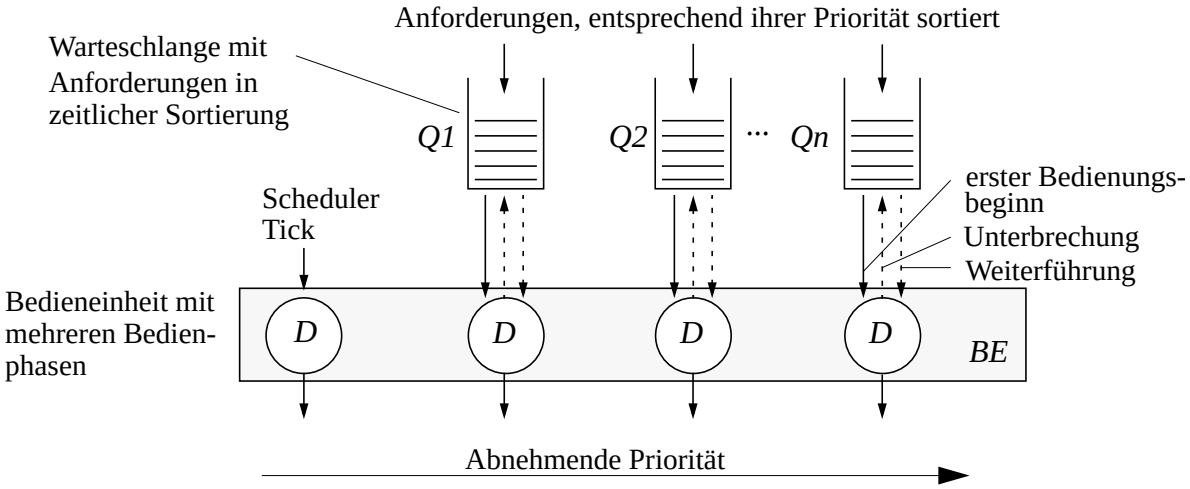


Bild 3-4: Prioritätensystem als Warteschlangenmodell

Die Bedieneinheit des Systems besteht aus mehreren Bedienphasen. Werden, wie oben beschrieben, zu bestimmten Zeiten die Entscheidungen getroffen, welche Anforderung als nächste bearbeitet wird, so muß ein Scheduler mit einer entsprechenden Bedienphase berücksichtigt werden. Der dafür nötige Scheduler-Tick (vgl. Timer-Interrupt in Abschnitt 2.2.2) hat i.d.R. die höchste Priorität im System.

Ankommende Anforderungen werden entsprechend ihrer fest vorgegebenen Priorität in die zugehörige Warteschlange *Q* einsortiert. Der linke Pfeil zwischen Warteschlange und Bedienphase zeigt den ersten Bedienungsbeginn an, der in jedem der betrachteten Systeme auftritt. Zusätzlich kann aber eine Anforderung ebenso wieder in die Warteschlange zurückgestellt und deren Bearbeitung zu einem späteren Zeitpunkt weitergeführt werden. Daß derartige Unterbrechungen mehrfach auftreten können, ist durch die beiden gestrichelten Pfeile angedeutet. Das Verhalten der Bedieneinheit charakterisiert das Prioritätensystem wesentlich und soll in den folgenden Unterabschnitten genauer untersucht werden.

3.4.1 Unterbrechende und nichtunterbrechende Systeme

Aus den oben aufgeführten Verhaltensstrategien eines fiktiven Schedulers lassen sich die beiden Grundformen des Prioritätenschedulings ableiten, die unterbrechenden und die nichtunterbrechenden Systeme.

- Unterbrechende Systeme (preemptive systems) erlauben einer höherprioren Anforderung, daß sie eine niederpriore Anforderung, die sich gerade im Besitz eines Betriebsmittels befindet, bei der Arbeit unterbricht und selbst das Betriebsmittel für sich in Anspruch nimmt. Ist die Belegung durch die hochpriore Anforderung beendet, und das Betriebsmittel wieder freigegeben, so kann die niederpriore Anforderung entweder an der unterbrochenen Stelle weiterarbeiten (resume) oder aber nochmals von vorne gestartet werden (restart). Dabei ist letzteres Verhalten in heutigen Systemen seltener anzutreffen. Das Weiterbearbeiten der niederprioren Botschaft wird noch weiter verzögert, wenn zwischenzeitlich weitere höherpriore Anforderungen in der Prioritätswarteschlange *Q2* eintreffen und die Betriebsmittel beanspruchen. Typische Vertreter dieser Strategie sind fast alle aktuellen Echtzeit-Betriebssysteme. Aber auch das UNIX-Betriebssystem arbeitet nach dieser Methode, wobei der Scheduler nach festen Zeitintervallen („*time slices*“) aktiviert wird und weitere Algorithmen dafür sorgen, daß die Priorität eines Prozesses verändert werden kann, sobald er unterbrochen wurde. Damit kann wechselnden Lastszenarien Rechnung getragen werden. Allerdings ist UNIX in seiner Urform nicht echtzeitfähig.
- In nichtunterbrechenden Systemen (non-preemptive systems) geben Prozesse ein Betriebsmittel erst wieder frei, wenn sie es nicht mehr brauchen. Dann wird durch Vergleich der Prioritäten der in Warteschlange *Q1* wartenden Anforderungen entschieden, welche Anforderung als nächste das Betriebsmittel zugesprochen bekommt. Typische Vertreter für nichtunterbrechende Systeme sind die bereits vorgestellten Bussysteme CAN und VAN. Auch einfachste Betriebssystemscheduler werden nach dieser Strategie implementiert.

Bild 3-5 zeigt Szenarien mit den beiden Grundvarianten des Prioritäten-Schedulings. Dabei wird der zeitliche Ablauf für eine Taskmenge dargestellt, deren drei Tasks die jeweils gleiche Bearbeitungszeit *C* beanspruchen. Die Bearbeitung des gesamten Schedules dauert in beiden Fällen gleich lang. Allerdings muß beim nichtunterbrechenden Fall die Anforderung mit mittlerer Priorität *mp* im Vergleich zur Anforderung mit niedriger Priorität sehr lange warten, bis sie fertig bearbeitet ist. Obwohl sie sich zum Zeitpunkt $t = 2$ bereits in der Warteschlange befindet, wird sie von *hp* verdrängt und kann somit erst zum Zeitpunkt $t = 5$ das Betriebsmittel beanspruchen.

Bei der Bezeichnung der Prioritäten wird in der Regel der höchsten Priorität der Wert „0“ zugeordnet und mit steigendem ganzzahligem Wert nimmt die Priorität ab. Im Fall eines unterbrechenden Systems, erhält der Scheduler selbst in der Regel die höchste Priorität. Anforderungen durch ihn können z.B. über einen Timer-Interrupt generiert werden (tick scheduler). Unterbrechende und nichtunterbrechende Systeme lassen sich einheitlich durch die Unterbrechungsdistanz-Prioritäten (UD-Prioritäten) charakterisieren [63]. Dazu werden die in einem System vorhandenen Prioritäten in Klassen eingeteilt, zwischen denen die Unterbrechungsdistanzen festgelegt werden. Im einfachsten Fall ist jeder Klasse genau eine Priorität zugeordnet. Die Unterbrechungsdistanz bezeichnet den Abstand zur nächsten Prioritätsklasse, die unterbrochen wird. Damit lassen sich auch Zwischenstufen der vorgestellten Grundmechanismen darstellen. Die Grundmechanismen selbst bilden die Spezialfälle der UD-Prioritäten. Ist die

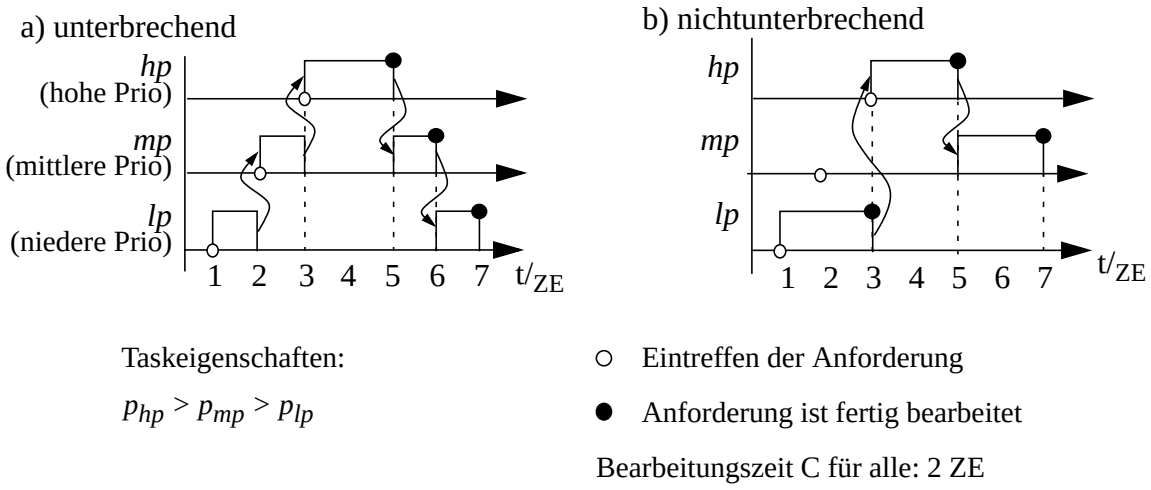


Bild 3-5: Gantt-Diagramme für nichtunterbrechende und unterbrechende Prioritätensysteme

Unterbrechungsdistanz gleich 1, so hat man den Fall des unterbrechenden Systems, da mit Abstand 1 bereits die nächste benachbarte Prioritätsklasse unterbrochen wird. Ist die Unterbrechungsdistanz gleich der Zahl der vorhandenen Prioritätsklassen, dann handelt es sich um ein nichtunterbrechendes System.

Auf eine weitere Charakterisierung von Prioritätssystemen, insbesondere in solche mit verdrängenden und mit nichtverdrängenden Prioritäten, bei denen im ersten Fall wartende Anforderungen und im zweiten Fall ankommende Anforderungen verloren gehen, wenn der Warteraum gefüllt ist, soll hier nicht eingegangen werden. Es wird vielmehr davon ausgegangen, daß durch entsprechendes Systemdesign ausreichend Warteraum zur Verfügung steht.

3.4.2 Vergabe von statischen Prioritäten in Einprozessorsystemen

Die Vergabe der Prioritäten legt wesentlich die zeitlichen Eigenschaften eines statischen Prioritätensystems fest. Der Planung des Schedules und der damit verbundenen Vergabe der Prioritäten fällt deshalb ein großes Gewicht zu. Erreicht man durch entsprechende Vergabe der Prioritäten, daß alle vorgegebenen Fristen eingehalten werden, so spricht man von einer „optimalen Verteilung“ bzw. von einem optimalen Schedule. Der Schedule wird somit als durchführbar (feasible) oder gültig bezeichnet (siehe Abschnitt 3.5.2).

Werden die Prioritäten nach dem *Rate-Monotonic*-Algorithmus vergeben, dann erhält die Task die höchste Priorität, die den kürzesten Ankunftsabstand besitzt [111], d.h. man vergibt die Prioritäten umgekehrt proportional zum Ankunftsabstand. Dieser Algorithmus ist für periodische Taskmengen eines unterbrechenden Systems mit folgenden Bedingungen optimal: $C_i \leq D_i = T_i$, d.h. die Frist D_i einer Anforderung mit der Priorität i muß gleich dem Ankunftsabstand T_i dieser Anforderung sein. Außerdem muß die erste Ankunft einer Task zum Zeitpunkt $t = 0$ erfolgen, d.h. es darf kein Offset O existieren. Die Gültigkeit eines derartigen Schedules kann für bestimmte Systeme nachgewiesen werden (siehe Abschnitt 3.5.2.1)

Die Vergabe von Prioritäten nach dem *Deadline-Monotonic*-Algorithmus versieht die Task mit der höchsten Priorität, welche die kürzeste Frist besitzt, d.h. Prioritäten werden umgekehrt proportional zur Frist vergeben [110]. Die Bedingung zur Verwendung des Algorithmus lautet $C_i \leq D_i \leq T_i$ und der Offset muß wiederum Null sein.

Weder der *Rate-Monotonic*- noch der *Deadline-Monotonic*-Algorithmus vergeben die Prioritäten für Systeme mit beliebigen Fristen optimal dergestalt, daß ein möglicher Schedule auch wirklich gefunden wird. Ein optimaler Algorithmus für derartige Systeme wird von Audsley [3] vorgestellt. Das Ziel des Algorithmus ist ebenfalls die Bereitstellung eines durchführbaren Schedules, d.h. die Antwortzeiten der Tasks überschreiten nicht ihre Fristen. Grundlage für den Algorithmus ist die Tatsache, daß eine Erhöhung der Priorität einer Task im Verhältnis zu den anderen Tasks des Systems eine Verkürzung der Antwortzeit für diese Task mit sich bringt, da sich die Interferenz verringert.

Ausgehend von einer zufälligen Prioritätsverteilung wird bei diesem Algorithmus zuerst die Task gesucht, die mit der niedrigsten Priorität ihre Frist erfüllen kann. Dabei ist die Prioritätsverteilung der restlichen Tasks im System unerheblich, da nur die Gesamtinterferenz durch diese Tasks von Interesse ist. Kann eine solche Task nicht gefunden werden, dann kann es auch keinen gültigen Schedule geben. Im nächsten Schritt wird versucht, die zweitniedrigste Priorität zu besetzen, und so weiter. Bild 3-6 zeigt den Ablauf des Algorithmus in Pseudo-Code.

```
einsortiert = zahl_aller_tasks;
repeat
    ready = FALSE;
    failed = TRUE;
    index = 1;
    repeat
        füge task[index] an Stelle priority[einsortiert]
        if (task[index] ist schedulable)
            einsortiert = einsortiert - 1;
            failed = FALSE;
            ready = TRUE;
        else
            füge task[index] zurück an alte Stelle

        index = index + 1;
    until (ready = TRUE) oder (index = einsortiert)
until (einsortiert = 0) oder (failed = TRUE)
```

Bild 3-6: Optimaler Algorithmus zur Vergabe von Prioritäten in einem Einprozessorsystem

Läßt sich in einem der Schritte keine geeignete Task finden, so gibt es auch keinen gültigen Schedule, d.h. der Algorithmus ist optimal bezüglich der Tatsache, daß ein existierender Schedule gefunden wird. Der Vorteil des Verfahrens besteht außerdem darin, daß von einer zufälligen Prioritätsanordnung ausgegangen werden kann. Der Nachteil ist, daß man ein Verfahren kennen muß, mit dem die Gültigkeit einer vergebenen Priorität und des damit entstandenen Schedules bestimmt werden kann.

3.5 Analysemethoden für Echtzeitprioritätensysteme mit einem Prozessor

Zur Bewertung von Echtzeitprioritätensystemen wird vorrangig die Berechnung der Antwortzeit (response time) eingesetzt. Zieht man die Echtzeitanforderungen in Betracht, dann hat insbesondere die Bestimmung der *Worst-case*-Antwortzeit von Tasks einen hohen Stellenwert. Zu ihrer Berechnung stehen für Einprozessorsysteme analytische Methoden zur Verfügung, die im folgenden eingeführt werden. Statistische Methoden der Verkehrstheorie [94, 131, 76], die vielfach auf der Verwendung der Momententheorie basieren, lassen sich hier nicht nutzbringend einsetzen, da in der Regel nur Algorithmen zur Bestimmung von Mittelwerten zur Verfügung stehen, dies aber für die Charakterisierung von Systemen nicht ausreicht, bei denen bestimmte Extremeigenschaften in jedem Fall garantiert werden müssen. (Siehe auch [79].)

3.5.1 Test der Belastung eines Prozessors

Mit Hilfe eines einfachen Tests kann überprüft werden, ob der Prozessor grundsätzlich die an ihn gestellten Anforderungen erfüllen kann. Dazu wird einfach das Verhältnis der Bearbeitungszeit C_i zur Periode T_i einer jeden Task im System aufsummiert:

$$\rho = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{3-1}$$

Voraussetzung für diesen Test ist, daß jede Anforderung eine eigene Priorität i zugeordnet bekommt und diese sequentiell und ohne Lücken vergeben sind. Ist das Ergebnis $\rho \leq 1$, so ist der Prozessor bei arbeitserhaltenden Systemen (kein preemptive repeat, d.h. es wird an der Stelle fortgefahren, an der vorher unterbrochen wurde) prinzipiell in der Lage, das Angebot zu bewältigen. Es werden damit aber noch keine Aussagen über die Erfüllung der zeitlichen Anforderungen gemacht. Dies muß mit Hilfe von Rechenplantests erfolgen.

3.5.2 Test auf die Durchführbarkeit eines Rechenplans

Für die ersten beiden vorgestellten Strategien, Rate-Monotonic- und Deadline-Monotonic-Scheduling, gibt es Tests, die die Durchführbarkeit des Schedules bewerten. Die Tests gehen dabei von Systemen mit bestimmten Voraussetzungen aus, welche die Anwendbarkeit auf reale Systeme deutlich einschränken. Für eine erste Näherung sind sie allerdings auch für komplexere Systeme brauchbar. Alle Tests basieren auf einem Worst-case, dem schlechtesten Fall, der für ein gegebenes System auftreten kann. Nur wenn alle Tasks ihre Frist auch für den Worstcase erfüllen, kann man die Einhaltung für alle anderen Fälle garantieren.

3.5.2.1 Test für den Rate-Monotonic-Algorithmus

Die Voraussetzungen für die Durchführung des Tests sind:

- konstante maximale Bearbeitungszeit der Tasks
- nur periodische Anforderungen
- keine Interprozeßkommunikation
- alle Tasks sind unterbrechbar

- kein Blockieren (siehe Abschnitt 3.5.3.1)

Treten auch sporadische und aperiodische Anforderungen auf, so können diese, wie bereits ausgeführt, als periodische modelliert werden. Problematisch bleibt aber die Angabe einer Periode dann, wenn ein Ereignis mit hoher Wahrscheinlichkeit nur einmal auftritt (vgl. „Airbag“). Es existieren, abhängig vom Verhältnis von Periode zur Frist, folgende Tests:

Die Frist ist gleich der Periodendauer

Für eine Menge von n periodischen Tasks läßt sich durch die Rate-Monotonic-Strategie ein gültiger Schedule aufstellen, falls gilt:

$$W \leq n(2^{1/n} - 1) \tag{3-2}$$

für $n \to \infty$ ergibt sich:

$$W \leq 0{,}693 \tag{3-3}$$

Dieser Test ist hinreichend, aber nicht notwendig. Leider ist die Rate Monotonic Strategie nicht optimal, da es einen gültigen Schedule geben kann, obwohl eine Verteilung den Test nicht besteht. Ein Nachteil dieses Tests ist, daß keine Aussage getroffen werden kann, welche der Tasks ihre Frist verletzen. Sollen außerdem Systeme, deren Frist ungleich der Periodendauer ist, getestet werden, muß die Gleichung modifiziert werden. Dazu unterscheidet man zwei Fälle [112]:

Die Frist ist kleiner als die Periodendauer

Für diesen Fall müssen alle Fristen die folgende Bedingung erfüllen: $D_i = kT_i$, mit $1 \leq i \leq n$ und $k = 1, 2, \ldots$. Das heißt, daß die Frist D in einem festen Verhältnis k zur Periodendauer T steht. Damit lautet die Bedingung für die Ausführbarkeit:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq \begin{cases} n[(2k)^{1/n} - 1] + (1-k) : & \frac{1}{2} \leq k \leq 1 \\ k : & 0 \leq k \leq \frac{1}{2} \end{cases} \tag{3-4}$$

Die Frist ist größer als die Periodendauer

Fristen, die größer als die Periodendauer sind, unterliegen der folgenden Randbedingung: $D_i = kT_i, (1 \leq i \leq n)$. Dann gilt als Bedingung für die Ausführbarkeit:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq \begin{cases} n[2^{1/n} - 1] : & k=1 \\ k\left[(n-1)\left(\left(\frac{k+1}{k}\right)^{\frac{1}{n-1}} - 1\right)\right] : & k=2,3,... \end{cases} \tag{3-5}$$

3.5.2.2 Test für den Deadline-Monotonic-Algorithmus

Die Voraussetzungen für diesen Test sind:

- Periodische Tasks
- keine Interprozeßkommunikation
- alle Tasks sind unterbrechbar
- kein Blockieren

Der wichtigste Unterschied der Deadline-Monotonic- gegenüber der Rate-Monotonic-Strategie ist die völlig frei wählbare Frist. Sie muß, als einzige Bedingung, kleiner oder gleich der Periode sein. Bei der Rate-Monotonic-Strategie mußten die Fristen in einem bestimmten Verhältnis zur Periode stehen. Dies ist für den folgenden Test nicht notwendig. Ein vorliegender Schedule ist durchführbar, falls gilt:

$$\frac{C_i}{D_i} + \frac{I_i}{D_i} \leq 1 \qquad 1 \leq i \leq n \tag{3-6}$$

d.h. die Summe aus Bearbeitungszeit C_i und Interferenz I_i muß kleiner oder gleich der Frist D_i sein. Die Interferenz I_i einer Task *i* wird wie folgt berechnet (siehe auch Abschnitt 3.5.4.1):

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \tag{3-7}$$

Der Dachoperator $\lceil x \rceil$ (ceiling operator) liefert als Ergebnis die kleinste ganze Zahl, die größer oder gleich *x* ist.

Der Test ist hinreichend, aber nicht notwendig. Durch den Ansatz mit der Frist im Zähler des Operators ist dieser Test sehr pessimistisch, da auch Tasklaufzeiten berücksichtigt werden, die zwar innerhalb der Frist begonnen haben, deren Ende aber außerhalb der betrachteten Frist liegt. Im Gegensatz zu den Tests für die Rate-Monotonic-Strategie wird bei diesem Test jede Task für sich untersucht. Erfüllt eine Task die Bedingung nicht, kann der Test abgebrochen und der Schedule als ungültig bewertet werden.

3.5.3 Berechnung der Worst-case-Antwortzeiten

Die Berechnung der Task-Antwortzeiten stellt in vielen Fällen die Grundlage für die Bewertung eines gültigen Schedules dar. Diese Testmethode ist, im Gegensatz zu den vorgestellten Testmethoden, sowohl notwendig als auch hinreichend zur Bewertung eines Schedules auf einem Einprozessorsystem. Da sie die Basis für die Analyse innerhalb dieser Arbeit ist, wird ihre Herleitung ausführlich dargestellt. Dabei berücksichtigt die Berechnung der Antwortzeit immer mehr Faktoren, so daß die Gleichung für die Berechnung immer weiterentwickelt wird.

Allgemein kann formuliert werden, daß die Echtzeitanforderung erfüllt ist, wenn die Antwortzeit R_i (*response time*) einer Task *i* die Frist D_i für diese Task nicht überschreitet: $R_i \leq D_i$. Verschärft man diese Anforderung dergestalt, daß die schlimmste mögliche Konstellation, der „*Worst-case*“, berücksichtigt wird, hat man die Sicherheit, daß das System in jedem anderen Zustand ebenfalls funktioniert.

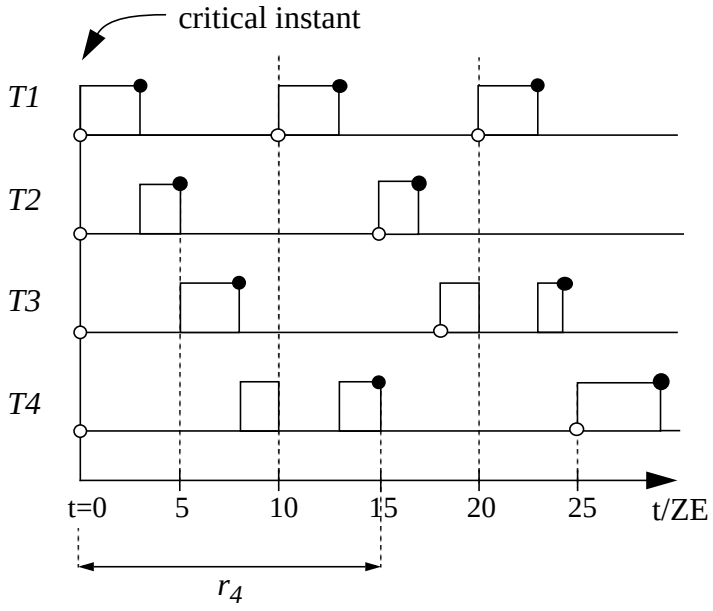
Die Grundanforderung an das System lautet deshalb:

$$r_i \leq D_i \tag{3-8}$$

Im weiteren Verlauf der Arbeit soll mit r_i die Worst-case-Antwortzeit einer Task *i* (später einer Aktivität *A*) gemeint sein.

3.5.3.1 Die Worst-case-Antwortzeit in unterbrechenden Systemen

Die Länge der Antwortzeit einer Task resultiert aus mehreren Ursachen: zum einen fließt die eigene Bearbeitungszeit in die Berechnung ein, zum anderen müssen die Einflüsse der mit um die Betriebsmittel konkurrierenden Anforderungen berücksichtigt werden. Die Grundlage für die Berechnung der Worst-case-Antwortzeit wurde von Harter [58] mit dem „time dilation"-Algorithmus gelegt. Die Idee des Algorithmus besteht darin, daß jede Task durch Tasks mit höherer Priorität eine „Dehnung" ihrer Antwortzeit erfährt. Der einfachste Fall sind dabei Systeme, bei denen die Antwortzeiten der Anforderungen kleiner sind als deren Perioden, d.h, es gilt $R \leq T$. Es läßt sich nun eine Situation konstruieren, bei der alle Anforderungen gleichzeitig in das System eintreten. Dieser Zeitpunkt wird als „kritischer Zeitpunkt" (critical instant) bezeichnet [111]. Bild 3-7 zeigt ein Beispiel für eine derartige Konstellation.



Task A	Periode T_i in ZE	Bearbeitungsdauer C_i in ZE	Priorität
T1	10	3	0
T2	15	2	1
T3	18	3	2
T4	25	4	3

Bild 3-7: Beispiel für den kritischen Zeitpunkt

Zum Zeitpunkt $t = 0$ sind alle vier Tasks des Beispiels ablaufbereit. Task *T1* erhält aufgrund der höchsten Priorität den Prozessor und kann ablaufen. Danach folgen *T2*, *T3* und *T4* mit jeweils fallender Priorität. *T4* kann allerdings nicht abschließen, da zum Zeitpunkt $t = 10$ bereits die nächste Anforderung von *T1* kommt und *T4* dadurch unterbrochen wird. Mit r_4 ist die Antwortzeit der Task *T4* für diesen critical instant gekennzeichnet. Sie setzt sich dabei aus zwei Komponenten zusammen: der eigenen Bearbeitungsdauer *C4* und der Interferenz durch höherpriore Tasks.[2]

2. Die Prioritätenvergabe erfolgt nach dem Rate-Monotonic-Algorithmus.

Die Worst-case-Antwortzeit r_i einer Task *i* berechnet sich damit wie folgt:

$$r_i = C_i + I_i \tag{3-9}$$

Die Interferenz I_i kennzeichnet dabei die maximale Zeit, in welcher der Prozessor durch die Menge der höherprioren Tasks *hp(i)* belegt ist und somit die betrachtete Task unterbricht. Sie läßt sich in diesem Fall, in Abwandlung von Gleichung 3-7 (statt der gegebenen Frist wird die gesuchte Antwortzeit in den Zähler des Dachoperators gestellt), berechnen:

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \tag{3-10}$$

Daß die Antwortzeit im Zähler des Dachoperators auftaucht ist unabdingbar, da über die gesamte Antwortzeit hinweg höherpriore Tasks die Möglichkeit haben, die betrachtete Task zu unterbrechen. Die Schwierigkeit der Beziehung besteht darin, daß der gesuchte Term für die Antwortzeit auf beiden Seiten der Gleichung auftaucht. Audsley gibt zur Lösung eine rekursive Gleichung an [4]:

$$r_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j \tag{3-11}$$

Verwendet man einen geeigneten Startwert (z.B. $r_i^0 = C_i$), so läßt sich zeigen [79], daß die Reihe konvergiert, solange die Prozessorlast kleiner als 100% ist. Abgebrochen wird, sobald gilt: $r_i^{n+1} = r_i^n$.

Blockierung durch kritische Bereiche, Prioritätsumkehrung und das Priority Inheritance Protocol

Zur Gewährleistung der korrekten Funktion kommt es oft vor, daß Anforderungen exklusiven Zugriff auf Betriebsmittel haben müssen. Dies kann z.B. sein, um eine Dateninkonsistenz in einem Speicherbereich zu vermeiden, auf den mehrere Prozesse Zugriff haben. Die Verwaltung solcher kritischer Bereiche (critical sections, cs) wird oftmals durch Semaphoren realisiert [109]. Dabei kann es in unterbrechenden Systemen zu dem ungewollten Effekt der Blockierung durch „Prioritätsumkehrung“ [102] kommen.

Es sei eine Taskmenge mit drei Tasks unterschiedlicher Priorität (siehe Bild 3-8) gegeben. Die niederpriore Task *lp* erhält das Bearbeitungsrecht zur Zeit *t* = 1 und beansprucht zum Zeitpunkt

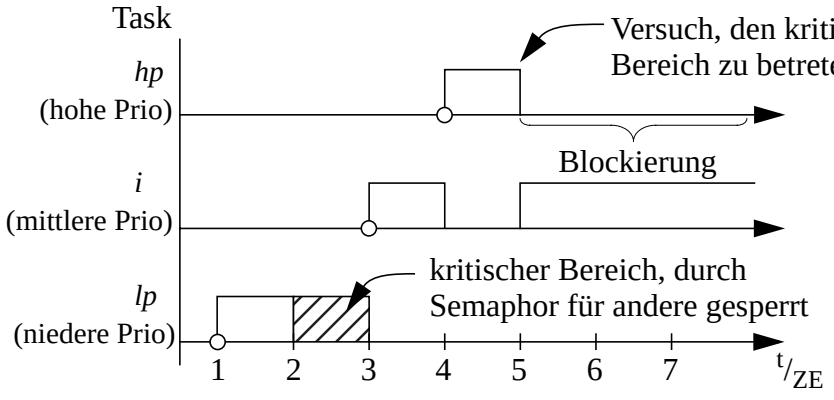


Bild 3-8: Prioritätsumkehrung

$t = 2$ den kritischen Bereich. Durch das Setzen eines Semaphors wird weiteren Tasks der Zugang zu diesem Bereich verwehrt. Wird diese Task durch die Task mittlerer Priorität unterbrochen, dann bleibt der kritische Bereich weiterhin gesperrt. Wird nun wiederum diese Task durch die Task höchster Priorität unterbrochen und diese beansprucht zum Zeitpunkt $t = 5$ Zugang zum kritischen Bereich, so ist dieser nicht möglich, da das Semaphor immer noch nicht freigegeben wurde. Somit bekommt Task *i* wieder den Prozessor zugesprochen und kann weiterarbeiten. Treten nun weitere Anforderungen von *i* auf, bevor *lp* den kritischen Bereich freigeben kann, wird *hp* auf unbestimmte Zeit blockiert, es tritt eine Prioritäteninversion ein. Um dieses Problem zu umgehen und um die Dauer einer Blockierung zu begrenzen, existieren folgende Lösungen:

- Das *Priority Inheritance Protocol* (PIP [137]) vermeidet eine Inversion, indem für den Zeitraum einer auftretenden Blockierung die Priorität der blockierten Task (hier *hp*) an die Task weitergegeben („vererbt“) wird, die den kritischen Bereich hält. Damit kann die Task *lp* nicht mehr von Task *i* unterbrochen werden und erhält so die Möglichkeit, den kritischen Bereich wieder freizugeben. Durch dieses Vorgehen wird außerdem die Blokkierungszeit für *hp* begrenzt, da die Blockierung maximal einmal pro kritischem Bereich jeder niederprioren Task auftreten kann. Allerdings gilt dies natürlich für sämtliche kritischen Bereiche, die von einer Task beansprucht werden, wodurch unter Umständen doch sehr große Antwortzeiten entstehen können.

- Das zuletzt angesprochene Problem, daß alle niederprioren Tasks einen kritischen Bereich einmal blockieren können, kann durch das *Priority Ceiling Protocol*, PCP [137] umgangen werden. Hierbei wird für jeden kritischen Bereich ein Semaphor angelegt, dem ein bestimmer Maximal- oder Dachwert (*ceiling*) zugeordnet wird. Dieser Dachwert entspricht dem Maximum der Prioritäten der Tasks, die auf den entsprechenden kritischen Bereich zugreifen. Jeder Task des Systems wird außerdem eine statische, vor der Systemlaufzeit ermittelte und eine dynamische Priorität zugeordnet. Die dynamische Priorität wird in dem Moment gesetzt, wenn eine Task einen kritischen Bereich beansprucht: Sie ist bildet das Maximum aus statischer Priorität der Task und dem Dachwert des Semaphors. Ist kein kritischer Bereich gesperrt, entspricht der statische dem dynamischen Prioritätenwert. Eine Task darf einen kritischen Bereich nur betreten, wenn der dynamische Prioritätswert der Task größer ist, als der Dachwert des kritischen Bereichs. Daraus resultiert, daß kein kritischer Bereich gesperrt werden kann, der von einer höherprioren Task ebenfalls benutzt wird.

Durch Verwendung des *Priority Ceiling Protocols* kann somit die Blockierungsdauer durch niederpriore Tasks bei der Verwendung kritischer Bereiche auf folgenden Wert beschränkt werden:

$$B_i = max_{j \in lp(i)}(cs_j) \tag{3-12}$$

Dabei gibt cs_j die Dauer der jeweiligen Blockierung an. Berücksichtigt man diese Blockierung *B*, so addiert sich zur Antwortzeit nach Gl. 3-9 außerdem der entsprechende Summand. Sie berechnet sicht dann wie folgt:

$$r_i = B_i + C_i + I_i \tag{3-13}$$

Berechnung der Worst-case-Antwortzeit unter Berücksichtigung von unterschiedlichen Anfangszeiten

Die im vorangehenden Kapitel vorgestellte Methode geht von der Annahme aus, daß alle Anforderungen zum gleichen Zeitpunkt periodisch in das System eintreten. In realen Systemen kann aber der Fall eintreten, daß ein bestimmtes Zeitfenster existiert, innerhalb dessen eine Task gestartet wird. In der Literatur werden dabei *Jitter* und *Offset* unterschieden [128], [114]. Ein Jitter *J* kann z.B. durch die Ungenauigkeit von Zeitnormalen oder durch eine zustandsabhängige Bearbeitungszeit einer Task auftreten, so daß eine Schwankung um den erwarteten Wert berücksichtigt werden muß. Ein Offset *O* verschiebt den Startpunkt um einen fest vorgegebenen Wert. Die Auswirkungen beider Effekte auf die Antwortzeit sind gleich, da der mögliche Jitter in bezug auf die Worst-case-Betrachtung als Verlängerung und damit als Offset für die Antwortzeit betrachtet werden kann. Beide werden im folgenden unter dem Begriff Offset *O* behandelt. Berücksichtigen muß man nun noch, daß der Offset, den die betrachtete Task erfährt, keinen Beitrag zur Interferenz leisten kann und deshalb getrennt addiert werden muß. Lehoczky [108] übernimmt zur Berechnung dieser Antwortzeit die Betrachtung der Aktivitätszeit (busy period) *P* aus der Nachrichtenverkehrstheorie. Sie gibt die Belegung des Prozessors durch die betrachtete Task *i*, auftretende Blockierungen und die Interferenz aller höherprioren Tasks *hp(i)* wieder. Somit wird die Berechnung der Worst-case-Antwortzeit entsprechend erweitert zu:

$$r_i = O_i + P_i \tag{3-14}$$

mit

$$P_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{O_j + P_i}{T_j} \right\rceil \cdot C_j \tag{3-15}$$

Antwortzeitanalyse für Anforderungen mit Antwortzeiten größer als die Periode

Der bisherige Ansatz hat die Beschränkung, daß sich nur Antwortzeiten berechnen lassen, die kürzer oder gleich der Periode sind. Läßt man Anforderungen zu, deren Antwortzeit größer ist als ihre Periode ($R_i > T_i$), dann liegt das Problem in der jetzt notwendigen Mitberücksichtigung neuer Anforderungen der Task, da deren Bearbeitung im schlechtesten Fall länger als die der vorangegangenen Anforderung dauern kann. In diesem Fall würde die neue Anforderung die Worst-case-Antwortzeit bewirken. Die Konsequenz daraus ist, daß alle Anforderungen einer derartigen Task untersucht werden müssen. Dies wird in der bisherigen Berechnung der Worst-case-Antwortzeit nicht berücksichtigt.

Bild 3-9 veranschaulicht die Situation einer Task *i*, deren Antwortzeit größer als ihre Periode ist. Die Aussage der busy period wird dazu so erweitert, daß sie die kontinuierliche Belegungszeit eines Prozessors durch alle Anforderungen einer betrachteten Task *i* beschreibt (Interferenz und Blockierung bleiben gleich). Da alle nachfolgenden Anforderungen der Task *i* in Betracht gezogen werden müssen, erhält man eine Folge von Aktivitätszeiten. Ein derartiges Szenario mit aufeinanderfolgenden Anforderungen resultiert z.B. aus der Segmentierung einer großen Datenmenge in mehrere übertragbare Datenpakete, die, mit derselben Priorität versehen, über ein Kommunikationsmedium verschickt werden. Die Folge der betrachteten Aktivitätszeiten kann abbrechen, sobald eine Anforderung innerhalb ihrer Periode beendet wird. Die letzte Aktivitätszeit dieser Folge ist die maximale Aktivitätszeit (im Bild ist dies P_4). Nach dieser Aktivitätszeit wird zum ersten Mal der Prozessor für die niederpriore Task *lp* freigegeben. Um die verschiedenen Anforderungen und Aktivitätszeiten zu unterscheiden, führt man den

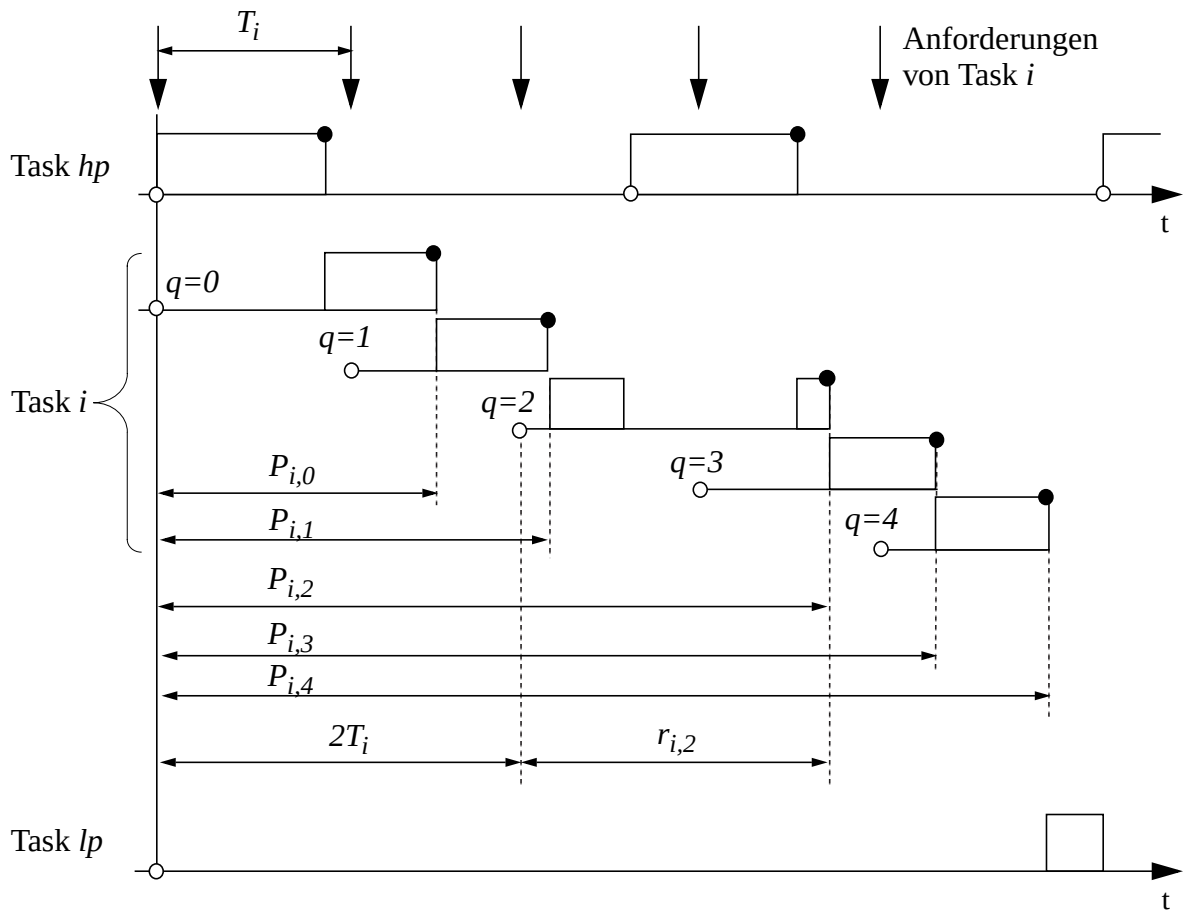


Bild 3-9: Darstellung von Aktivitätszeiten P_i für die Task *i*. Zur besseren Übersicht sind die Anforderungen von Task *i* in mehreren Stufen dargestellt. Ankommende Anforderungen der Task *i*, die nicht sofort bearbeitet werden können, werden zwischengespeichert.

Index *q* ein. Dieser Index startet mit dem Wert $q = 0$ und wird für jede Anforderung inkrementiert. Jede Anforderung besitzt ihre eigene Antwortzeit. Um diese berechnen zu können, benötigt man die zugehörige Aktivitätszeit, den Anforderungsindex *q* und die Periode *T*. Die Antwortzeit $r_{i,2}$ berechnet sich z.B. wie folgt:

$$r_{i,\,q\,=\,2} \;=\; P_{i,\,q\,=\,2} - (2 \cdot T_i) \tag{3-16}$$

Zur Bestimmung der Worst-case-Antwortzeit berechnet man für jede Anforderung die Antwortzeit und ermittelt davon den Maximalwert.

$$r_i \;=\; \max_{q=0,1,2,..}(P_{i,\,q} - qT_i) \tag{3-17}$$

Betrachtet man alle Antwortzeiten in Bild 3-9, so erkennt man, daß $r_{i,2}$ den größten Wert hat und damit die gesuchte Worst-case-Antwortzeit ist. Die Berechnungsfolge kann abbrechen, sobald die Wartezeit der *q*-ten Anforderung innerhalb der nächsten Periode bearbeitet werden kann, d.h.:

$$b_{i,\,q} \leq (q + 1)T_i \tag{3-18}$$

Um die Worst-case-Antwortzeit zu bestimmen, benötigt man noch die Aktivitätszeit $P_{i,q}$:

$$P_{i,q} = qC_i + C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{P_{i,q}}{T_j} \right\rceil C_j \tag{3-19}$$

Durch den Term qC_i werden die Anforderungen berücksichtigt, die zwischen dem kritischen Zeitpunkt und der betrachteten Anforderung aufgetreten sind. Zieht man außerdem den möglichen Offset in Betracht, dann ergeben sich für unterbrechende Systeme folgende Gleichungen zur Berechnung der Worst-case-Antwortzeit:

$$r_i = \max_{q=0,1,2,..}(O_i + P_{i,q} - qT_i) \tag{3-20}$$

mit

$$P_{i,q} = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{O_j + P_{i,q}}{T_j} \right\rceil C_j \tag{3-21}$$

3.5.3.2 Die Worst-case-Antwortzeit in nichtunterbrechenden Systemen

In der bisherigen Analyse wurde von Systemen ausgegangen, die mit unterbrechbaren Tasks arbeiten. Nun sollen nichtunterbrechbare (non-preemptive, *nup*) Systeme betrachtet werden. Im Vergleich zur oben aufgeführten Blockierung durch kritische Bereiche bei unterbrechenden Systemen liegt hier ein Blockieren durch das Besetzen von Betriebsmitteln von niederprioren Tasks vor.

Im schlechtesten Fall kann eine Task *i* für die maximale Bearbeitungszeit aller niederprioren Tasks blockiert werden, wenn diese niederpriore Task kurz vor der betrachteten Task das Betriebsmittel zugesprochen bekommt:

$$B_{i,nup} = max_{j \in lp(i)}(C_j) \tag{3-22}$$

Dabei steht *lp(i)* für die Menge aller niederprioren Tasks relativ zur Task *i*.

Die Bestimmung der Antwortzeit muß nun gegenüber Gleichung 3-20 leicht modifiziert werden, da die Wartezeit in dem Moment endet, in dem der betrachteten Task der Prozessor zugeteilt wird. Da die Bearbeitung der Task nicht unterbrechbar ist, müssen keine neuen Anforderungen von höherprioren Tasks in der Bearbeitungszeit berücksichtigt werden. Für die Bestimmung des Index *q* für den Worst-case der Wartezeit gilt wiederum:

$$W_{i,q,nup} \leq (q+1)T_i \tag{3-23}$$

Damit erhält man für die Antwortzeit:

$$r_{i,nup} = \max_{q=0,1,2,..}(O_i + W_{i,q,nup} - qT_i + C_i) \tag{3-24}$$

Anstelle der in den unterbrechenden Systemen betrachteten Aktivitätszeit kann hier direkt die Wartezeit der Anforderungen berechnet werden:

$$W_{i,q,nup} = qC_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{O_j + W_{i,q}}{T_j} \right\rceil C_j \tag{3-25}$$

Ein weiterer Unterschied zu den unterbrechenden Systemen besteht in der Ermittlung der Wartezeit *W*. Da diese nur bis zum Start der Task dauert und dann keine weiteren Beiträge zu dieser Zeit kommen können, kann die Bearbeitungszeit nicht als Startwert benutzt werden. Auch der Blockierungswert *B* kann nicht verwendet werden, da die Task mit der niedrigsten Priorität nicht blockiert wird. Somit wäre in diesem Fall der Startwert 0. Mit einem Offset von ebenfalls 0 würde sich keine Iteration ergeben und die Wartezeit konstant auf 0 bleiben. Berücksichtigt man die Tatsache, daß nach dem kritischen Zeitpunkt in jedem Fall die höherprioren Tasks wenigstens einmal zur Abarbeitung kommen, könnte man die Bearbeitungszeit einer höherprioren Task als Startwert verwenden. Allerdings würde die höchstpriore Task somit den Startwert 0 besitzen. Verbindet man beide Möglichkeiten, so kann als Startwert zur Berechnung für die Task *i* die Summe aus dem Blockierungswert und der Bearbeitungszeit der nächsthöherprioren Task, d.h. P_{i+1} verwendet werden. Somit werden beide Extremfälle abgedeckt und der Startwert 0 kann nur noch bei einem Ein-Task System auftreten, was in der korrekten Wartezeit $W = 0$ für die höchstpriore Task resultieren würde.

$$W^{0}_{i,\,0,\,nup} = B_i + C_{i-1} \tag{3-26}$$

Mit Hilfe der in den letzten Abschnitten vorgestellten Beziehungen ist man in der Lage, quantitative Aussagen über das Worst-case-Verhalten von Tasks auf einem Prozessor zu machen. Diese Methode bildet die Grundlage für die Garantie des Echtzeitverhaltens eines Systems und wird in Kapitel 5 auf verteilte Mehrprozessorsysteme erweitert.

3.6 Bestimmung der Task-Bearbeitungszeiten

Eine Grundlage für die Berechnung der Antwortzeiten von Tasks ist das Wissen über die Bearbeitungszeit *C* der Tasks. Genauer gesagt muß die maximal mögliche Bearbeitungszeit bekannt sein. Daß dies keine triviale Aufgabe ist, hängt mit den in Programmiersprachen vorhandenen Konstrukten für die Steuerung des Programmablaufes zusammen. Kontrollflußelemente wie die bedingte Verzweigungen (if-Abfragen, Schleifenbedingungen) lassen, je nach momentanem Zustand bestimmter Variablen, mehrere Alternativen für den Programmfluß zu, die sich in unterschiedlichen Laufzeiten der Task äußern.

Über die Bestimmung der Task-Bearbeitungszeiten wurden schon mehrere Untersuchungen durchgeführt (siehe [155, 124]). Ziel ist dabei immer, bestimmte Konstrukte, wie z.B. Schleifen ohne Begrenzung oder Sprünge im Programm, zu vermeiden, um den Kontrollfluß genau festlegen zu können. Treten trotzdem mehrere Alternativen im Kontrollfluß auf, wird der zeitlich länger dauernde Zweig für die Bestimmung der maximalen Bearbeitungszeit verwendet (der Worst-case-Fall der Bearbeitungszeit).

Die Bestimmung der Arbeitszeit kann auf mehreren Ebenen erfolgen, je nach Kenntnisstand über das verwendete System. Liegt kein umfangreiches Wissen über das System vor, dann kann eine zeitliche Abschätzung nur auf der Ebene einer höheren Programmiersprache erfolgen. Kennt man dagegen die verwendeten Prozessoren und insbesondere auch die Compiler, kann eine sehr genaue Angabe der Bearbeitungszeit auf Maschinensprachenebene stattfinden. Eventuelle Optimierungsvorgänge der Compiler, bzw. Architektureigenschaften der Prozessoren, wie z.B. die Fließbandverarbeitung (Pipelining), müssen dabei gesondert beachtet werden.

Kapitel 4

Grundlagen der Systemoptimierung

Im vorliegenden Kapitel werden Grundlagen und Methoden der Systemoptimierung vorgestellt. Mit diesen Verfahren ist es möglich, bestehende Systeme, durch geeignete Veränderung bestimmter Systemparameter, zu verbessern (Parameteroptimierung). Optimierungsmethoden werden oft mit den Aktivitäten des Operations Research (OR) gleichgestellt. Deren vordringlichstes Ziel ist es, bezüglich eines Systems und seiner Veränderung, „optimale Entscheidungen zu treffen und zu deren Findung deterministische oder probabilistische Modelle der Realität zu verwenden“ [65]. Dabei ist das Anwendungsfeld für diese Methoden kaum eingeschränkt, wobei sich nicht jede Methode für jedes Problem gleichermaßen eignet.

Eine wichtige Aufgabe bei der Optimierung ist die Festlegung des Optimierungsziels. Notwendig dafür ist ein Qualitätsmaß, das ausdrückt, wie gut ein System die gestellten Anforderungen erfüllt. Hat man das Optimierungsziel definiert, kann man versuchen, mit Hilfe bestimmter Algorithmen dieses Ziel zu erreichen.

Nach einer Taxonomie von Optimierungsalgorithmen und einer Diskussion der Komplexität von Optimierungsproblemen sollen vor allem die naturanalogen und insbesondere die evolutionären Algorithmen vorgestellt werden, da deren Einsatz für das in dieser Arbeit diskutierte Planungsproblem der Prioritätenvergabe in Steuergerätenetzen den größten Erfolg verspricht.

4.1 Quantitative Erfassung der Optimierungsziele

Sollen Systeme optimiert werden, so ist es zunächst unabdingbar, eine Aussage darüber zu machen, wie gut das System ist („Ist“-Zustand). Gibt es in dieser Aussage eine negative Differenz zu einem gewünschten Systemzustand („Soll“-Zustand), dann muß das System so modifiziert werden, daß dieser Soll-Zustand erreicht wird, soweit dies nach Berücksichtigung der Randbedingungen überhaupt möglich ist. Ist der Systemzustand von Systemparametern abhängig, so kann versucht werden, aus diesen Parametern eine quantitative Aussage über das System zu erhalten. Man spricht dabei von den „Kosten“, die für das System zur Erbringung des gewünschten Systemverhaltens aufgebracht werden müssen, oder auch von der „Qualität“ des Systems. Funktionen, die eine Bewertung des Systemzustands erlauben oder eine Aussage darüber zulassen, wie gut der Systemzustand äußere Bedingungen an das System erfüllt, werden somit Qualitätsfunktion, Kostenfunktion oder auch Zielfunktion genannt, abhängig vom Terminus der verwendeten Optimierungsalgorithmen bzw. -strategien. Es handelt sich dabei jedoch immer um dieselbe, oben vorgestellte Idee der Systemquantifizierung. In dieser Arbeit wird der Begriff der Qualitätsfunktion verwendet. Dabei berechnet sich die Qualität Q eines

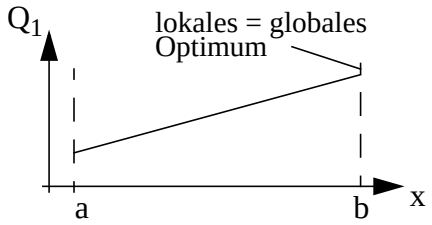
Systems aus dem Systemparametervektor $S = (s_1, s_2, s_3, \ldots s_n)^T$, der ein Element der nichtleeren Parametervektormenge M sein soll. Für M wiederum gilt $M \subseteq R^n$, wobei R^n die Menge aller Vektoren darstellt. Das Idealziel ist, einen Vektor $S^* \in M$ zu finden, so daß gilt:

$$\forall S \in M \qquad Q(S) \leq Q(S^*). \tag{4-1}$$

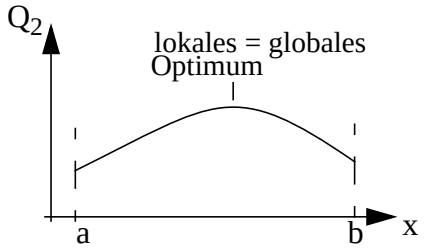
S^* kennzeichnet den optimalen Systemzustand. Die Beschränkung auf die Qualitätsbetrachtung ist ohne Konsequenz für die Allgemeingültigkeit einer Optimierungsstrategie, da gilt:

$$max\{Q(S)\} = -min\{-Q(S)\} \tag{4-2}$$

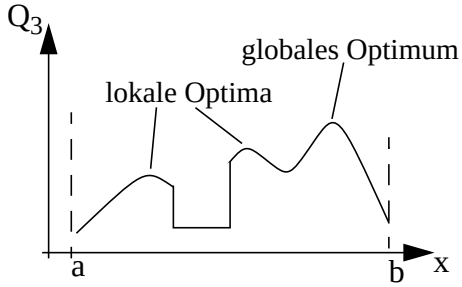
D.h. daß genausogut nach dem Minimum der Kosten gesucht werden könnte, wenn die Kosten das Gegenteil der Qualität bilden. Vereinfachend soll im folgenden außerdem angenommen werden, daß der Zustand eines Systems durch den Systemparametervektor eindeutig beschrieben ist.



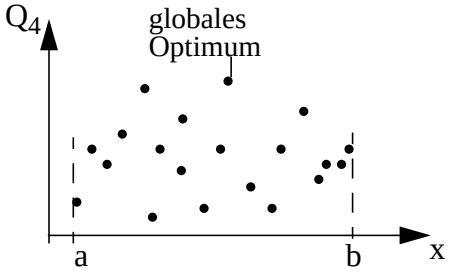
Qualitätsfunktion eines linearen Optimierungsproblems mit einer Variablen. Das Maximum befindet sich immer am Rand des Wertebereichs.



Qualitätsfunktion eines einfachen nichtlinearen (konkaven) Problems mit einem Optimum.



Qualitätsfunktion eines nichtlinearen Problems mit mehreren Optima. Die Funktion ist, bedingt durch die Sprünge, nicht differenzierbar.



Qualitätsfunktion mit diskreten Werten.

Bild 4-1: Beispiele für den Verlauf von Qualitätsfunktionen

Abhängig von Art und Verlauf der Qualitätsfunktion werden zwei grundlegende Optimierungsaufgaben unterschieden: lineare und nichtlineare Optimierungsprobleme, wobei die letzteren die Untermenge der diskreten Probleme abdecken. Der Unterschied und die damit zusammenhängenden Probleme sollen an einfachen Beispielen kurz aufgezeigt werden. Es werden dazu die Qualitätsfunktionen Q_1 bis Q_4 aus Bild 4-1 betrachtet, die jeweils nur von einem Parameter x abhängen. Als Randbedingung gelte $a \le x \le b$. Für den Fall von Q_1 und Q_2 ergeben sich keine Probleme, wenn ein Optimum bestimmt wurde: hier gibt es keinen besseren Wert, da es sich um das *globale Optimum* des Problems im betrachteten Lösungsbereich handelt. Im Fall von Q_3 ist die Sachlage schwieriger. Befindet man sich hier in einem lokalen Optimum, so kann nicht ohne weiteres auf das Vorhandensein eines weiteren, besseren Optimums geschlossen werden. Eine Technik, die in den ersten Fällen zum Erfolg führt, nämlich die Differentiation und die Nachfolge der Steigung der Funktion (Gradientenmethode), versagt in diesem dritten Beispiel. Eine Qualitätsfunktion vom Typ Q_3, mit mehreren lokalen Optima, wird auch als *multimodale* Funktion bezeichnet.

Einen weiteren Schwierigkeitsgrad bringen schließlich Qualitätsfunktionen mit diskreten Werten (siehe Q_4 in Bild 4-1). Hier lassen sich die klassischen Verfahren des Operations Research nur sehr bedingt anwenden; man muß hierbei häufig auf Suchverfahren und Näherungsmethoden (siehe Kapitel 4.4) ausweichen.

4.2 Optimierungsalgorithmen und -methoden

Die Zahl der unterschiedlichen Optimierungsansätze ist aufgrund der unterschiedlichen Optimierungsprobleme sehr groß; es gibt keinen allgemeingültigen Optimierungsalgorithmus, der für alle Probleme gleich gut anwendbar ist. Bild 4-2 zeigt eine mögliche Klassifizierung derartiger Algorithmen und gibt deren typische Vertreter an. Der dunkel hinterlegte Bereich kennzeichnet zusätzlich die Verfahren, die für große kombinatorische Optimierungsprobleme verwendbar sind.

Zunächst werden die beiden bereits genannten Hauptgruppen, lineare Optimierung und die nichtlineare Optimierung, unterschieden. Alle Verfahren der *linearen Optimierung* gehen von folgendem Grundproblem (Normalform der linearen Optimierung) aus: maximiere die Qualitätsfunktion *Q:*

$$max\left(Q(S) = \sum_{i=1}^{n} c_i s_i \right) \tag{4-3}$$

Dabei soll wiederum der Variablenvektor $S = (s_1, s_2, s_3, \ldots s_n)^T$ so bestimmt werden, daß die Qualitätsfunktion, die sich aus einer Summe linearer Terme berechnen läßt, ihr Maximum annimmt. Außerdem sind dabei m Nebenbedingungen (Restriktionen) zu beachten, für die jeweils gilt:

$$\sum_{i=1}^{n} a_{ji} s_i \le b_j \qquad \text{mit } j \in \{1 \ldots m\} \tag{4-4}$$

wobei $s_i \ge 0$ (Nichtnegativitätsbedingung) gilt und die Variablen voneinander unabhängig sind. Eine *zulässige* Lösung erfüllt dabei alle Nebenbedingungen, die *optimale* Lösung ist die

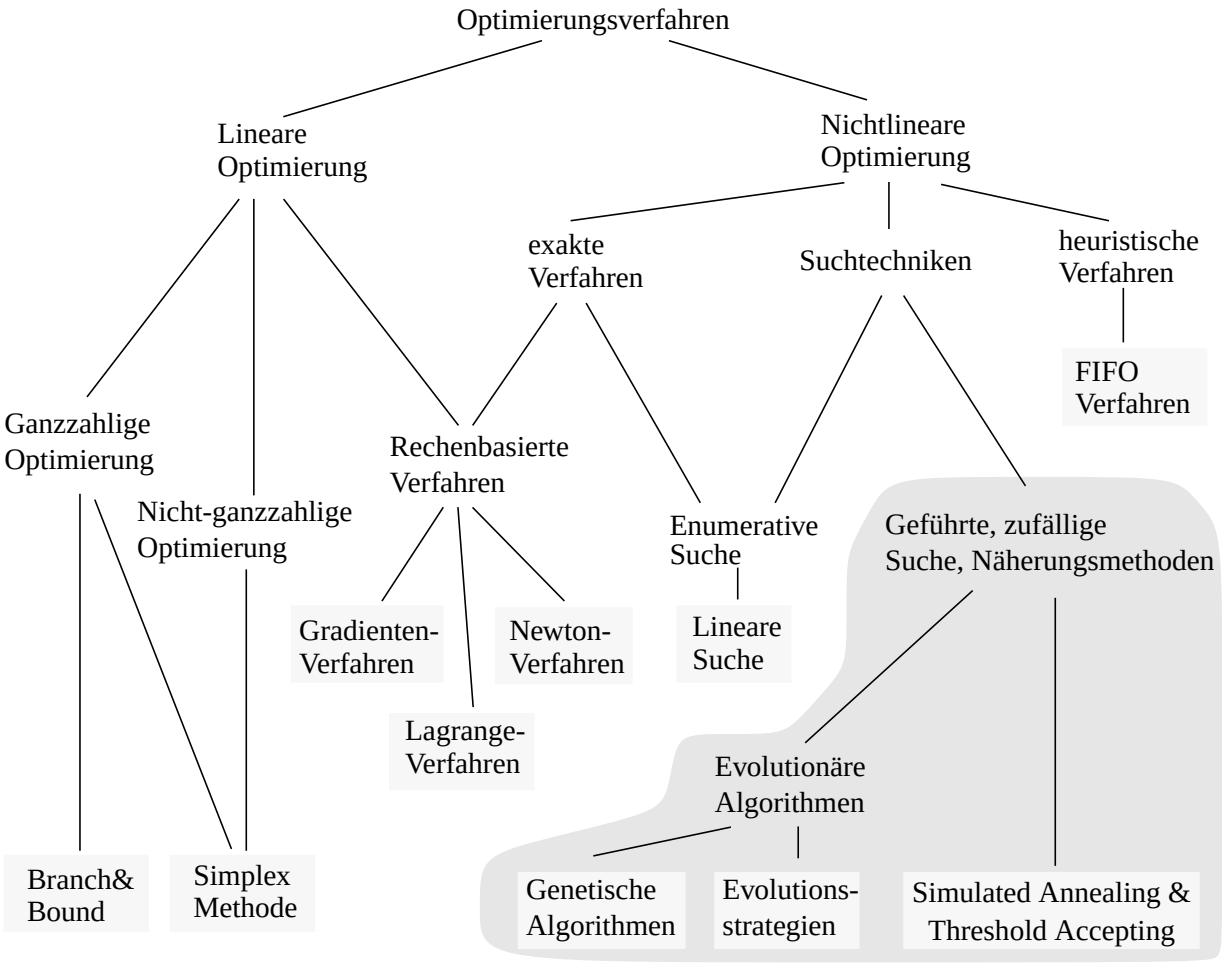


Bild 4-2: Taxonomie der Such- und Optimierungsalgorithmen. Es werden außerdem typische Vertreter angegeben. Der dunkel hinterlegte Bereich kennzeichnet Verfahren, die für große kombinatorische Optimierungsprobleme verwendbar sind.

Lösung, die den besten Qualitätswert überhaupt ergibt. Diese Festlegung gilt für alle Optimierungsprobleme.

Die lineare Optimierung läßt sich weiter in die ganzzahlige Optimierung (*integer programming*), für die $s_i \in N$ gilt, und die nicht-ganzzahlige Optimierung (*non-integer programming*) unterteilen, für die jeweils spezielle Lösungsalgorithmen bestehen. Einer der bekanntesten Algorithmen der linearen Optimierung ist dabei das *Simplex Verfahren* [65]. Es kann sowohl für ganzzahlige Probleme als auch für nicht-ganzzahlige Probleme verwendet werden. Dabei wird bei diesem Verfahren davon ausgegangen, daß das zu suchende Optimum ein Eckpunkt des durch die linearen Gleichungen aufgespannten Lösungsraums (Simplex) ist. Das Problem besteht darin, an den Rand dieses Lösungsraums zu gelangen und einen gültigen Eckpunkt zu finden. Von dort aus werden die benachbarten Eckpunkte betrachtet und dann als bessere Lösung akzeptiert, wenn deren Qualität besser ist, als die des aktuell besetzten Ecks. Ein weiterer typischer Vertreter für die ganzzahligen Algorithmen ist das *Branch & Bound* Verfahren [65, 26].

Der Vorteil der beiden angesprochenen Methoden besteht darin, daß mathematisch bewiesen werden kann, daß eine durch sie gefundene Lösung gleichzeitig das globale Maximum ist. Der Nachteil besteht allerdings darin, daß im ungünstigsten Fall alle möglichen Lösungen über-

prüft werden müssen. Diese Tatsache macht diese Methoden für große kombinatorische Probleme (siehe Kapitel 4.3), durch die immense Zahl an zu prüfenden Lösungen, ungeeignet.

Leider sind viele der aktuellen Optimierungsprobleme (auch das in dieser Arbeit betrachtete Prioritäten-Optimierungsproblem) nicht auf die Form nach Gleichung 4-3 rückführbar. Es handelt sich hierbei um nichtlineare Probleme. Da es dafür keine Normalform gibt, wird nur von der allgemeinen Qualitätsfunktion ausgegangen, die maximiert werden soll (die lineare Programmierung ist als Untermenge der nichtlinearen Optimierung ebenfalls enthalten):

$$max[Q(S)] \tag{4-5}$$

mit $S = (s_1, s_2, s_3, \ldots s_n)^T$ und den Nebenbedingungen

$$h_j(\vec{s}) \leq b_j \tag{4-6}$$

für $j = 1, 2, ..., m$ und $s_j \geq 0$. Es existiert bisher kein Algorithmus, der jedes spezifische Problem, für das dieses allgemeine Format zutrifft, lösen kann. Exakt berechenbare Lösungen gibt es nur für Spezialfälle, bei denen z.B. keine oder nur eine Nebenbedingung zugelassen sind und weiterhin nur eine oder wenige Systemparameter erlaubt werden. Zu diesen Verfahren gehören u.a. das Newton- und das Lagrange-Verfahren [65].

Die Gruppe der kombinatorischen Optimierungsprobleme läßt sich durch diese Methoden i.d. R. nicht lösen. Hier bieten Heuristiken eine bessere Möglichkeit. D.h. man verwendet bestimmte Regeln, die sowohl aus der Systembeobachtung als auch aus plausibel scheinenden Annahmen gewonnen wurden. Das „First-in, first-out“-Verfahren [2] zur Lagerbestandsoptimierung ist dabei ein typischer Vertreter: Man entfernt die Elemente wieder aus einem Speicher, die zuerst in den Speicher abgelegt wurden. Derartige Verfahren kommen meist ohne große Berechnung aus, sind aber i.a. nicht besonders effizient. Eine weitere Alternative bieten die intelligenten Suchtechniken. Eine lineare Suche, die in jedem Fall das globale Optimum für ein System liefern würde, in dem jeder Zustand, den das System einnehmen kann, überprüft wird (exaktes Verfahren), scheidet in den meisten Fällen der kombinatorischen Probleme aus, da der Suchraum schnell zu groß wird. Ein Bussystem, wie in Kapitel 2 vorgestellt, mit 50 verschiedenen Botschaften, erlaubt $50! = 3 \times 10^{64}$ Möglichkeiten, die Prioritäten zu vergeben. Da eine vollständige Berechnung damit ausscheidet, müssen andere Methoden angewendet werden, die meist aus einer zufallsgesteuerten Komponente und einer deterministischen Komponente zusammengesetzt sind. Solche Methoden werden, nach einer Komplexitätsbetrachtung kombinatorischer Probleme, in den nachfolgenden Kapiteln vorgestellt.

4.3 Kombinatorische Probleme und die *NP*-Vollständigkeit

Die schwierigsten Probleme der Optimierung sind solche, die zum einen durch die Anzahl ihrer Zustände ein sequentielles, lineares Absuchen des Lösungsraumes praktisch unmöglich machen und zum anderen ein nichtlineares Verhalten der Qualitätsfunktion aufweisen und somit nicht geschlossen berechenbar sind. Wie das im vorhergehenden Abschnitt gezeigte Beispiel der Prioritätsverteilung eindrücklich darlegt, helfen hier auch keine Parallelisierungsmöglichkeiten: durch sie könnte die Potenz um zwei bis maximal drei Einerstellen verkleinert werden (1000 parallel betriebene Maschinen!). Dies fällt jedoch bei dem zu leistenden Gesamtaufwand überhaupt nicht ins Gewicht.

Zur Charakterisierung der Schwierigkeit eines Problems wird allgemein der Aufwand betrachtet, der zur Ermittlung einer gültigen Lösung notwendig ist. („Gültige Lösung“ steht im

Gegensatz zu einer ungültigen Lösung, die zwar einen bestimmten Systemzustand repräsentiert, dabei aber vorgegebene Randbedingungen verletzt.) Es werden dazu zwei Grundmengen an Problemen unterschieden, für die gültige Lösungen gesucht werden:

- Probleme, die der Menge P (*polynomial*) zugeordnet werden, können mit Hilfe eines deterministischen Algorithmus in polynomialer Zeit gelöst werden. Deterministisch bedeutet hier, daß zu jedem Zeitpunkt der Bearbeitung durch den Algorithmus feststeht, was er als nächstes tun wird, d.h. es treten keine zufälligen Ereignisse ein. Als Lösungsalgorithmen fallen hierunter die exakten Verfahren und die Verfahren der Linearen Optimierung. Solche Probleme werden auch als „leicht“ (*easy*) bezeichnet [80].
- Probleme, die der Menge NP (*nondeterministic polynomial*) angehören, können durch nichtdeterministische Algorithmen in polynomialer Zeit gelöst werden. Den Ablauf kann man sich in zwei Schritte aufgeteilt vorstellen: zuerst wird mittels nichtdeterministischer Suche eine Lösung bestimmt. Im zweiten Schritt wird in polynomialer Zeit berechnet, ob es sich um eine gültige Lösung handelt. Zur Lösung dieser Probleme können die Algorithmen der geführten, zufälligen Suche verwendet werden. Derartige Probleme werden mit „schwer“ (*hard*) charakterisiert.

Polynomiale Zeit bedeutet, im Gegensatz zu einer exponentiellen Zeit, daß der Algorithmus eine bestimmte Bearbeitungszeit benötigt, die proportional zum Wert eines Polynoms ist, dessen Grad k nicht von der Zahl der Eingabeparameter n abhängt (bei minimaler Codierung). Weiterhin gilt $P \subset NP$, d.h. alle Probleme, die P angehören, gehören auch NP an. Wie groß P im Bezug auf NP ist, oder ob sie gar gleich sind, konnte bisher nicht nachgewiesen werden und stellt deshalb eines der großen Probleme der Komplexitätstheorie dar [135].[1]

Bestimmte Probleme der NP-Menge haben die Eigenschaft, daß, falls für irgendeines dieser Probleme der Nachweis der Zugehörigkeit zur Menge P gelingt, alle diese Probleme zu P gehören würden. Solche Probleme werden als NP-vollständig (*NP-complete*) bezeichnet. In der Regel besteht der Nachweis der NP-Vollständigkeit darin, zu zeigen, daß das gegebene Problem auf ein bekanntes NP-vollständiges Problem polynomial reduzierbar ist. Das bedeutet, daß eine Instanz des bekannten Problems in eine Instanz des neuen Problems transformiert wird, wobei die Transformation in höchstens polynomialer Zeit (s.o.) ausgeführt werden darf. Dieser Nachweis kann allerdings nur gelingen, wenn überhaupt ein NP-vollständiges Problem existiert. Dies wurde 1971 von Cook in Form des sog. „SAT“-Problems[2] gefunden und von Garey und Johnson nachgewiesen [42].

Ausgehend von diesem ermittelten Grundproblem wurden weitere *NP*-vollständige Probleme gefunden, die als Referenz für andere Probleme Verwendung finden. Für diese Probleme gilt immer die oben aufgeführte Eigenschaft, daß ein nichtdeterministischer Algorithmus eine Lösung ermitteln muß, die daraufhin in polynomialer Zeit überprüft werden kann. Zu diesen Referenzproblemen gehören z.B. der Hamilton-Zyklus „HC“ und davon abgeleitet das Problem des Handlungsreisenden „TSP“ und schließlich das Partition-Problem. Die Anforderung an das Partition-Problem ist, eine Menge S von Elementen s so in zwei Teilmengen S' und $S\text{-}S'$ aufzuspalten, daß die Summe über ein Maß $r(s)$ dieser Elemente in beiden Mengen gleich groß ist: $\sum_{s \in S'} r(s) = \sum_{s \in S - S'} r(s)$. Das hier diskutierte Problem des Echtzeitschedulings in einem verteilten System läßt sich auf dieses Problem reduzieren (Flow-Shop Problem SS15 in [42][3]). Es ist sogar für nichtperiodische Systeme für $n = 3$ Prozessoren bereits *NP*-vollständig.

1. Allerdings wird die Wahrscheinlichkeit, daß dies zutrifft, als sehr gering eingestuft.
2. Satisfiability-Problem
3. In [42] findet sich eine ausführliche Liste von bekannten *NP*-vollständigen Problemen.

4.4 Naturanaloge Verfahren

Naturanaloge Verfahren sind Strategien, bei denen Abläufe, wie sie in der Natur zu beobachten sind, nachgebildet werden. Hierzu zählt das simulierte Ausglühen (*simulated annealing*), bzw. das Toleranzschwellenverfahren (*threshold accepting*) sowie die simulierte Evolution. Alle Verfahren haben das gemeinsame Merkmal, daß zunächst keine spezifischen Anforderungen an die Qualitätsfunktion gestellt werden und daß prinzipiell kein Wissen über die Qualitätsfunktion in den Lösungsalgorithmus einfließt, da es sich um nichtanalytische Verfahren handelt.[4] Der Nachteil besteht allerdings darin, daß keine Aussage gemacht werden kann, wie gut, im Bezug auf andere mögliche Lösungen, eine gefundene Lösung tatsächlich ist, d.h. ob nur ein lokales Optimum gefunden wurde. Es kann weder mathematisch bewiesen noch näherungsweise abgeschätzt werden, wie gut die Lösung ist oder ob vielmehr die optimale Lösung für das Problem gefunden wurde.

4.4.1 Simuliertes Ausglühen und das Toleranzschwellenverfahren

Beim simulierten Ausglühen (*Simulated Annealing, SA*) wird ein thermodynamischer Prozeß der Natur nachgebildet. Dabei kopiert man den Effekt, daß sich Atome bei bestimmten Voraussetzungen in einem Material so anordnen, daß sie einen Zustand mit niedrigster freier Energie annehmen. Dieser Vorgang ist Grundlage für das zweite Gesetz der Thermodynamik. Allerdings wird dabei selten das tatsächliche Energieminimum erreicht. Dies gelingt nur in einem sehr langen und langsamen Abkühlungsprozeß. Diese Eigenschaft wird beim sogenannten „Ausglühen“, speziell bei der Metall- und Glasverarbeitung, ausgenutzt, um sprödes Verhalten zu vermeiden. Hierbei wird ein Festkörper erhitzt und anschließend extrem langsam abgekühlt. Dabei nähert sich das Material von selbst, durch die Wärmebewegung der Atome, einem Zustand mit niederer freier Energie; die Atome ordnen sich dabei in der für das jeweilige Material typischen Kristallstruktur an.

Überträgt man diesen Ausglühvorgang auf ein Optimierungsproblem, dann entsprechen die Atome den änderbaren Parametern des zu optimierenden Systems. Ein Parametervektor S repräsentiert damit einen Zustand des Systems. Das zu erreichende Optimierungsziel faßt man als die Energie auf. Das Maß der Änderung der Parameter wird mit der Temperatur des Ausglühvorgangs identifiziert. Bei einer hohen Temperatur kann sich nahezu jeder beliebige Zustand, entsprechend der zufälligen Wärmebewegungen der Atome, einstellen. Diese Bewegungen werden mit Hilfe eines Zufallsprozesses modelliert und die jeweilige Qualitätsfunktion berechnet. Das Absenken der Temperatur bedeutet, daß, ausgehend von einem aktuellen Zustand, die zugelassenen Änderungen für die Parameter immer kleiner werden. Ein neuer Zustand wird sicher akzeptiert, wenn das System dadurch ein geringeres Energieniveau aufweist, d.h. die Qualitätsfunktion besser erfüllt wird. Ergibt sich durch die veränderten Parameter ein schlechterer Wert, so wird dieser mit einer Wahrscheinlichkeit akzeptiert, die davon abhängt, um wieviel schlechter der neue Zustand ist und wie groß die Temperatur, d.h. die noch zugelassene Änderung, ist. Dies führt dazu, daß auch schlechtere Zustände angenommen werden können und somit z.B. ein lokales Minimum durchschritten werden kann. Der Ablauf des „Annealing“-Vorgangs ist in Bild 4-3 beschrieben.

4. In Kapitel 6 werden trotzdem einige Anforderungen an die Qualitätsfunktion formuliert, die aus den Anforderungen des untersuchten Optimierungsproblems resultieren.

```
Ermittle Initialzustand Sn
Ermittle Initialtemperatur T > 0
while( das System hat zuviel Energie )
    while( index < L )
        Wähle einen zufälligen Nachbarzustand Sn+1
        Berechne Qualitätsdifferenz: ΔQ = Q(Sn+1)-Q(Sn)
        if (ΔQ > 0) then Sn = Sn+1 //neu ist besser
        else                       //neu ist schlechter
            if (AkzeptanzWK PA > U) then Sn = Sn+1
        index = index + 1
        Verringere T
    return Sn
```

Bild 4-3: Darstellung des simulierten Ausglühens nach dem Algorithmus von Metropolis [116] in Pseudo Code

Um die Akzeptanzwahrscheinlichkeit P_A zu berechnen, verwendet man folgende Gleichung:

$$P_A = \begin{cases} e^{\frac{-(Q(S_{n+1}) - Q(S_n))}{T}} & Q(S_{n+1}) < Q(S_n) \\ 1 & sonst \end{cases} \tag{4-7}$$

Das Ergebnis muß mit einer gleichverteilten Zufallszahl U verglichen werden und falls P_A größer ist als diese Zahl, wird der neue Zustand angenommen. Dabei wird das Ergebnis der Akzeptanzwahrscheinlichkeit mit fallender Temperatur T immer kleiner, so daß die Wahrscheinlichkeit für ein Annehmen des neuen, schlechteren Zustands immer geringer wird.

Das Problem bei der Anwendung dieses Verfahrens ist die geeignete Wahl der Parameter. Johnson et al. [77] unterscheiden zwei prinzipielle Bereiche für die Parameterwahl: die problemspezifischen und die algorithmenspezifischen Parameter. Bei den problemspezifischen Parametern muß zunächst festgelegt werden, wie der Zustand S_n eines Systems und wie dessen Nachbarzustand S_{n+1} aussieht. Außerdem muß geklärt werden, wie ein Initialzustand gefunden wird und auf welche Weise die Qualität des Systems berechnet wird. Auf der Seite der algorithmenspezifischen Parameter muß sowohl die Initialtemperatur bestimmt werden als auch eine Funktion `Verringere,` die festlegt, wie diese Temperatur bei jedem Schleifendurchlauf verkleinert wird (hier sind beliebige Funktionsverläufe denkbar). Außerdem müssen die Zahl der Schleifendurchläufe L (und damit die minimale Temperatur) sowie das Abbruchkriterium der äußeren Schleife (System hat zuviel Energie) festgelegt werden. Die Gesamtheit der Parameter bezeichnet man als „Cooling Schedule“, da sie für den Verlauf des dynamischen Ausglühvorgangs verantwortlich sind.

Das Toleranzschwellenverfahren (*Threshold Accepting, TA*) kann als vereinfachte Version des simulierten Ausglühens betrachtet werden. Hier wird die Entscheidung, ob ein neuer Zustand angenommen wird oder nicht, mit Hilfe einer Toleranzschwelle t getroffen [33]. Berechnet man die Qualität $Q(S_{n+1})$ für ein, gegenüber dem Vorzustand modifiziertes System, dann wird der neue Zustand nur angenommen, wenn gilt:

$$Q(S_{n+1}) - Q(S_n) > t \tag{4-8}$$

Beim simulierten Ausglühen kann ebenso ein wesentlich schlechterer Zustand, wenn auch mit sehr geringer Wahrscheinlichkeit, angenommen werden, während dies beim Toleranzschwellenverfahren nicht möglich ist. Der Vorteil besteht darin, daß ein geringerer Berechnungsaufwand für die Akzeptanzentscheidung notwendig ist, da keine Zufallszahlen benötigt werden. Kritiker [33] sehen in dem Nichtvorhandensein des Zufalls den großen Vorteil des Toleranzschwellenverfahrens. Allerdings ist dadurch die Wahrscheinlichkeit, in einem lokalen Maximum stecken zu bleiben, größer als beim simulierten Ausglühen, insbesondere, wenn die Zielfunktion extrem nichtlinear ist (z.B. ein zweidimensionaler Lösungsraum, der aus lauter „Zuckerhüten“ besteht, d.h. lokalen Optima mit sehr steilen Flanken). Hier schafft es das Toleranzschwellenverfahren nicht, wieder von einem solchen lokalen Optimum wegzukommen.

4.4.2 Evolutionäre Algorithmen

4.4.2.1 Biologische Grundlagen

Die Idee der Verwendung von evolutionären Algorithmen zur Systemoptimierung beruht auf der Beobachtung, daß sich Lebewesen über mehrere Generationen hinweg an ihre Umgebung anpassen. Der dabei zugrundeliegende Mechanismus der Evolution wurde zuerst von Darwin [27] erkannt. Lebewesen, die aufgrund bestimmter Eigenschaften besonders gut an ihre Umgebung angepaßt sind, haben eine größere Chance auf eine große Zahl von Nachkommen, als diejenigen, die aufgrund ihrer Ausprägungen nicht mit den Randbedingungen ihres Lebensraumes zurechtkommen. Man spricht in diesem Zusammenhang von „survival of the fittest“. Angepaßte Individuen finden z.B. leichter oder mehr Nahrung als ihre Artgenossen, haben keine Freßfeinde (weil sie z.B. durch Tarnung nicht erkannt werden) oder können diesen zumindest entkommen. Die morphologischen und anatomischen Ausprägungen der Individuen sind dabei das Produkt der Umsetzung des genetischen Codes, der im Zellkern einer jeden lebenden Zelle in Form von Chromosomen vorhanden ist. Wird dieser Code ausgelesen, so werden über Transkriptionsmechanismen Proteine (Eiweiße) gebildet, die den baulichen Grundstoff der Lebewesen bilden.

In den folgenden Teilkapiteln soll zunächst der bereits angesprochene genetische Code dargestellt werden. Anschließend wird gezeigt, welche Veränderungen dieses Codes möglich sind und wie sie sich auswirken. Dabei kann jedoch nur einführend auf die komplexen Mechanismen der Natur eingegangen werden. Die Beschreibung beschränkt sich deshalb auf die Erläuterung der biologischen Grundbegriffe, die zu einem allgemeinen Verständnis der technischen Umsetzung vonnöten sind. Schließlich findet eine Vorstellung der aus den biologischen Vorgängen und Mechanismen abgeleiteten Optimierungsstrategien statt.

4.4.2.1.1 Der genetische Code

Die Informationscodierung in Organismen erfolgt über insgesamt drei Stufen. Grundelemente der Codierung sind dabei vier verschiedene Basen. Dreiergruppen (sog. Codone) dieser Basen bilden jeweils ein Codewort, das einer Aminosäure entspricht. Aus der Kombination der insgesamt 20 verschiedenen Aminosäuren werden die Proteine gebildet, die wiederum den Grundstoff für den Aufbau eines Organismus bilden. Für die Bildung eines Proteins werden dabei ca. 1000 Aminosäuren benötigt. Die Gruppe dieser Aminosäuren, welche das Protein codieren, wird als sog. *Gen* bezeichnet. Die Gene wiederum sind sequentiell auf den *Chromosomen* angeordnet. Gene, die den Ort eines Proteins auf dem Chromosom festlegen, können unter-

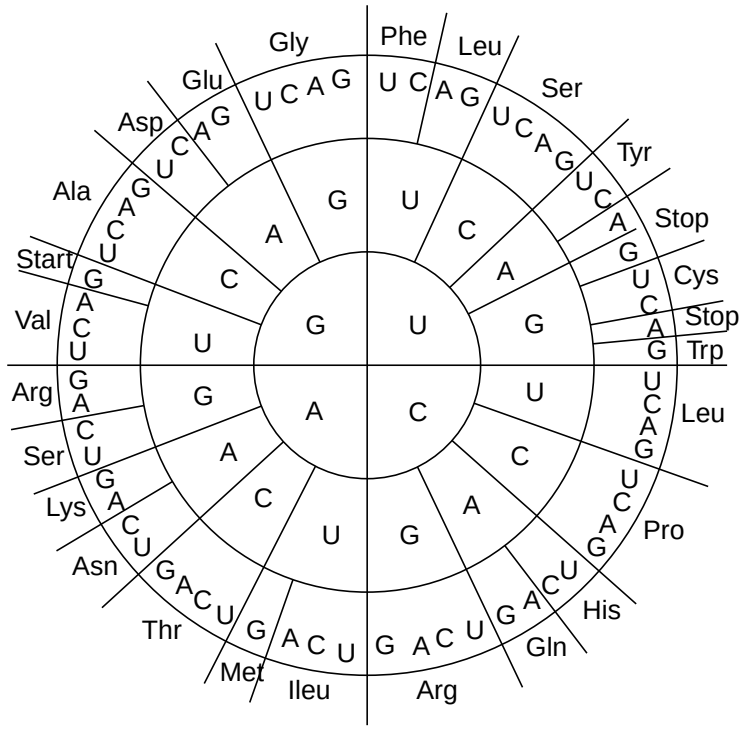


Bild 4-4: Die erste Stufe der genetischen Codierung, der genetische Code. Aus einem Triplett (Codon) der 4 Basen Adenin (A), Cytosin (C), Guanin (G) und Uracil (U) werden die 20 verschiedenen Aminosäuren codiert. Außerdem gibt es zusätzlich zu redundanten Codonen auch solche, die bei der Synthese der Aminosäuren den Anfang (Start) und das Ende (Stop) einer Sequenz kennzeichnen. Die Darstellung einer Sequenz erfolgt von innen nach außen; am Rand stehen die Abkürzungen der codierten Aminosäuren. Darstellung nach [18].

schiedliche Eigenschaften des Gens codieren; man spricht in diesem Zusammenhang von den *Allelen*, die zu einer bestimmten Ausprägung des Gens führen. (Die Allele des Augenfarbengens sorgen z.B. für blaue oder braune Augen.) Weitere Information zu dieser Codierung findet sich u.a. in [100]. Die Gesamtheit der Gene, das sog. *Genom*, ist dabei für die Ausbildung eines bestimmten Individuums (Phänotyp) mit seinen individuellen Eigenschaften verantwortlich. Alle artgleichen Individuen bilden zusammen eine Population; die Gene einer Population werden als Gen-Pool bezeichnet. Auf die, noch eine Stufe darunterliegende, Codierung der Basen aus bestimmten Molekülen und Atomen sowie den nur zulässigen Kombinationen der Basen innerhalb der DNS soll hier nicht weiter eingegangen werden.

4.4.2.1.2 Die Modifikationen der genetischen Information

Betrachtet man die getrenntgeschlechtliche Fortpflanzung von Individuen, so existieren zwei Mechanismen, die dafür sorgen können, daß die Nachkommen andere Eigenschaften aufweisen, als ihre Eltern: die Rekombination, ermöglicht durch die Verknüpfung der Gene der beiden Eltern und die Mutation, die eine weitere Änderung des Erbguts ermöglicht. Die Selektion entscheidet anschließend darüber, ob die neuen Merkmale an nachfolgende Generationen weitergegeben werden können.

Die *Rekombination* ist ein Grundmechanismus für die getrenntgeschlechtliche Merkmalsvererbung und -modifikation. Durch sie ist die genetische Vielfalt möglich. Jeweils gleichartige

(homologe) Chromosomen der Eltern werden zu neuen Chromosomen kombiniert. Dabei ist dieses Kombinieren nur an bestimmten Stellen des Chromosoms, nämlich den Grenzen der Gene, möglich. Hier werden die Chromosomenstücke durch ein *Crossover* ausgetauscht (siehe Bild 4-5). Interessanterweise werden nur bestimmte Stellen der Chromosomen durch Crosso-

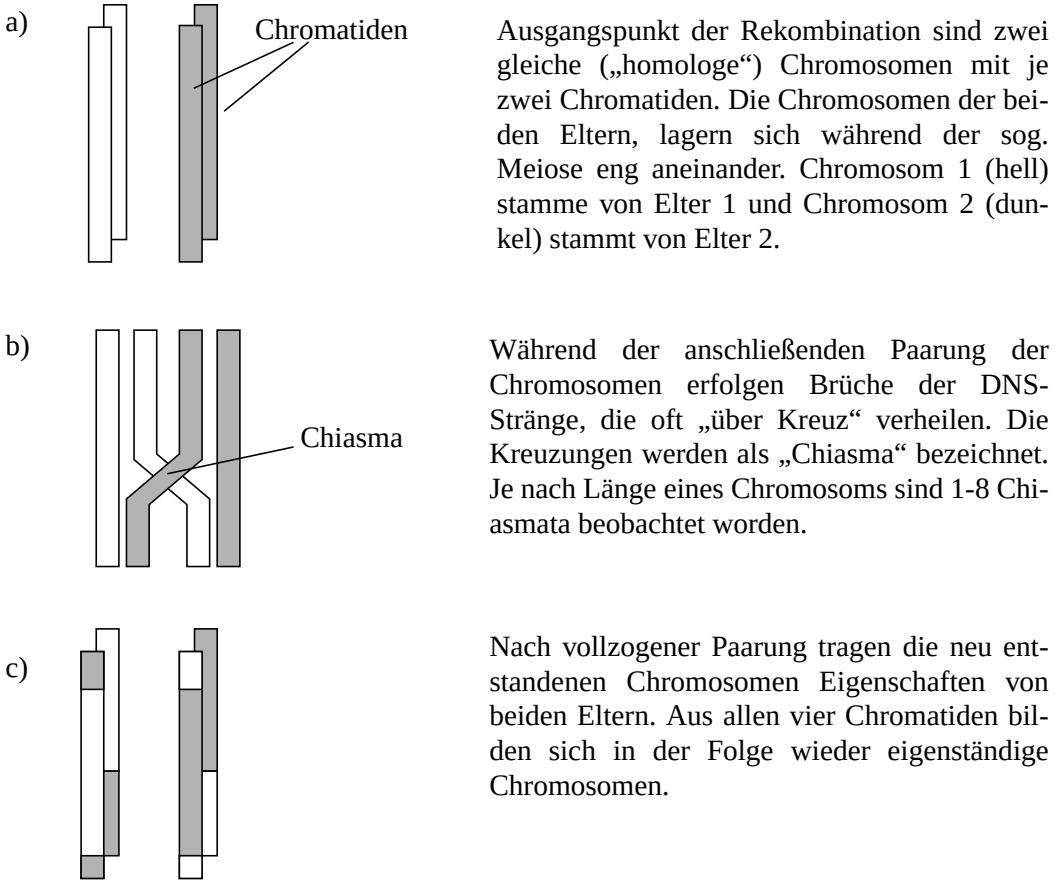


Bild 4-5: Rekombination durch Crossover während der Reifeteilung (Meiose)

ver rekombiniert. Die Steuerung dazu wird wiederum durch bestimmte Gene vorgenommen. Offensichtlich nimmt die Crossover-Wahrscheinlichkeit von zwei Genen mit ihrem Abstand zueinander auf einem Chromosom zu. Da die Rekombination in der Regel homologe Chromosomen betrifft, entstehen nicht völlig neue Eigenschaften durch die Neukombination. Es werden vielmehr bestehende Grundeigenschaften verändert.

Wird die Abfolge der genetischen Information nach der Rekombination zudem durch bestimmte Einflüsse geändert, dann entstehen bei der Umsetzung dieser Information, d.h. bei der Ausbildung des Phänotyps, Individuen mit neuen Eigenschaften. Bleibt diese Veränderung erhalten, so spricht man von *Mutation*, das neue Individuum ist ein Mutant. Auslöser für Mutationen sind z.B. chemische Stoffe, ionisierte oder energiereiche Strahlung. Mutationen können einzelne oder auch mehrere Gene betreffen. Je nach Anzahl der betroffenen Gene unterscheidet man zwischen Punktmutationen, Segmentmutation (hier wird die Chromosomenstruktur verändert) sowie Genom-Mutation, bei der die Anzahl ganzer, vollständiger Chromosome verändert wird.

- Bei der Punkt-Mutation ist nur ein Gen von der Mutation betroffen. Die Häufigkeit

(Mutationsrate), mit der sich ein Gen verändert, beträgt dabei im Mittel 10^{-4} bis 10^{-9} je Generation und Gen. Stellen, die bedingt durch ihre Molekülstruktur häufig von Mutationen betroffen sind, werden als *hot spot* der Mutation bezeichnet.

- Bei der Segment- oder Chromosomen-Mutation sind kleinere oder größere Stücke der DNS betroffen. Hierbei kann es zu Verlust, Umlagerung oder auch Verdopplung dieser Stücke kommen.
- Die Genom-Mutation ändert die Anzahl ganzer Chromosome. Beim Menschen führt dies in der Regel zu sehr großen negativen Anomalien bei den entstehenden Individuen.

In den meisten Fällen sind Mutationen für ihre Träger gleichgültig (der größte Teil der natürlich vorkommenden Mutationen wird durch sog. Reparatur-Gene wieder rückgängig gemacht), sehr oft aber auch nachteilig (Krebserkrankungen!). Im schlimmsten Fall sind sie sogar tödlich; hierbei spricht man von Letalfaktoren. Seltener treten vorteilbringende Mutationen auf, die allerdings über die Generationen hinweg erhalten werden und die weniger vorteilhaften Formen durch die Selektionsvorgänge verdrängen.

4.4.2.1.3 Selektion

Die *Selektion* der Individuen, die sich zu einer neuen Generation fortpflanzen können und damit die eigenen Gene an die Nachkommen weitergeben, bildet den zweiten Kernpunkt des Evolutionsgeschehens. Verschiedene Individuen einer Population bringen aufgrund unterschiedlicher „Eignung“ eine unterschiedliche Anzahl von Nachkommen in die nächste Generation. Höhere Nachkommenzahlen bestimmter Phänotypen führen somit zu einer gerichteten Verschiebung der Genhäufigkeit im Genpool der Population. Die Gründe für die Selektion, die Selektionsfaktoren, können dabei sehr vielgestaltig sein: Einflüsse der belebten und der unbelebten Umwelt. Aufgrund dieser Faktoren vermehren sich, wie bereits angesprochen, die am besten angepaßten Individuen. Allgemein spricht man von einem Selektionsdruck auf die Individuen, wenn sich ihr Phänotyp nicht mit den Umweltbedingungen vereinbaren läßt. Der Selektionskoeffizient $k \in \{0,1\}$, bei dem ein großer Wert für einen hohen Selektionsdruck steht, berechnet sich dabei zu [117]:

$$k = 1 - \frac{\text{Anzahl der Nachkommen eines betrachteten Phänotyps}}{\text{Anzahl der Nachkommen des besten Phänotyps}} \tag{4-9}$$

Abschließend anzumerken ist die Tatsache, daß es sich bei der Selektion um eine passive Erscheinung handelt und nicht um eine aktive, wie man zunächst durch die Beschreibung vermuten könnte.

4.4.2.2 Verwendung des Gray-Codes zur Informationscodierung

Betrachtet man die Wirkung von Mutationsvorgängen, dann ist es sinnvoll, bei einer technischen Umsetzung eine Informationscodierung zu wählen, die dem natürlichen Mutationsverhalten, insbesondere bei Punktmutationen, entspricht. Durch derartige Mutationen ändert sich die damit codierte Information in den meisten Fällen nur sehr gering. Überträgt man diese Eigenschaft auf die binäre Codierung eines Optimierungsproblems mit Hilfe genetischen Algorithmen, so bietet sich hier die Verwendung des *Gray-Codes* [50, 97] an. Er zeichnet sich dadurch aus, daß durch eine Veränderung einer einzelnen Stelle eines Codeworts immer ein Codewort erzeugt wird, das der direkte Nachbar des ursprünglichen Codeworts im gegebenen Coderaum ist. Im Dualcode ist diese Eigenschaft nicht vorhanden. Ändert man z.B. dort beim

Wert 0100*b* = 4*d* das höchstwertige Bit zu 1100*b*, so erhält man den Wert 12*d*, der offensichtlich kein direkter Nachbar zu 4*d* ist. Mit dem Gray-Code hat man die Möglichkeit, die sog. *Hamming Cliffs*, d.h. eben die großen Hamming-Distanzen bei benachbarten Codeworten, zu umgehen.

Die Gray-Codierung kann man dazu verwenden, ganze Zahlen, wie sie z.B. die Prioritäten eines Echtzeit-Prioritätensystems sind, zu codieren. Eine denkbare Lösung ist die Codierung der einzelnen Stellen einer ganzen Dezimalzahl durch 4-bit Codeworte. Dies würde einer BCD-Codierung auf Gray-Code Basis entsprechen. Das Ende einer Zahl (eines Gens) kann durch ein zusätzliches Codewort markiert werden. Damit besteht die Möglichkeit, beliebig große Zahlen zu codieren. Der Nachteil dieser Methode entsteht wieder durch die Kombination der Dezimaldarstellung und der Auswirkung von Mutationen, die je nach Ort unterschiedliche Konsequenzen hat: Eine Punktmutation eines Bits in der Hunderterstelle der Zahl hat größere Auswirkungen als in der Einerstelle. Dieses Problem läßt sich durch eine angepaßte Mutationsrate entschärfen: für jede Stelle/Allel einer Zahl/eines Gens wird ein entsprechender Korrekturfaktor angegeben. Eine weitere Möglichkeit ist die Vorgabe eines festen Zahlenraumes und die vollständige Codierung dieses Raumes mit Hilfe des Gray-Codes. Der Nachteil besteht in der vorher festzulegenden Beschränktheit des Zahlenraumes.

Ein weiteres Problem des Codes besteht darin, daß unter Umständen Codeworte, die zur Bildung des Gray-Codes gebraucht werden, nicht im ursprünglichen Alphabet vorkommen: sollen z.B. die Zahlen 0..9 binär mit einem Gray-Code codiert werden, so sind die Werte von A-F bei einer 4-bit Codierung nicht benutzt und man erhält zwischen den Codeworten für 9 und 0 wieder ein Hamming Cliff. Eine Lösung dieses Problems bestünde darin, daß nur Grundalphabete zugelassen werden, deren Elementzahlen durch Zweierpotenzen darstellbar sind.

4.4.2.3 Grundlagen der evolutionären Algorithmen

Nimmt man die Vorgänge in der Biologie/Natur als Vorlage, so läßt sich ein allgemeiner Ablauf für evolutionäre Algorithmen darstellen (siehe Bild 4-6). Zuerst muß eine Ausgangspopulation $\Pi(t = 0)$ ermittelt werden. Dies kann im Extremfall nur ein Individuum S sein oder eine Menge von m Individuen, die durch Zufall ermittelt worden sind. Als nächster Schritt erfolgt eine Evaluierung der Individuen: die Bestimmung ihrer Qualität $Q(S)$ ist eine Vorbedingung zur Ermittlung, ob die Abbruchbedingung der nachfolgenden Hauptschleife bereits erfüllt ist. Die Abbruchbedingung kann erfüllt sein, wenn die Qualität eines Individuums der Population ausreichend hoch ist, oder der Generationen(=Schleifen)zähler einen bestimmten Wert erreicht hat. Ist das Abbruchkriterium nicht erfüllt, so wird zuerst eine Selektion durchgeführt. Die Mächtigkeit der Menge der künftigen ersten Elterngeneration μ ist in der Regel vorgegeben. Sie wird durch die Zahl der selektierten Individuen festgelegt. Die Elternindividuen werden durch die Rekombination miteinander verknüpft. Dabei entsteht die Zwischengeneration $\Pi'(t)$. Der genaue Ablauf der Rekombination (z.B. die Zahl der erlaubten Crossover oder die Strategie, mit der die jeweiligen Eltern ausgesucht werden) kann über Parameter dieser Funktion festgelegt werden. Bedingt durch die Rekombination, kann die Zahl der Kinder die Zahl der Eltern übersteigen. In der Regel wird hier ebenfalls eine feste Zahl λ für die Kinderindividuen vorgegeben. Durch Aufruf des Mutationsoperators, wiederum mit Berücksichtigung spezifischer Mutationsparameter, wird anschließend die mutierte Kindgeneration $\Pi''(t)$ erzeugt. Die anschließende Evaluierung ist notwendig, um in der Selektionsphase genau die Individuen für die neue Elterngeneration $\Pi(t + 1)$ auszuwählen, die am erfolgversprechendsten sind. In der Selektion wird zusätzlich die Menge $E \in \{\varnothing, \Pi(t)\}$ berücksichtigt, die es erlaubt, auch die Eltern nochmals in den Selektionsprozeß einzubeziehen. Werden die Eltern

```
t = 0; // Generationenzähler
Initialisierung:Π(t=0) = {S1,S2,...Sm}
Evaluierung:Π(t=0): {Q(S1(0)),Q(S2(0)),...Q(Sm(0))}
while( Abbruch() != TRUE )
   Selektion:    Π(t+1)= select(Π(t) u E) // Eltern
   Rekombination:Π'(t) = recombine(Π(t)); // Zwischenstufe
   Mutation:     Π''(t)= mutate(Π'(t));  // Kinder
   Evaluierung:  Π''(t): {Q''(S1(t)),Q''(S2(t)),..Q''(Sl(t))}
   t = t + 1;
```

Bild 4-6: Ablauf der Optimierung bei der Verwendung von evolutionären Algorithmen

komplett durch die Nachkommen ersetzt, so spricht man von einer *generation* oder *general replacement policy* [29], ansonsten handelt es sich um eine *steady-state policy*, bei der bestimmte Individuen über mehrere Generationen hinweg weiterexistieren können.

Dieses beschriebene Vorgehen bildet die Grundlage für alle folgenden Algorithmen. Insbesondere die Genetischen Algorithmen und die Evolutionsstrategien sollen dabei ausführlich dargestellt werden. Das Evolutionäre Programmieren wird nur kurz beschrieben, da die Unterschiede zu den vorangegangenen Verfahren nicht sehr groß sind. Da der biologische Hintergrund immer derselbe ist, ist überhaupt die Frage berechtigt, worin die großen Unterschiede in den Algorithmen bestehen, da die oben vorgestellten Grundoperationen in allen Algorithmen mehr oder weniger ähnlich vorkommen. Dabei liegt die Vermutung nahe, daß der Hauptgrund für die Unterscheidung darin begründet ist, daß die Algorithmen an unterschiedlichen Orten der Welt entwickelt wurden...

4.4.2.3.1 Genetische Algorithmen und das Schematheorem

Die genetischen Algorithmen (GA) sind, wenn man die Verbreitung betrachtet, die Hauptvertreter der Evolutionsalgorithmen [67, 158, 36]. Bei der Verwendung genetischer Algorithmen, werden die Eigenschaften der Individuen in Form von Zeichen- oder Bitketten auf den Chromosomen mit konstanter Länge l kodiert [66]. Dadurch läßt sich direkt eine Gray-Codierung verwenden. Einfache Bitmanipulationen ermöglichen die notwendigen Modifikationen wie Crossover und Mutationen, wobei die Crossover als die „treibende Kraft“ der GA angesehen werden. Eine Mutation wird mit der Wahrscheinlichkeit p_m ein Bit der Zeichenkette verändern. Die Auswahl von Eltern, die eine Rekombination durchführen, wird per Zufall getroffen, ebenso die Position des Crossovers. Die Zahl der Crossover läßt sich über entsprechende Parameter einstellen.

Die Auswahl zukünftiger Eltern einer Generation, die Selektion, kann z.B. über das „Rouletteverfahren“ (proportionale Selektion) geschehen. Dabei wird jedem Individuum ein Sektor des Rouletterades zugeordnet, dessen Größe proportional seiner Qualität ist. Wird es ausgelost, dann kommt es in die neue Population. Die Ermittlung der neuen Population erfolgt durch mehrmaliges Drehen des Rads, bis die gewünschte Zahl an Individuen wieder hergestellt ist. Die Wahrscheinlichkeit für die Selektion des Individuums S_k als zukünftiges Elter ist somit

$n \cdot Q(S_k) / \sum_i^n Q(S_i) = Q(S_k) / \overline{Q}$. Damit ist die Zahl der Nachkommen eines Individuums im Mittel proportional zu seiner Qualität bezogen auf die mittlere Qualität der Population. Der Vorteil dieser Methode besteht darin, daß auch Individuen mit geringer Qualität die (wenn auch geringe) Chance haben, Nachkommen zu haben. Damit ist der Algorithmus in der Lage, ein lokales Optimum wieder zu verlassen, um bessere Optima zu finden. Nachteilig an diesem Verfahren ist, daß nur positive Qualitäten betrachtet werden können. Dies macht den Einsatz von Skalierungsfunktionen nötig, die einen beliebigen Wertebereich der Qualitätsfunktion in den positiven Bereich transformieren.

Der Beweis für die Funktionsfähigkeit der GAs zur Lösung eines spezifischen Optimierungsproblems wird im sog. Schematheorem (siehe Gl. 4-10) gesehen. Die Basis dafür ist die Überlegung, daß bestimmte Muster der Gene $H \in \{0,1,*\}^l$ zum einen in mehreren Individuen einer Population auftreten und daß sie außerdem für bestimmte „gute“ Eigenschaften der Individuen verantwortlich sind. Instanzen dieser Muster sind an allen Stellen identisch, die durch eine „1“ oder eine „0“ gekennzeichnet sind. Stellen die durch „*“ gekennzeichnet sind, sind wahlfrei. Diese Muster werden auch als Schemata oder *hyperplanes* bezeichnet. Mit $m(H^t)$ wird die Zahl der Instanzen des Schemas H^t in der Population $\Pi(t)$ bezeichnet. Weiterhin legt die Ordnung $o(H^t)$ eines Schemas die Zahl der festen Stellen (also der „0“ und „1“) fest und $\delta(H)$ ist die sog. *defining length*, d.h. die maximale Entfernung zwischen den festgelegten Positionen eines Schemas. Folgendes Beispiel verdeutlicht diese Festlegungen: Das Schema $H = (01*1*)$ kann maximal vier verschiedene Instanzen haben. Damit ist $m(H^t) \leq 4$, für die Ordnung gilt $o(H^t) = 3$ und $\delta(H)$ ergibt sich zu 3. Interessant ist es nun zu wissen, wie häufig ein bestimmtes Schema in einer Nachfolgegeneration t+1 vertreten sein wird, da dies im steigenden Fall als die treibende Kraft für die GA angesehen werden kann. Formal läßt sich das Schematheorem wie folgt beschreiben:

$$m(H^{t+1}) \geq m(H^t) \cdot \frac{Q(H^t)}{Q^t} \cdot \left(1 - p_c \frac{\delta(H^t)}{l-1}\right)(1-p_m)^{o(H^t)} \qquad (4\text{-}10)$$

Der erste Quotient bewertet dabei die Qualität des Schemas in bezug auf die mittlere Qualität der gesamten Population, der zweite Ausdruck beschreibt die Wahrscheinlichkeit, daß durch Rekombination das Schema zerstört wird, und der dritte Term berücksichtigt die Auswirkungen der Mutation. Die Aussage des Theorems besteht darin, daß kurze, überdurchschnittliche Schemata niederer Ordnung in nachfolgenden Generationen immer mindestens gleichstark vertreten sind. Solche Schemata werden auch als *building blocks* bezeichnet. Der Grund für die Bevorzugung kurzer Schemata liegt darin, daß die Wahrscheinlichkeit, durch ein Crossover zerstört zu werden, geringer ist als bei langen Schemata. Die Wahrscheinlichkeit, daß durch Mutation ein Schema zerstört wird, hängt direkt von der Ordnung des Schemas ab.

Über diesen Grundmechanismus der GA (auch als *Simple Genetic Algorithm*, SGA, bezeichnet) hinaus gibt es eine Vielzahl von Weiterentwicklungen und Anpassungen. Einige der Nachteile des SGA werden in Kapitel 4.4.3 diskutiert.

4.4.2.3.2 Evolutionsstrategien (ES)

Evolutionsstrategien wurden in den 60er Jahren an der TU Berlin entwickelt [134, 10]. Die Individuencodierung findet hier nicht mit Hilfe von Bitstrings statt, sondern man verwendet die direkten Parameterwerte s, die hier als „Objektvariable“ bezeichnet werden. Damit kann eine aufwendige und u.U. verfälschende Umcodierung entfallen. Diese Parameter werden in ihrer

Beschreibung noch durch die Mutationsweite σ und den Rotationswinkel α ergänzt, so daß ein Individuum S wie folgt dargestellt wird:

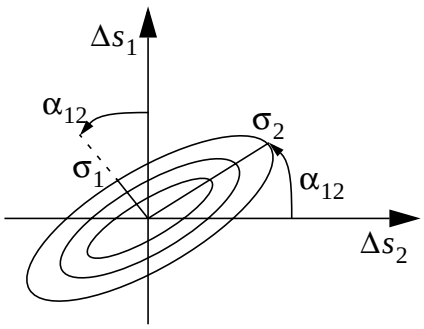
$$S = ((s_1, \ldots s_n), (\sigma_1, \ldots \sigma_{n_\sigma}), (\alpha_1, \ldots \alpha_{n_\alpha})) \tag{4-11}$$

Die Mutationsweite, welche die mittlere Mutationsschrittweite bestimmt, und die Rotationswinkel, die eine Korrelation der Mutationen der Parameter untereinander beschreiben, werden dabei als Strategieparameter bezeichnet. Mit diesen Strategieparametern werden die Mutationseigenschaften der Individuen bestimmt (genauer: die Wahrscheinlichkeitsdichtefunktion der Mutationen), die durch die Winkel eine bestimmte Richtung und durch die Standardabweichung eine bestimmte Amplitude im Werteraum M erfahren. Die Zahl der jeweiligen Strategieparameter ist dabei nicht vorgegeben. Vielmehr lassen sie die in Tabelle 4-1 beschriebenen Möglichkeiten und Kombinationen zu.

n_σ	n_α	Bemerkung
1	0	Normale Mutation, alle Variablen werden gleichstark mutiert, keine Korrelation
n	0	Normale Mutation, jede Variable kann getrennt mutiert werden, keine Korrelation
n	$n \cdot (n-1)/2$	Korrelierte Mutation d.h. die Mutation wirkt sich auf mehrere Parameter aus.
$1 \le n_\sigma \le n$	$(n - n_\sigma/2) \cdot (n_\sigma - 1)$	Allgemeiner Fall der Mutation

Tabelle 4-1: Mögliche Kombinationen der Strategieparameter

Über den Rotationswinkel und die Mutationsweite kann ein Bereich festgelegt werden, in den die Mutation eines Systemparameters mit einer bestimmten Wahrscheinlichkeit fallen wird (siehe Bild 4-7). Die Besonderheit dieses Ansatzes besteht darin, daß diese Strategieparameter über die Generationen hinweg ebenfalls veränderbar sind, und so eine immer bessere Anpassung des Evolutionsvorgangs erzielen. Der biologische Hintergrund für diese gesteuerte Mutation ist zum ersten die Tatsache, daß Nachkommen nicht vollständig unabhängig von ihren



Beispiel eines zweidimensionalen Problems: $n = 2$, $n_\sigma = 2$ und $n_\alpha = 1$. Die Ellipsen geben die Orte mit gleicher Wahrscheinlichkeitsdichte für die Mutation an. Die Mutationen sind hier korreliert, d.h. sie bewegen sich nicht nur auf den Achsen sondern mit hoher Wahrscheinlichkeit in den Quadranten.

Bild 4-7: Mutationsellipsoid bei Evolutionsstrategien

Vorfahren sind, wie es bei einer beliebig gerichteten Mutation der Fall wäre und zum zweiten der in der Einleitung (Abschnitt 4.4.2.1.3) erwähnte Selektionsdruck.

Die Mutation läuft so ab, daß zunächst die Strategieparameter zu σ'_i und α'_j mutiert werden und dann, mit der daraus resultierenden Wahrscheinlichkeitsdichtefunktion, die Systemparameter mutieren. Die Parameter berechnen sich nach [10] wie folgt:

$$\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot U + \tau \cdot U') \tag{4-12}$$

$$\alpha'_j = \alpha_j + 0{,}0873 \cdot U' \tag{4-13}$$

wobei U wieder eine normalverteilte Zufallszahl ist und für i und j die im allgemeinen Fall in Tabelle 4-1 aufgeführten Werte gelten. Außerdem werden die folgenden heuristisch bestimmten Werte verwendet [10]:

$$\tau \sim (\sqrt{2\sqrt{n}})^{-1} \tag{4-14}$$

$$\tau' \sim (\sqrt{2n})^{-1} \tag{4-15}$$

Der erste Summand des Exponenten in Gl. 4-12 ist ein globaler Faktor, der die Gesamtmutabilität beeinflußt. Der zweite Summand erlaubt die individuelle Änderung der Mutationsweite. Dazu wird die Zufallszahl U' für jeden Strategieparameter berechnet, während für den globalen Faktor immer dieselbe Zufallszahl innerhalb einer Generation verwendet wird. Schließlich folgt die Mutation der Systemparameter:

$$S' = S + \text{random}(0, \vec{\sigma}', \vec{\alpha}') \tag{4-16}$$

Dabei drückt die random-Funktion einen Zufallsvektor aus, der, ausgehend vom Elter S, normalverteilt im n-dimensionalen Raum mit dem Erwartungswert 0, den Standardabweichungen $\vec{\sigma}$ und den Rotationswinkeln $\vec{\alpha}$ den Nachkommen des Elters kennzeichnet.

Die Rekombination, als weiterer genetischer Operator, berücksichtigt außer den Systemparametern auch die Strategieparameter. Von zwei zufällig gewählten Eltern werden zunächst die Strategieparameter durch Crossover neu auf die Nachkommen verteilt. Angepaßtere Operatoren behandeln dabei die einzelnen Bereiche (Parameter) eines Individuums unterschiedlich. Andere Formen verwenden für jeden Teil eines neuen Individuums den komplementären Teil eines (zufälligen) anderen Elters.

Bilden μ Individuen die Elterngeneration, so entstehen bei den ES durch deterministische Selektion λ Individuen einer Kindergeneration. Wie bereits in Kapitel 4.4.2.3 dargelegt, können unterschiedliche Methoden bei der Selektion zum Tragen kommen. Die generation policy wird bei den ES als (μ, λ)-Selektion bezeichnet, eine steady-state policy durch die $(\mu + \lambda)$-Selektion beschrieben. Interessanterweise[5] ist die erstgenannte „Komma“-Strategie erfolgreicher: nur dadurch, daß bestimmte Strategieparameter von bereits sehr gut angepaßten Individuen auch wieder vergessen werden können, kann eine Stagnation der Evolution durch einen immer stärker konvergierenden Genpool vermieden werden. Bei einer (15, 70)-Selektion entstehen aus 15 Eltern insgesamt 70 Nachkommen, aus denen wiederum die 15 Eltern der nächsten Generation ausgewählt werden. Empirische Untersuchungen [134] haben gezeigt, daß ein Verhältnis $\lambda/\mu \cong 7$ relativ gute Konvergenzeigenschaften besitzt um ein globales Maximum zu finden. Ist das Verhältnis größer (d.h. man verwendet ein scharfes Selektionskriterium), so lassen sich schneller lokale Optima finden.

5. In der Natur ist dies der normale Vorgang, um Inzuchtprobleme zu vermeiden.

Zusammenfassend läßt sich sagen, daß sich das Lernen bei Evolutionsstrategien - im Gegensatz zu den GA - auf zwei Ebenen abspielt: zum einen auf der auch bei den GA vorhandenen Phänotyp-Ebene, d.h. der Anpassung und Optimierung der Qualität der Individuen und zum anderen auf der Genotyp-Ebene, d.h. bei der Optimierung der Strategieparameter, welche die Evolutionsrichtung bestimmen.

4.4.2.3.3 Evolutionäre Programmierung (EP)

Das Evolutionäre Programmieren [37] weist sehr große Übereinstimmungen mit den ES auf. Hier werden Individuen ebenfalls nicht codiert, sondern die u.U. reellwertigen Systemparameter werden direkt zur Berechnung der Systemqualität verwendet. Die Mutationen, die verwendet werden, um neue Populationen zu erzeugen, weisen zunächst eine große Streuung bezüglich ihrer Stärke auf. Allerdings wird die Stärke der Mutationen verringert, je mehr man sich dem Optimum nähert. Dieses Verhalten führt allerdings in eine Tautologie: da man das Optimum nicht kennt, kann auch nicht gesagt werden, wann man sich in dessen Nähe befindet. Zur Lösung dieses Problems werden sog. „Meta-Evolutionstechniken“ angewendet, bei denen z.B. die Varianz der erlaubten Mutationen über den Optimierungsverlauf hinweg modifiziert, d.h. vermindert wird. Dies entspricht im wesentlichen dem Cooling-Schedule beim simulierten Ausglühen. Crossover bzw. die Rekombination werden bei EP nicht verwendet, da die Abstraktion der Evolution hier auf der Ebene der Populationen stattfindet (bei den ES und GA findet die Abstraktion auf der Ebene der Individuen statt). EP führt i.d.R. eine stochastische Selektion durch, d.h. nach der zufälligen Auswahl zweier oder mehr Nachkommen wird durch Vergleich entschieden (*tournament selection*), welches als Elter in die nächste Generation übernommen wird. Ein Individuum gewinnt, wenn es gleichgut oder besser ist, als seine Kontrahenten. Schließlich werden die Individuen ausgewählt, welche die meisten Siege verzeichnet haben.

Tabelle 4-2 zeigt nochmals die Unterschiede der drei vorgestellten Evolutionären Algorithmen.

Genetische Algorithmen (SGA)	Evolutionsstrategien (ES)	Evolutionäres Programmieren (EP)
Codierung (i.d.R.) durch Binär-Strings	Codierung mit realen Werten	Codierung mit realen Werten
Probabilistische Selektion	Deterministische Selektion	Probabilistische Selektion
Keine Selbstanpassung	Selbstanpassung der Selektionsfaktoren	Selbstanpassung der Selektionsfaktoren
Rekombination als Hauptoperator	Mutation als Hauptoperator	Mutation als Hauptoperator, keine Rekombination
Bit-Mutationen	Normalverteilte Mutationen	Normalverteilte Mutationen

Tabelle 4-2: Vergleich von (S)GA, ES und EP

4.4.3 Bewertung der Algorithmen

Die Bewertung der Einsatzmöglichkeit der vorgestellten Algorithmen ist nicht trivial, da es vom jeweiligen Problem abhängt, welcher Algorithmus sich am besten eignet. Interessant ist

dabei die Tatsache, daß sich ES und EP durch geeignete Wahl der Parameter so modifizieren lassen, daß sie dem simulierten Ausglühen (SA) sehr ähnlich sind.

Durch die Komplexität der Parameterwahl hat man in allen naturanalogen Algorithmen das Problem nachzuweisen, daß der Algorithmus nur deshalb nicht gut funktioniert, weil die Parameter schlecht gewählt sind. Ein schlechtes Abschneiden des Algorithmus kann somit immer auf eine schlechte Wahl der Parameter zurückgeführt werden [77].

Kritikpunkt und Hauptvorteil zugleich ist bei ES, EP und SA, daß keine spezielle Codierung für die Individuen stattfinden muß. Dieses Vorgehen weicht vom natürlichen Vorbild ab, hat aber den Vorteil, außer der höheren Effizienz durch das fehlende Codieren/Decodieren, daß Verfälschungen der Systeme durch Abbildungsfehler vermieden werden können.

Einer der Hauptkritikpunkte der GA ist die starke Betonung der Rekombination als bestimmenden Operator. Der dadurch entstehende Nachteil ist, daß bei vielen ähnlichen Individuen in einer Population die Rekombination keine neuen Eigenschaften mehr in die Population bringen kann. Es werden vielmehr immer dieselben Eigenschaften zwischen den Individuen ausgetauscht. Diese Tatsache stellt auch die Building-Block-Hypothese in Frage, die annimmt, daß durch die Kombination der Building-Blocks Individuen mit guten Eigenschaften und letztendlich hoher Qualität entstehen. Dies hängt jedoch vielmehr von der Qualitätsfunktion ab, so daß durch die Aneinanderreihung von Building Blocks durchaus auch Individuen mit sehr schlechten Eigenschaften erzeugt werden können [140].

Ein Vorteil aller vorgestellten evolutionären Algorithmen besteht in ihrem impliziten Parallelismus. Da der Hauptaufwand eines solchen Algorithmus normalerweise in der Evaluierung, also der Berechnung der Qualitätsfunktion für ein System/Individuum zu suchen ist, kann eine Beschleunigung dieser Aufgabe durch eine Aufteilung auf mehrere Maschinen realisiert werden. Dabei sind keine größeren Synchronisationsprobleme zu erwarten.

Die naturanalogen Verfahren (namentlich Simulated Annealing) sind schon mit Erfolg in verschiedenen Job Shop/Flow Shop Problemen angewendet worden ([151, 53, 30], siehe auch Kapitel 6), so daß ein Einsatz dieser Methoden für das vorliegende Problem naheliegt. Dies um so mehr, da es sich um ein *NP*-vollständiges, kombinatorisches Optimierungsproblem handelt, welches mit den klassischen Optimierungsmethoden, wie bereits gesehen, nicht behandelbar ist.

In allen Verfahren bleibt jedoch das Problem der Lage eines globalen Optimums und die Frage nach der effizienten Parametrierung der Algorithmen offen. Für diese liegen, außer einigen Heuristiken für ausgewählte Probleme, noch immer keine genauen Anhaltspunkte vor. Aus diesem Grund scheint der Ansatz der Evolutionsstrategien vielversprechend zu sein, da hier auch eine evolutionäre Anpassung des Algorithmus stattfindet. Allerdings wird bei den ES - im Gegensatz zu den GA - zuwenig auf die Möglichkeiten der Rekombination eingegangen. Deshalb scheint, auch unter der Berücksichtigung des biologischen Vorbilds, eine Kombination aus ES und GA am vielversprechendsten.

Nachsatz: Der Versuch einer Kopie der Vorgehensweise der Natur zur Lösungsoptimierung ist noch nicht sehr alt. Die Versuchung, im Umkehrschluß, damit das „Vorgehen der Natur“ vollständig erklärt zu haben, ist sehr groß; zeigen doch die verwendeten Algorithmen sehr gute und robuste Eigenschaften. Allerdings sind diese Erkenntnisse nicht ausreichend zur Erklärung der Evolution und unseres Daseins. Zwei Gründe sprechen gegen diese einfache Annahme. Der erste Grund basiert auf dem Irrtum, durch die Evolution würden nur „optimale“ oder

„gute“ Individuen geschaffen. Tatsächlich aber entstehen nur solche Individuen, die immer besser an die Umwelt angepaßt sind.

Der zweite Irrtum in dieser Beziehung besteht in der Annahme, daß die organische Evolution lediglich ein gigantisches, gerichtetes Würfelspiel sei, obwohl dieses Evolutionsmodell so einfach nicht funktioniert haben kann: Eine menschliche Keimzelle enthält ca. 70000 Gene [121], die wiederum im Mittel aus je 300 Basentriplets zusammengesetzt sind. Nimmt man die damit codierten 20 Aminosäuren als Grundlage für den Bauplan der Individuen, so ließen sich theoretisch $20^{21000000}$ unterschiedliche Genotypen bilden. Die Existenz unseres Planeten, und somit die Zeit, um diese Genotypen „auszutesten“, beträgt aber bisher gerade einmal ca. 4 Mrd. Jahre. Dies entspricht einem Wert von ca. 10^{17} Sekunden!

Kapitel 5

Modellierung und Analyse von Fahrzeugnetzen auf der Basis von Echtzeitprioritätensystemen

Das Steuergerätenetz eines Fahrzeugs muß in seiner Gesamtheit unterschiedliche Dienste erbringen. Viele dieser Dienste werden durch mehrere, räumlich getrennte Komponenten erbracht, so daß Kommunikation zwischen diesen Komponenten eine funktionale Basis für das Erbringen des Dienstes ist. Die Plazierung von Steuergeräten (SG) im Fahrzeug und damit auch im Netz ist durch ihre Funktion offensichtlich weitgehend vorgegeben und deshalb nicht Gegenstand weiterer Betrachtung.

Die Dienste („job“, „service“ oder auch „transaction“ [25, 69]) eines Steuergerätenetzes *S* werden durch die Zusammenarbeit mehrerer Tasks erbracht, die auf den Steuergeräten dieses verteilten Systems ablaufen. Außerdem werden Nachrichten über die Kommunikationsmedien verschickt, um diese Zusammenarbeit zu ermöglichen. Dabei muß aus einer Spezifikation der Dienste, die das zeitliche Auftreten der Dienste und die zeitlichen Grenzen für deren Erbringung festlegt, sowie der vorhandenen Software-Komponenten, wie etwa Anwendungsprogramme und Betriebssystemroutinen, eine Abbildung auf die vorhandene Hardware-Konfiguration erfolgen. Aus dieser Spezifikation wird ein Modell erstellt, das der anschließenden Analyse zugänglich ist. In dieser Arbeit wird zur Modellierung und Darstellung des Systems eine Dienst-Prozessor-Aktivitäten-Matrix (DPA-Matrix) eingeführt.

Basierend auf dem erstellten Modell des Zielsystems sollen die in den vorangegangenen Kapiteln vorgestellten Analysemethoden erweitert und angepaßt werden. Dies bedeutet im wesentlichen eine Erweiterung der Antwortzeitanalyse für Einprozessorsysteme auf verteilte Echtzeitprioritätensysteme, die mit heterogenen Scheduling-Verfahren arbeiten.

Zur Validierung der Analyseergebnisse wird eine zeitdiskrete ereignisgesteuerte Simulation eines Steuergerätenetzes verwendet. Da die Simulationstechnik nicht Gegenstand dieser Arbeit ist, soll sie nicht weiter vertieft werden. Die grundlegende Architektur des Simulationsmodells ist im Anhang A3 kurz dokumentiert. Eine gute Übersicht zur verwendeten Simulationsumgebung gibt [90]. Detailliertere Information zum Simulationsmodell selbst kann [40], [34], [125] und [11] entnommen werden.

5.1 Modellierung des Steuergerätenetzes

Die Modellierung eines Steuergerätenetzes geschieht in zwei Schritten. Zunächst werden aus einer Dienstespezifikation und der (geplanten) Hardware-Konfiguration die Software-Anforderungen der Dienste ohne Berücksichtigung des Kommunikationsnetzes bestimmt. Im zweiten Schritt wird daraus, unter Berücksichtigung der Verbindungsstruktur, die DPA-Matrix für das gegebene Komplettsystem abgeleitet.

Basierend auf den in Kapitel 3 vorgestellten Betrachtungen zu Echtzeitprioritätensystemen wird, ohne Einschränkung der Gültigkeit des Beschriebenen, folgende Regel formuliert:

Regel 1: Prozessoren mit prioritätsgesteuerten Schedulern und ihnen entsprechende Kommunikationsmedien wie Busse mit bitweiser Arbitrierung können aus der Sicht der Analyse dieser Systeme exakt gleich behandelt werden. Nachrichten, die über das Kommunikationsmedium ausgetauscht werden, haben die gleichen zeitlichen Eigenschaften wie Tasks, die auf einem Prozessor ablaufen.

Es wird deshalb in dieser Beziehung im weiteren Verlauf der Betrachtung kein Unterschied zwischen Bus und Prozessor gemacht, sondern lediglich von *Prozessoren k* (vgl. auch „Bedieneinheiten“) und allgemein von *Aktivitäten A* gesprochen, die auf diesen Prozessoren ablaufen.

5.1.1 Abbildung der Diensteanforderungen

Für die erste Abbildung ist eine Dienstespezifikation des Systems notwendig. Sie umfaßt die Festlegung der Dienstemenge X des Gesamtsystems S. Diese beinhaltet alle m Dienste des Systems. Ein Dienst $s \in X$ wird im ersten Modellierungsschritt durch das Parameter-Tupel $s = (D_s, T_s)$ gekennzeichnet, wobei

- D_s seine Frist angibt, bis zu der, gemessen ab der Anforderung des Dienstes, dieser erbracht sein muß und
- T_s die minimale Zeit (vgl. Kapitel 3: T_{min}) zwischen zwei aufeinanderfolgenden Anforderungen des Dienstes ist.

Die Zeiten für diese beiden Dienstparameter müssen, wie in Abschnitt 3.3 bereits beschrieben, aus den physikalischen Anforderungen an die Dienste abgeleitet werden.

Schwieriger ist die Festlegung der minimalen Zeit zwischen zwei Anforderungen, insbesondere, wenn aperiodische und sporadische Vorgänge beschrieben werden sollen. Hierbei müssen ebenfalls die physikalischen Randbedingungen berücksichtigt und außerdem die motorischen Eigenschaften der Fahrzeugbenutzer einbezogen werden: Das Drücken eines Knopfes ist ein durchaus aperiodischer sporadischer Vorgang, der allerdings nur mit einer bestimmten maximalen Frequenz ausgeführt werden kann. Der Kehrwert der maximalen Frequenz wird zur Festlegung der Anforderungsabstände des Dienstes verwendet.

Um aus dieser Information mit Hilfe einer Abbildung die ersten Software-Anforderungen zu generieren, wird zunächst die Annahme getroffen, daß jeder Dienst auf getrennte Aktivitäten abgebildet wird. Eine Ausnahme bilden die Dienste, bei denen Information von einer Komponente an mehrere Komponenten versendet werden muß (bei 1:n-Kommunikation).

Aus diesen Festlegungen läßt sich die bisherige Dienstbeschreibung wie folgt erweitern: Hinzu kommt eine Menge G_s von Aktivitäten A (mit $A \in G_s$) des Dienstes s, die in diesem Schritt

zunächst von der Anfangsaktivität und der Endaktivität des Dienstes gebildet wird. Durch diese explizite Vergabe von Ablaufpositionen der Aktivitäten läßt sich damit ein gerichteter Aktivitätengraph erstellen (auch „task chain“ [47]), der die Abfolge der Aktivitäten innerhalb eines Dienstes wiedergibt: $A_{Anfang} > A_{Ende}$. Dabei bedeutet „x > y“, daß „y“ erst ausgeführt werden kann, wenn „x“ beendet ist. Somit ist „x“ der Vorgänger von „y“. Bild 5-1 zeigt nochmals die beschriebenen Zusammenhänge, wobei für den Aktivitätengraph eine alternative graphische Darstellung mit Knoten und gerichteten Kanten verwendet wird.

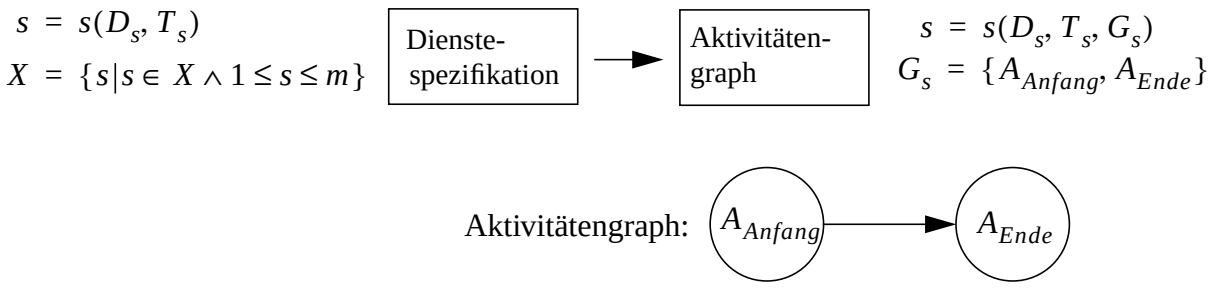


Bild 5-1: Erste Abbildung der Anforderungen und Aktivitätengraph

Parallel zur Spezifikation der Dienstanforderungen und der ersten Erstellung der Aktivitätengraphen muß eine Spezifikation der Hardwarekomponenten erfolgen. Diese umfaßt die nachfolgenden Punkte:

- Die verwendeten Komponenten *k* im System, d.h. die Steuergeräte mit ihren Mikrocontrollern und, mit Anwendung von Regel 1, die verwendeten Bussysteme. Die Menge der Komponenten K mit $k \in K$ und $1 \leq k \leq n$ bildet die physikalische Grundlage des Systems.[1]
- Die Leistungsfähigkeit l_k der verwendeten Komponenten *k*. Bei den Mikrocontrollern in den Steuergeräten handelt es sich um die Bearbeitungsgeschwindigkeit, bei den Kommunikationssystemen um die Bitrate, mit der sie Nachrichten übertragen können. Die Angabe der Leistungsfähigkeit erfolgt in einer normierten Einheit *tick/s*, wobei ein tick die kleinste Zeiteinheit darstellt, die auf der jeweiligen Komponente existiert. Dies erlaubt eine einheitliche Darstellung des Bearbeitungsaufwands einer Aktivität unabhängig von der verwendeten Komponente, d.h. dieser Wert gilt für alle potentiellen Prozessoren, auf denen die Aktivität bearbeitet werden kann.
- Der Typ des auf dem Prozessor verwendeten Schedulers: unterbrechendes (*k*: up) oder nichtunterbrechendes (*k*: nup) Prioritätensystem. Da bei den betrachteten Systemen in der Regel beide Schedulingarten vertreten sind, muß dies für den jeweiligen Prozessor explizit angegeben werden können.

5.1.2 Abbildung auf das Zielsystem

Im zweiten Schritt erfolgt eine Erweiterung des Aktivitätengraphen durch die Einbeziehung der Komponentenverbindungen des Zielsystems und der damit einhergehenden Einflußnahme zusätzlicher Software-Teile (z.B. Gateway-Funktionen oder Kommunikationsdienste eines

1. Die örtliche Verteilung und zahlenmäßige Dimensionierung der Steuergeräte ist wiederum eine eigene Optimierungsaufgabe.

Prozessors). Ergebnis daraus ist die DPA-Matrix, welche alle gefundenen Aktivitätengraphen ergänzt, über der Hardware des Systems faltet und dadurch die Zusammenhänge zwischen Diensten, Prozessoren und Aktivitäten des Systems anschaulich macht.

Für diese Modellierung muß eine Verbindungsmatrix *V* angegeben werden, die beschreibt, wie die Komponenten *k* des verteilten Systems untereinander verknüpft sind. Dies geschieht durch Auswahl der Verbindungstopologie und durch die Zuordnung der einzelnen Steuergeräte zu den entsprechenden Teilnetzen. Bild 5-2 zeigt diesen Modellierungsschritt.

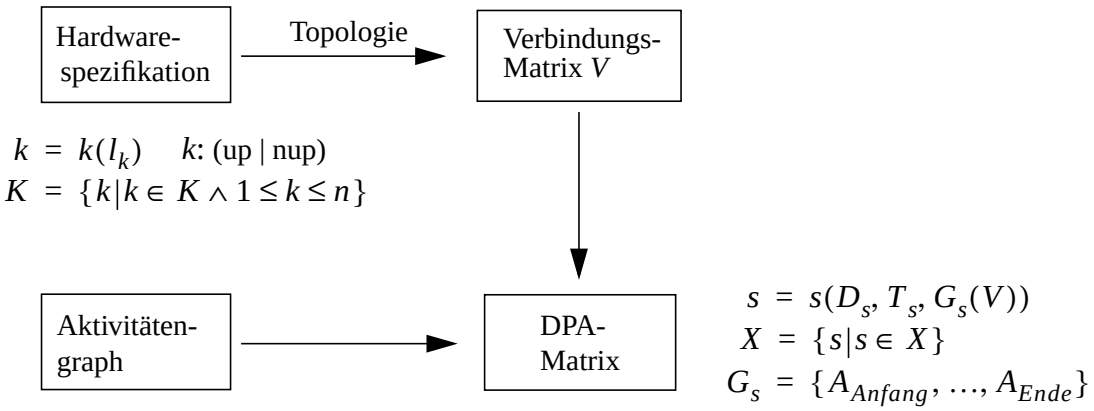


Bild 5-2: Zweite Abbildung der Anforderungen

- In einer ersten Phase wird die formale Beschreibung der Topologie durch eine Verbindungsmatrix *V* realisiert. Durch diese Verbindungsmatrix wird angegeben, ob zwei Komponenten des Systems miteinander verbunden sind oder ob sie keine Verbindung haben. Verbindungen innerhalb einer Komponente werden hier nicht explizit angegeben.
- Anhand der Verbindungsmatrix läßt sich in der zweiten Phase die geordnete Menge $G_s(V)$ aller Tasks ermitteln, die am Dienst *s* beteiligt sind. Dazu muß lediglich mittels eines geeigneten Algorithmus‘ (z.B. Shortest-Path Algorithmus [135]) ein kurzer Weg durch die Verbindungsmatrix von Start- zu Endsteuergerät (und damit auch Task) des Dienstes gesucht werden. Beim Einsatz von Buskomponenten wird an dieser Stelle die Übertragung einer Botschaft eingeplant.

Die Wegesuche durch das System ist im Regelfall trivial, da aus Kostengründen keine Alternativrouten zur Verfügung stehen werden und somit meist genau ein Weg existiert. Müssen jedoch Alternativen berücksichtigt werden, so ist ein gegenseitiges Abwägen der Gesamtkosten (Leitungslänge, Prozessorleistung, etc.) dieser Wege unerläßlich. Bild 5-3 zeigt ein einfaches Beispiel für diesen Teil des Modellierungsvorgangs.

Durch die Modellierung steht auch die Zahl der Botschaften fest, die über die Busse ausgetauscht werden. Über die dadurch erhaltene Kommunikationsleistung jedes Steuergeräts läßt sich ein Kommunikationssubsystem (wie beispielsweise in Abschnitt 2.1.4.2 beschrieben) dimensionieren. Liegen dessen Eigenschaften z.B. in Form von Kommunikationsmodulen in einer Software-Komponentenbibliothek vor, dann können seine Aktivitäten auf dem Prozessor ebenfalls als Tasks modelliert und in den Schedule eingeplant werden.

Unter Anwendung von Regel 1 kann aus den bisherigen Festlegungen eine DPA-Matrix (siehe Bild 5-4) aufgestellt werden, welche die Verteilung aller an einem Dienst beteiligten Aktivitä-

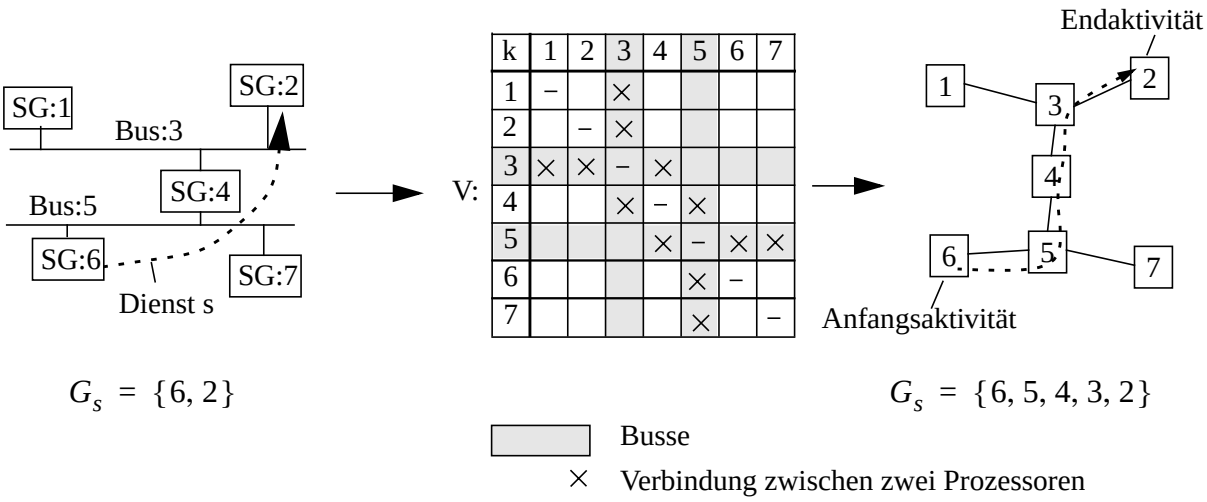


Bild 5-3: Beispiel für die Modellierung eines Steuergerätenetzes

ten auf der zugrundeliegenden Hardware und die Ablaufreihenfolge (siehe auch folgenden Abschnitt) dieser Aktivitäten innerhalb eines Dienstes beschreibt. Die Matrix bildet somit eine Darstellung aller gerichteten Aktivitätengraphen auf dem Hardware-System. Zum Verständnis sind im Bild 5-4 die Sensoren und Aktoren vermerkt, die von den jeweiligen Aktivitäten kontrolliert werden. Aktivitäten ohne zusätzliches Attribut (weder „Sensor“ noch „Aktor“) sind Nachrichten auf dem Bus. Jedes Element der Matrix, also jede Aktivität $A_{s,k}$, wird durch folgende Attribute beschrieben:

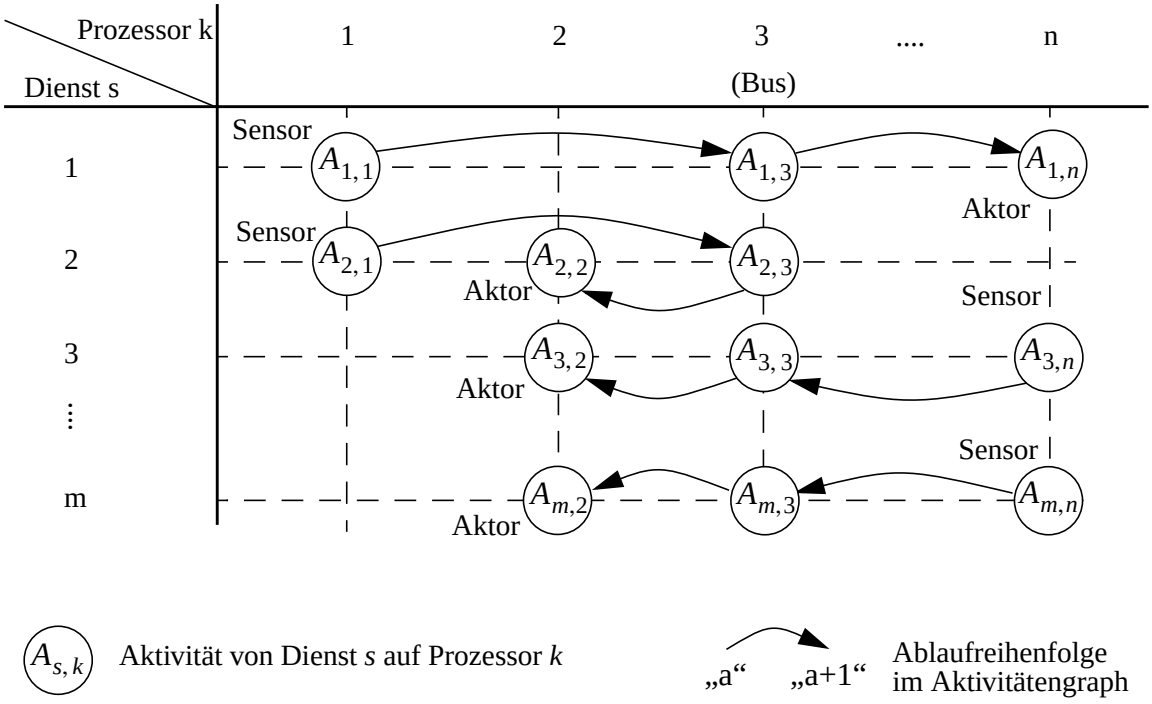


Bild 5-4: Die Dienste-Prozessor-Aktivitäten (DPA) Matrix

- $p = p(A_{s,k})$, die statische Priorität[2], mit der die Aktivität auf dem jeweiligen Prozessor eingeplant wird,
- $C^* = C^*(A_{s,k})$, die bereits angesprochene und auf *ticks* normierte Bearbeitungszeit[3] dieser Aktivität,
- $a = a(A_{s,k})$, die Ablaufposition der Aktivität innerhalb des Dienstes, wobei für die erste Aktivität des Dienstes $a = 1$ gilt und a_{max} die letzte Aktivität des Dienstes markiert, und
- s und k geben die Position in der Matrix an (und damit den Dienst, dessen Teil die Aktivität ist sowie den Prozessor, auf dem die Aktivität abläuft).
- Aus der Matrix läßt sich schließlich die Zahl der Aktivitäten e_k ablesen, die von einem Prozessor k bearbeitet werden müssen.

Elemente der Matrix, die zu keinem Dienst gehören, haben die Eigenschaften $p = \infty, C^* = 0, a = 0$, d.h. keine Priorität, keine Ausführungszeit und keine Position innerhalb eines Dienstes. Zur Vereinfachung der Darstellung werden nicht immer alle Indizes aufgeführt, sondern nur die für die Erläuterung notwendigen.

Aus der normierten Bearbeitungszeit C^* und der Prozessorgeschwindigkeit l_k läßt sich leicht die reale Bearbeitungszeit berechnen: $C = C^*/l_k$. D.h. eine Aktivität mit einer Bearbeitungszeit $C^* = 300$ *ticks* benötigt auf einem Prozessor, der z.B. mit 1000 ticks/ms arbeitet, insgesamt 0,3 ms.

5.1.3 Abfolgerelation und Attributpropagierung

Die Abfolge der Aktivitäten in einem Dienst s erfolgt immer streng deterministisch, d.h. eine Aktivität $A_s|_a$ an der Abfolgeposition a muß abgeschlossen sein, bevor die ihr nachfolgende Aktivität des Dienstes $A_s|_{a+1}$ ausgeführt werden kann (sog. „flow-shop“ [25]). Dies soll am Beispiel in Bild 5-5 veranschaulicht werden.

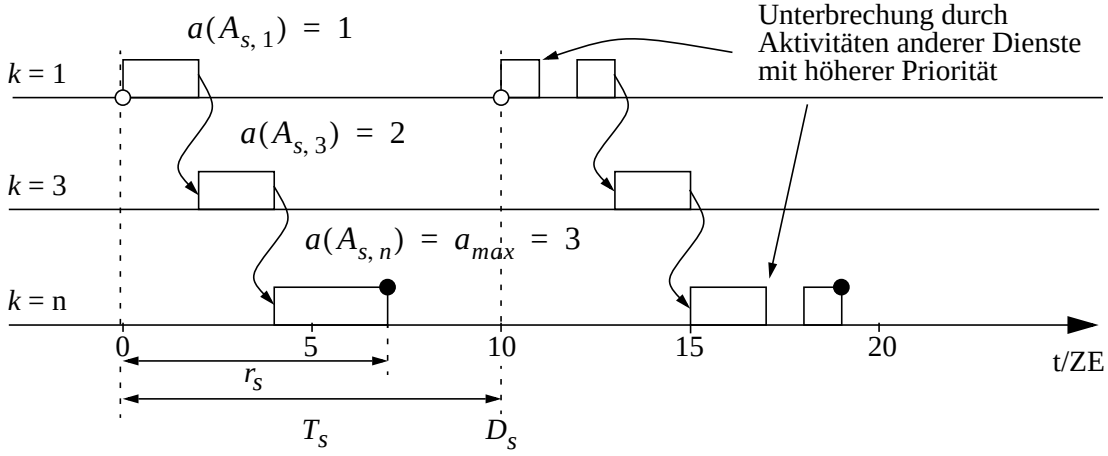


Bild 5-5: Ablauf eines Dienstes *s*, der von drei Aktivitäten auf drei Prozessoren erbracht wird

2. Die statische Priorität p ist eng mit dem in Kapitel 3 eingeführten Index i verknüpft: Während p eine beliebige positive ganze Zahl darstellt, die der Spezifikation entnommen wurde und somit ein Resultat der Planung ist, gibt i die nach Prioritäten sortierte Position einer Aktivität auf dem jeweiligen Prozessor an. Da im verteilten System zwischen den Prozessoren unterschieden werden muß, reicht die in Kapitel 3 verwendete Darstellung nicht mehr aus.
3. Zur Ermittlung der normierten Bearbeitungszeit: siehe Abschnitt 3.6.

Der dargestellte Dienst startet mit der Aktivität $A_{s,1}$ auf Prozessor $k = 1$. Hier wird z.B. ein Sensorsignal ermittelt, das, nach kurzer Vorverarbeitung, über einen weiteren Prozessor ($k = 2$, z.B. dem Bussystem) an Prozessor $k = 3$ übertragen wird. Die Abfolge der Aktivitäten des Dienstes lautet $A_{s,1} > A_{s,2} > A_{s,3}$. Den auf den Prozessoren arbeitenden Schedulern können durchaus verschiedene Prioritätensysteme zugrunde liegen. Während die Aktivitäten auf den Steuergeräten 1 und 2 z.B. unter einem unterbrechenden Scheduler arbeiten, kann der Bus ein nichtunterbrechendes System darstellen.

Um die Worst-case-Analysemethoden, wie in Kapitel 3 vorgestellt, anwenden zu können, ist es in einem weiteren Transformationsschritt notwendig, bestimmte Eigenschaften der Dienste an die einzelnen Aktivitäten dieser Dienste weiterzugeben. Ein grundlegendes Attribut ist die Periodizität, die von den Diensten an ihre Aktivitäten propagiert wird. Deshalb gilt für den Ankunftsabstand T der ersten Aktivität $a(A_s) = 1$ des Dienstes s:

$$T(A_{s,k}|_{a=1}) = T_s \tag{5-1}$$

Alle Aktivitäten $A_{s,k}|_a$ mit $1 < a \le a_{max}$ eines Dienstes erfahren außerdem eine Verzögerung durch vorhergehende Aktivitäten des Dienstes. Damit ergibt sich der in Bild 5-6 dargestellte Sachverhalt. Die Aktivitäten werden jetzt mit eigenen Startzeitpunkten in das System eingeplant.

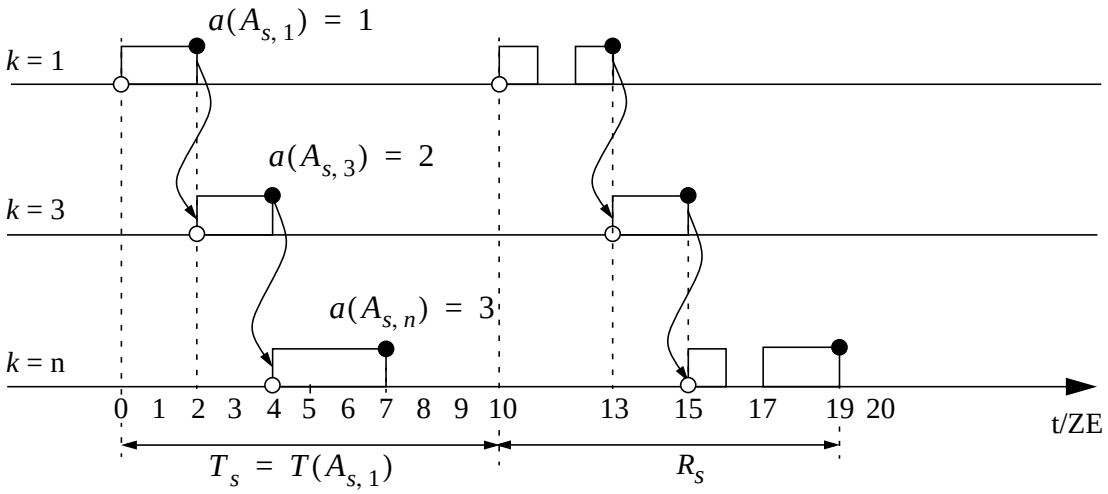


Bild 5-6: Propagierung der Diensteigenschaften an die beteiligten Aktivitäten

Durch diese Form der Modellierung wird der Zeitraum begrenzt, innerhalb dessen eine Aktivität $T(A_s|_a)$ auf ihrem Prozessor eingeplant werden kann:

$$T(A_s|_{a-1}) + C(A_s|_{a-1}) - r(A_s|_{a-1}) \le T(A_s|_a) \le T(A_s|_{a-1}) + r(A_s|_{a-1}) - C(A_s|_{a-1}) \tag{5-2}$$

Die untere und die obere Grenze des Einplanungszeitpunktabstandes einer Aktivität wird durch die Periode $T(A_s|_{a-1})$, die Bearbeitungszeit $C(A_s|_{a-1})$ und die Worst-case-Antwortzeit $r(A_s|_{a-1})$ der vorausgehenden Aktivitäten vorgegeben (Beweis: siehe Anhang A2). Insbesondere die Festlegung der unteren Grenze, d.h. des garantierten maximalen Mindestabstands zwischen zwei Anforderungen dient im folgenden als eine Randbedingung bei der Ermittlung von Worst-case-Szenarien.

5.2 Bestimmung der Worst-case-Antwortzeit in verteilten Systemen

Die Scheduling-Literatur beschäftigte sich in den letzten Jahren hauptsächlich mit Einprozessorsystemen [7], [57], [81], [161]. (Wesentliche Ergebnisse daraus wurden bereits in Kapitel 3 vorgestellt). Dabei werden hauptsächlich periodische Dienste betrachtet, die genau eine Aktivität besitzen und somit das Hauptproblem in der Bestimmung der Interferenz der Dienste auf dem Prozessor liegt. Ist die Zahl der Dienste $m = 1$, dann wird das Problem trivial. Systeme aus mehreren Prozessoren, bei denen ebenfalls nur eine Aktivität pro Dienst benötigt wird, lassen sich genau gleich behandeln wie Einprozessorsysteme, da sie keine hier zu berücksichtigende Kopplung aufweisen.

Bestehen Dienste aus mehreren Aktivitäten, die in einer bestimmten Reihenfolge auf bestimmten Prozessoren bearbeitet werden müssen (flow-shop, im Gegensatz zu einem job-shop, bei dem die Zuordnung der Aktivitäten auf die Prozessoren nicht fest ist [25]), so wird eine Ende-zu-Ende-Planung wichtig. Gründe für die Aufteilung in mehrere Aktivitäten sind zum einen die Modularisierung großer Software-Systeme in Einzelaktivitäten, um die Komplexität der Software besser beherrschbar zu machen und zum anderen der hier betrachtete Fall, daß bestimmte Aktivitäten räumlich an dedizierte Prozessoren gebunden sind. Die Schwierigkeit bei der Analyse im komplexesten Fall solcher Systeme (beliebige Anzahl von Dienste auf beliebiger Zahl von Prozessoren) liegt, außer in der Berechnung der Interferenz, in der Berücksichtigung der Abfolge der Aktivitäten. Bild 5-7 zeigt eine kurze Einordnung der Komplexität bei der Berechnung möglicher dienstebearbeitender Systeme, wobei die Komplexität von oben nach unten und von links nach rechts zunimmt.

		1 Prozessor	n Prozessoren
m Dienste	1 Aktivität	Interferenz oder triviales Problem	Interferenz oder triviales Problem
	a_{max} Aktivitäten	Interferenz	Interferenz und Reihenfolge

Bild 5-7: Analysekomplexität periodischer Dienste

Bekannte Algorithmen für diese Planungsprobleme gelten immer nur unter bestimmten Randbedingungen, die für die in dieser Arbeit betrachteten Steuergerätenetze nicht zutreffen. Häufig werden Heuristiken eingesetzt [47, 162], bzw. dienen als Anfangslösung für weitere Algorithmen, die jedoch ausschließlich Lösungen für einen Prozessor liefern [159, 44, 45]. Andere Vorschläge wiederum beschränken sich auf unterbrechende Systeme [69], bzw. betrachten nur nichtperiodische Dienste auf einem Multiprozessor-System [47], [1]. Diese Algorithmen liefern in der Regel einen funktionierenden Schedule, können das Funktionieren des Worst-case-Falls allerdings nicht garantieren. Daß probabilistische bzw. statistische Methoden hier ebenfalls nicht weiterhelfen, wurde bereits in Kapitel 3 erwähnt.

Der grundlegende Unterschied der im folgenden vorgestellten Analysemethode zu den oben angesprochenen Methoden besteht darin, daß nicht nur ein beliebiger funktionierender Schedule gesucht wird. Vielmehr besteht das Ziel darin zu zeigen, daß der Worst-case-Fall für ein gegebenes System immer noch funktionsfähig ist. Die Systemanforderungen, die dazu beachtet werden müssen, sind:

- die heterogene Schedulingstruktur, d.h. auf einem betrachteten Prozessor kann entweder ein unterbrechendes oder aber ein nichtunterbrechendes Prioritätensystem realisiert sein. Das Gesamtsystem besteht aus einer Mischung derartiger Prozessoren,
- die Periodizität der Dienste des Systems, die mit einem bestimmten, festen Ankunftsabstand eingeplant werden,
- die Frist eines Dienstes, die nicht an die Periode gekoppelt ist. Es gibt im Gegenteil hier u.U. sehr große Abweichungen (insbesondere bei sog. „Sicherheitsfunktionen“, die sehr selten auftreten, jedoch nur eine sehr kurze Frist zulassen).
- Das Funktionieren des Systems, das für jeden Fall garantiert werden muß.

Kann man die Anwortzeit der Dienste in einem verteilten System bestimmen, dann existiert eine hinreichende Aussage über das Funktionieren des Systems in einer bestimmten Situation. Kann gar der Worst-case der Antwortzeit angegeben werden, so ist dies die Grundlage für eine Garantieaussage, daß das System jederzeit funktioniert.

Nach einer allgemeinen Formulierung dieses Problems sollen in den nachfolgenden Abschnitten Algorithmen entwickelt und präsentiert werden, welche die Berechnung der Worst-case-Antwortzeit zulassen. Der erste, exakte Algorithmus gibt den genauen Wert des Worst-case wieder, ist aber für größere Systeme (bereits für $n > 2$) *NP*-vollständig und somit nicht mehr mit vertretbarem Aufwand berechenbar. Eine berechenbare Alternative dazu ist der im Anschluß vorgestellte Offset-Algorithmus, der eine ausreichende Eingrenzung der Worst-case-Antwortzeit erlaubt.

In den folgenden Abschnitten werden heterogene Systeme untersucht, bei denen ein Prozessor entweder mit einem unterbrechenden oder aber auch mit einem nichtunterbrechenden Betriebssystem (Scheduler) arbeiten kann. Zur Verkürzung der Ausdrucksweise wird von „unterbrechenden Schedulern“ und von „nichtunterbrechenden Schedulern“ gesprochen und damit das Gesamtsystem aus Prozessor, Betriebssystem und darin enthaltenem Scheduler betrachtet.

Aus dem Gesagten läßt sich das grundlegende Planungsziel, die Einhaltung der Echtzeitanforderungen aller Dienste *s* des Systems *S*, formulieren: Die Antwortzeit eines Dienstes R_s muß kleiner sein als dessen Frist D_s. Wird diese Bedingung auf die Worst-case-Antwortzeit r_s verschärft, dann kann die Systemfunktion für jeden Fall garantiert werden. Somit gilt folgender Satz:

Satz 1: Für Steuergerätesysteme S, deren Dienste $s \in S$ Fristen D_s haben, die kleiner oder gleich der Periode der Anforderungen des Dienstes T_s sind, ist das Planungsziel „Echtzeitfähigkeit des Systems“ erreicht, wenn (in Erweiterung von Gl. 3-8) gilt:

$$r_s \leq D_s, \ \forall s \in S. \tag{5-3}$$

Die Antwortzeit muß jetzt mit Hilfe der Taskantwortzeiten des Dienstes bestimmt werden, wenn die Ablaufreihenfolge, wie oben beschrieben, vorgegeben ist und die Ausführung einer Aktivität $A_{s,k}|_a$ vom Ende der Aktivität $A_{s,k}|_{a-1}$ abhängt.

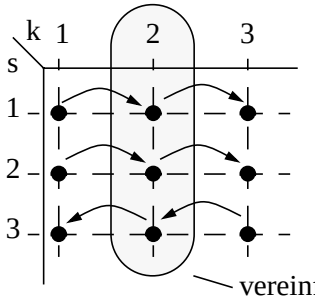
Voraussetzung für die beiden folgenden Algorithmen ist, daß die Prozessoren ausreichend leistungsfähig sind, um die Anforderung nach Gl. 3-1 zu erfüllen.

5.2.1 Ein optimaler Algorithmus

Die Aufgabe eines Algorithmus zur Bestimmung der Worst-case-Antwortzeit eines Dienstes besteht darin, die Konstellation der Aktivitäten auf den Prozessoren zu bestimmen, die für die längste Antwortzeit des betrachteten Dienstes sorgt. Um Worst-case-Situationen erkennen zu können, sollen zunächst einfache Szenarien betrachtet werden. Daraus wird schließlich ein allgemeingültiger Algorithmus abgeleitet.

Zur Validierung der Ergebnisse dieser Analysemethode wurden Simulationen von Steuergerätenetzen durchgeführt, deren Ergebnisse hier als Vergleich dienen (Simulationsbeschreibung siehe Anhang).

Das in Bild 5-8 dargestellte System zeigt ein einfaches Netz aus $n = 3$ Prozessoren, auf dem $m = 3$ Dienste ablaufen. In einem ersten Schritt soll nur Prozessor/Bus $k = 2$ betrachtet werden (grau hinterlegt), d.h. jeder Dienst besteht zunächst aus genau einer Task. In einem weiteren Schritt werden alle Dienste durch die jeweils drei beschriebenen Aktivitäten erbracht.



$G_1 = \{A_{1,1}, A_{1,2}, A_{1,3}\}$ $T_1 = 10$ ms, $k = 1$: up

$G_2 = \{A_{2,1}, A_{2,2}, A_{2,3}\}$ $T_2 = 20$ ms, $k = 2$: up/nup

$G_3 = \{A_{3,3}, A_{3,2}, A_{3,1}\}$ $T_3 = 30$ ms, $k = 3$: up

Bild 5-8: DPA-Matrix und Dienstespezifikation des Beispielsystems (ohne Fristen)

Die Koordinaten der Aktivitäten auf Prozessor $k = 2$ und die mittels Analyse nach Gl. 3-20 (unterbrechende Systeme, up) und Gl. 3-24 (nichtunterbrechende Systeme, nup) berechneten Werte können Tabelle 5-1 entnommen werden. Außerdem werden, ohne Einschränkung der Allgemeinheit, folgende Randbedingungen angenommen:

- Der Offset O ist in allen Fällen 0.
- Das System ist zeitlich so dimensioniert, daß keine zusätzliche Speicherung notwendig ist, d.h. die Antwortzeiten sind höchstens gleich groß wie die Fristen ($D \leq T$ und damit $q = 1$).

Dienst/Aktivität	C in ms	T in ms	Prio	r_{up} in ms	r_{nup} in ms
$A_{1,2}$	1,0	10	0	1	2,4
$A_{2,2}$	1,2	20	1	2,2	3,6
$A_{3,2}$	1,4	30	2	3,6	3,6

Tabelle 5-1: Koordinaten und Analyseergebnisse für ein vereinfachtes Beispielszenario

Um eine simulative Aussage über die Antwortzeit der Dienste zu bekommen, wurde die komplementäre Verteilungsfunktion als Funktion der gemessenen Laufzeit der Dienste aufgezeichnet, d.h. es wird die Wahrscheinlichkeit angegeben, daß ein Dienst nicht länger als die angegebene Zeit zu seiner Fertigstellung braucht. In der Simulation erfolgt die Betrachtung

einer Vielzahl von Dienstanforderungen mit variablen (aber zulässigen) Ankunftsabständen, deren Antwortzeiten gemessen werden. Die Bilder 5-9 und 5-10 zeigen zunächst die Simulationsergebnisse des vereinfachten Systems. Im linken Bild wird ein unterbrechendes System dargestellt, im rechten Bild ein nichtunterbrechendes.

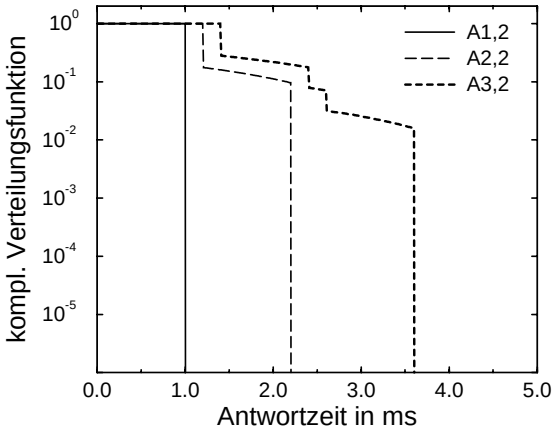


Bild 5-9: Simulation des vereinfachten 3-Dienste Szenarios mit einem unterbrechenden Scheduler

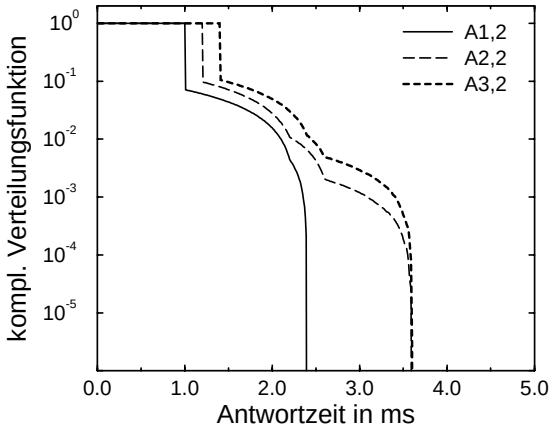


Bild 5-10: Simulation des vereinfachten 3-Dienste Szenarios mit einem nichtunterbrechenden Scheduler

Die gemessenen Verteilungskurven zeigen am Anfang erwartungsgemäß den Wahrscheinlichkeitswert „1“. Dies resultiert aus der Bearbeitungszeit *C* der Task, die in jedem Fall verstreichen muß. Die Forderung an die Analyse ist nun, daß der berechnete Wert immer größer oder maximal gleich den gemessenen Werten sein darf, wenn die Worst-case-Anforderung (und somit die Echtzeitfähigkeit) in allen Fällen gewährleistet werden soll. Der genaue Verlauf der Verteilungsfunktion ist deshalb unerheblich. Es interessieren hauptsächlich die maximalen gemessenen Werte. Zum leichteren Ablesen der gemessenen Maximalwerte wird die Verteilungskurve mit dem letzten, maximalen Wert durch die Abszissenachse geführt.

Die gemessenen Werte stimmen mit den berechneten Werten in Tab. 5-1 genau überein. Anschaulich klar ist, daß im unterbrechenden System die höchstpriore Aktivität $A_{1,2}$ keine weitere Verzögerung über ihre Bearbeitungszeit hinaus erfährt und immer sofort bearbeitet wird. Im nichtunterbrechenden System kann Aktivität $A_{1,2}$ maximal um 1,4 ms durch die anderen Aktivitäten blockiert werden. Überraschen mag eher die Tatsache, daß in Bild 5-10 für die Aktivitäten $A_{2,2}$ und $A_{3,2}$ trotz unterschiedlicher Prioritäten dieselbe Worst-case-Antwortzeit von 3,6 ms ermittelt wird. Dies soll, in Wiederholung der Beziehungen aus Kapitel 3, anhand der Bilder 5-11 und 5-12 kurz verdeutlicht werden. Dabei bildet im Fall von Aktivität

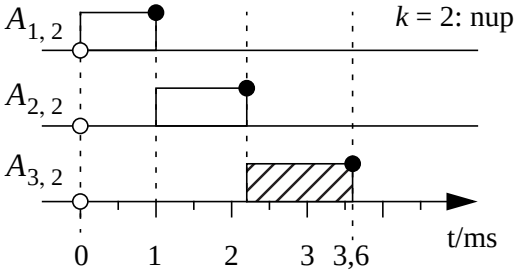


Bild 5-11: Worst-case-Fall für Aktivität $A_{3,2}$

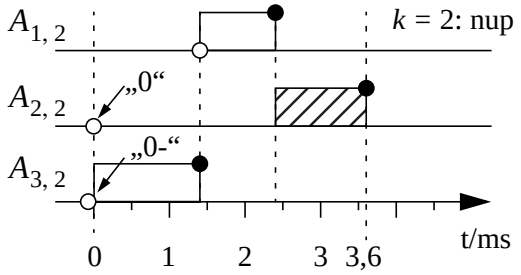


Bild 5-12: Worst-case-Fall für Aktivität $A_{2,2}$

$A_{3,2}$ die Interferenz durch die jeweils höhere Priorität der beiden anderen Aktivitäten den Grund für die Antwortzeit. Im Fall von Aktivität $A_{2,2}$ ist es die Blockierung durch Aktivität $A_{3,2}$, die einen minimal kleinen Zeitraum ($t = 0^-$) vor Aktivität $A_{2,2}$ eingeplant wurde und dann nicht mehr unterbrochen werden kann. (Dieses Verhalten, einer gemeinsamen Worst-case-Antwortzeit der beiden niederpriorsten Aktivitäten in nichtunterbrechenden Systemen, gilt immer, wenn die Perioden der höherprioren Aktivitäten größer sind, als die Wartezeiten der betrachteten Aktivitäten. Satz und Beweis: siehe Anhang A2).

5.2.1.1 Die Worst-case-Antwortzeit von Diensten in verteilten Systemen

Es gibt zur Zeit noch keine Möglichkeit, die Worst-case-Antwortzeit von Diensten in Systemen mit realer Größe ($n > 10$) geschlossen zu berechnen. Deshalb soll im folgenden ein prinzipieller Algorithmus entwickelt werden, der die Ermittlung der Worst-case-Antwortzeit zumindest für kleine Systeme zuläßt. Dazu sollen die Verhältnisse in dem verteilten System betrachtet werden, in dem zwei Prozessoren ($k = 1,3$) mit unterbrechenden Schedulern und einer ($k = 2$) mit einem nichtunterbrechenden Scheduler betrieben werden. Gegenüber den Einprozessorsystemen kommt hier als weitere Anforderung hinzu, daß die entsprechenden Vorgängerrelationen der Aktivitäten berücksichtigt werden müssen. Die beiden ersten Dienste verwenden dazu der Reihe nach die Prozessoren 1, 2 und 3. Der dritte Dienst weist eine andere Abfolge der Prozessoren auf: $A_{3,3} > A_{3,2} > A_{3,1}$. Tabelle 5-2 gibt die vollständigen Koordinaten der Aktivitäten dieses Systems wieder.

Aktivität	C in ms	T in ms	Prio	Aktivität	C in ms	T in ms	Prio
$A_{1,1}$, $A_{1,3}$	1,0	10	0	$A_{1,2}$	1,0	10	0
$A_{2,1}$, $A_{2,3}$	1,0	20	1	$A_{2,2}$	1,2	20	1
$A_{3,1}$, $A_{3,3}$	1,0	30	2	$A_{3,2}$	1,4	30	2

Tabelle 5-2: Koordinaten für das 3-Dienste-Beispielszenario

Ausgangspunkt der Betrachtung ist die Untersuchung, welche *lokalen* Worst-case-Antwortzeiten die einzelnen Aktivitäten A_s eines Dienstes s auf ihrem jeweiligen Prozessor $k(A_s)$ erfahren können. Der lokale Worst-case ist somit die Situation einer einzelnen Aktivität auf einem Prozessor, bei der sie die größte Antwortzeit auf diesem Prozessor erfährt. Ein Dienst mit a_{max} Aktivitäten hat demnach a_{max} lokale Worst-case-Antwortzeiten. Die lokale Worst-case-Situation einer Aktivität läßt ein Maximum der Gesamtantwortzeit des Dienstes entstehen. Dazu muß überprüft werden, wie sich diese Situation auf die übrigen Prozessoren auswirkt, die an diesem Dienst beteiligt sind. Zur Validierung dieser Annahme wird Dienst $s = 3$ näher untersucht und dessen Aktivitäten auf den einzelnen Prozessoren betrachtet.

Das Startszenario ist der lokale Worst-case für Dienst $s = 3$ auf dessen erstem Prozessor ($k = 3$, grau hinterlegt). Dabei wird Aktivität $A_{3,3}$ eingeplant und vor ihrer Beendung von beiden höherprioren Aktivitäten auf diesem Prozessor unterbrochen (siehe Bild 5-13). Das ist die maximale Interferenz, die Aktivität $A_{3,3}$ erfahren kann, da die Zeitverhältnisse in diesem Beispiel keine weitere Möglichkeit zulassen. Die Startzeitpunkte der Aktivitäten können nicht gleichzeitig sein, da die Aktivitäten der Dienste $s = 1$ und $s = 2$ zuvor auf einem nichtunterbrechenden System bearbeitet wurden, welches das „quasi-gleichzeitige“ Abschließen nicht ermöglicht. Jetzt können, ausgehend von den Nachfolgerelationen innerhalb eines Dienstes, die Aktivitäten auf den weiteren Prozessoren konstruiert werden, das hierbei aber zu keinem

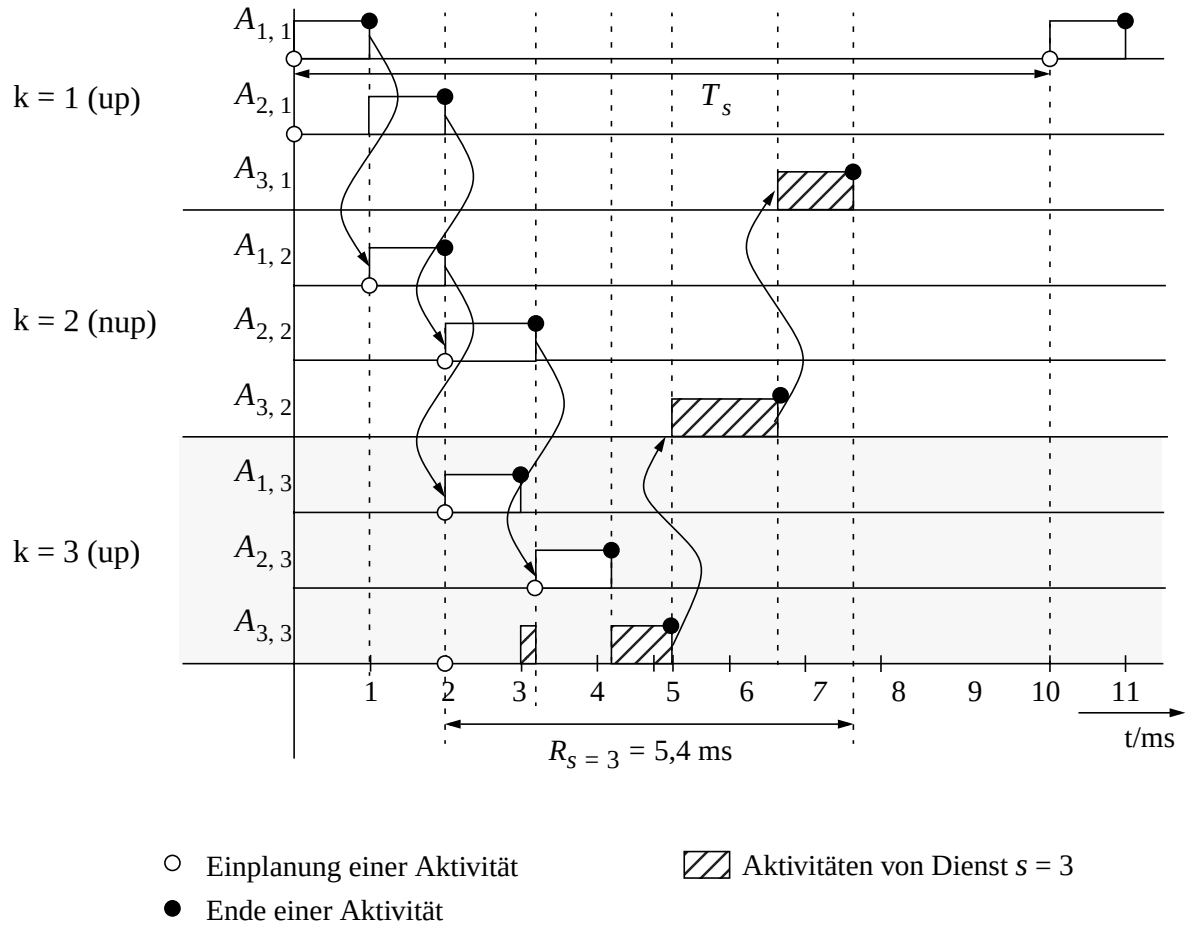


Bild 5-13: Worst-case-Szenario auf dem ersten Prozessor von Dienst $s = 3$

Problem mit der Bedingung führt, daß die Ankunftszeiten der Aktivitäten $A_{k,s}$ nicht geringer sein dürfen als T_s (vgl. Gl. 5-2). Die größten Probleme in diesem Beispiel kann Aktivität $A_{1,1}$ durch ihre Wiedereinplanung verursachen, da sie die kürzeste Periode hat. Diese Aktivität kann aber frühestens zum Zeitpunkt $t = 10$ ms wiederkehren und hat damit keinen Einfluß auf den Ablauf des Dienstes. Für $A_{2,1}$ gilt der entsprechende Sachverhalt. Die ermittelte Antwortzeit für dieses Szenario beträgt 5,4 ms.

Derselbe Vorgang wird für die anderen Prozessoren, die an Dienst $s = 3$ beteiligt sind, ebenfalls durchgeführt. In Bild 5-14 wird das Szenario ausgehend vom Worst-case auf dem zweiten Prozessor dargestellt und in Bild 5-15 wird der schlimmste Fall für den in der Aktivitätenabfolge des Dienstes letzten Prozessor ($k = 1$) betrachtet. Auch hier können jeweils Szenarien geplant werden, ohne daß die Randbedingungen verletzt werden. Die ermittelten Zeiten betragen 5,6 ms und 5,4 ms. Somit kann die Worst-case-Antwortzeit dieses Dienstes zu 5,6 ms angenommen werden, da es keine weiteren Möglichkeiten für die Kombination der Aktivitäten gibt, die eine längere Antwortzeit zur Folge hätten.

Entsprechende Überlegungen können auch für die anderen Dienste des Systems angestellt werden, die zu den Ergebnissen $r_{s=1} = 4,4$ ms und $r_{s=2} = 6,6$ ms (!) führen. D.h. Dienst $s = 3$ hat eine kleinere Worst-case-Antwortzeit als Dienst $s = 2$, obwohl dessen Aktivitäten auf

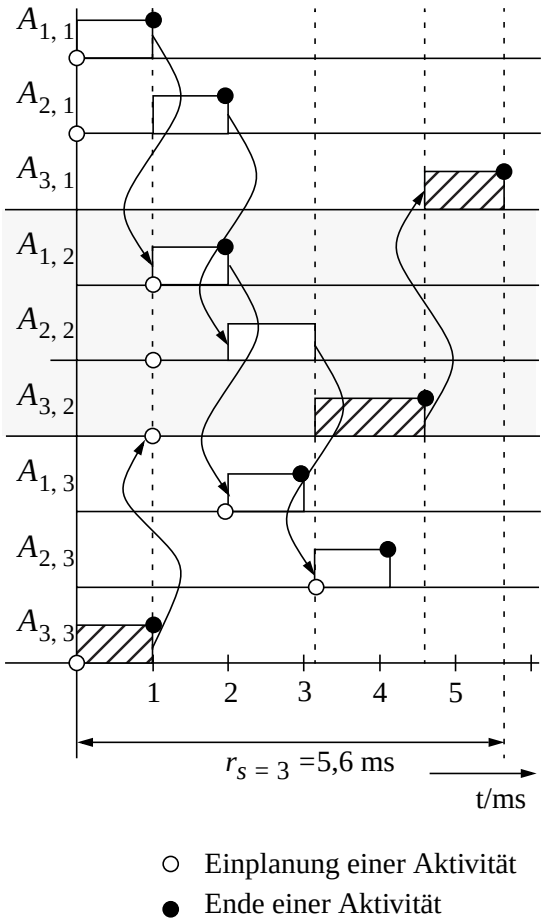


Bild 5-14: Worst-case-Szenario für $s = 3$ auf dem zweiten Prozessor

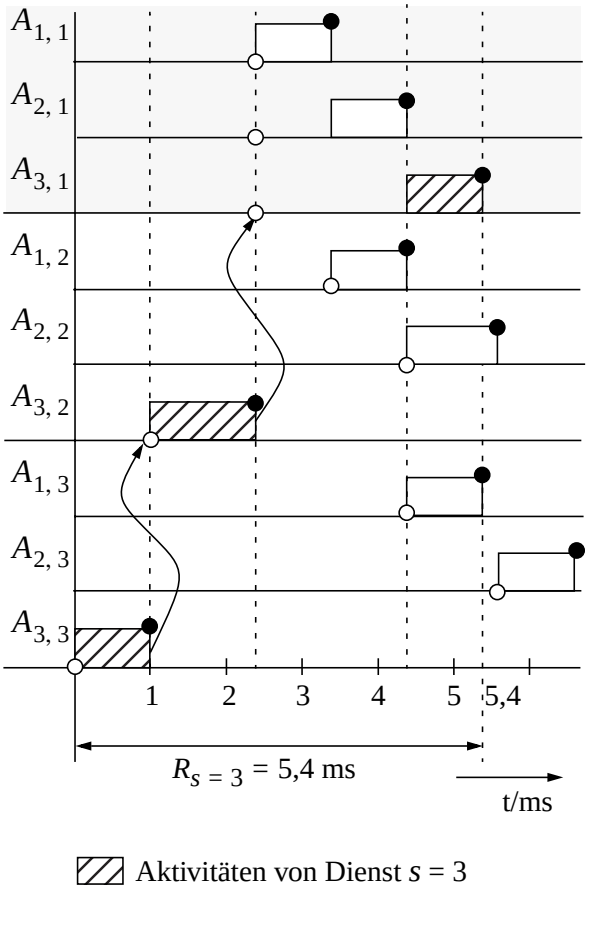


Bild 5-15: Worst-case-Szenario für $s = 3$ auf dem dritten Prozessor

jedem Prozessor eine höhere Priorität als die Aktivitäten von Dienst $s = 3$ haben. Dies rührt daher, daß Dienst $s = 2$ zum einen auf zwei Prozessoren von Aktivitäten des Dienstes $s = 1$ unterbrochen werden kann und daß zum anderen seine Aktivität auf Prozessor $k = 2$ dieselbe Worst-case-Antwortzeit erfährt, wie die von Dienst $s = 3$ (s.o.).[4]

Zur Validierung der vorausgegangenen Überlegungen werden die Simulationsergebnisse in Bild 5-16 herangezogen. Dabei läßt sich erkennen, daß die ermittelten Werte exakt mit den gemessenen Werten in der Simulation übereinstimmen. Hier zeigt sich, daß der zweite Dienst, trotz höherer Priorität als der dritte Dienst, eine größere Worst-case-Antwortzeit als Dienst $s = 3$ besitzt. Dies resultiert wieder, wie oben erwähnt, aus der Blockierung durch die Aktivitäten $A_{3,2}$ und die Interferenz durch Aktivität $A_{1,1}$, während $s = 3$, aufgrund seiner umgekehrten Abarbeitungsrichtung im Vergleich zu den beiden anderen Diensten, nur auf dem zweiten Prozessor gestört werden kann.

In diesem Beispiel ergibt erwartungsgemäß das Szenario den Worst-case, dessen lokaler Worst-case den größten Beitrag zur Dienstantwortzeit liefert.

4. Die Simulationsergebnisse und die dazugehörenden Gantt-Diagramme für dieses Szenario mit ausschließlich unterbrechenden und ausschließlich nichtunterbrechenden Schedulern finden sich im Anhang A1.

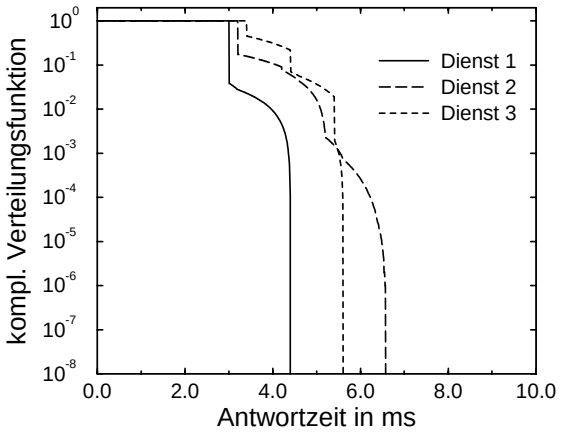


Bild 5-16: Simulation des 3-Dienste-Systems mit einem unterbrechenden und zwei nichtunterbrechenden Schedulern

5.2.1.2 Ermittlung der Worst-case-Antwortzeit von Diensten in verteilten Systemen

Für die Ermittlung des Worst-case-Schedules und die Umsetzung in einen entsprechenden Algorithmus können die folgenden Regeln angegeben werden (falls nicht anders vermerkt, befinden sich die Beweise für die Regeln im Anhang):

1. Es muß für alle Dienste ein gültiger Schedule ermittelt werden. Dies impliziert funktionierende Schedules auf den einzelnen Prozessoren.

2. Ist eine Aktivität auf einem Prozessor eingeplant, so dürfen später keine Zustände entstehen, die dieser Einplanung und dem Ablauf der Aktivität widersprechen. Der Schedule muß dann als ungültig verworfen werden.

3. Für die Abstände T der Einplanungszeitpunkte gilt Gl. 5-2. Treten hier Widersprüche auf, muß der Schedule ebenfalls verworfen werden. Andererseits muß der minimale Einplanungsabstand berücksichtigt werden, da er zu weiterer Interferenz führen kann. Er kann über Gl. 5-2 rekursiv aus der Dienstespezifikation ermittelt werden.

4. Muß ein Schedule verworfen werden, so muß die Worst-case-Bedingung verringert werden, indem die Aktivität(en) nicht weiter dazu berücksichtigt wird, die den geringsten Einfluß auf den Worst-case hat (haben). Ist der errechnete Worst-case auf diesem Prozessor immer noch größer als die anderen prozessorlokalen Fälle, kann wieder ein neuer Schedule konstruiert werden. Hat ein anderer Prozessor einen stärkeren Einfluß, so ist dieser die Grundlage der weiteren Betrachtung. Im schlimmsten Fall müssen alle denkbaren Kombinationen eines Prozessors überprüft werden.

5. Das Ende einer Aktivität auf einem Prozessor $k(A|_a)$ ist der Zeitpunkt der Einplanung auf dem nachfolgenden Prozessor $k(A|_{a+1})$. Das zeitliche Verhalten des Betriebssystems muß dabei, wie in Abschnitt 3.6 beschrieben, entsprechend mitberücksichtigt werden.

6. Befindet sich die Vorgängeraktivität $A|_{a-1}$ einer Aktivität $A|_a$ auf einem nup-Prozessor, dann wurde dieser Vorgängerprozessor auch direkt zuvor über die Bearbeitungszeit C der Vorgängeraktivität hinweg von dieser belegt.

7. Der Worst-case einer Aktivität A auf einem up-Prozessor findet dann statt, wenn mind. eine höherpriore Anforderung auf dem Prozessor zum gleichen Zeitpunkt wie die betrachtete eingeplant wird und alle andern höherprioren Anforderungen vor dem Ende der betrachteten Anforderung eingeplant werden. (Beweis siehe Abschnitt 3.5.3.1.)

8. Beim Worst-case auf einem nup-Prozessor muß zwischen höherprioren und niedrigerprioren Aktivitäten/Diensten unterschieden werden. Der Worst-case für eine Aktivität A_i besteht dann, wenn die längste niedrigerpriore Aktivität ($max(r_h)$ mit $h \in lp(i)$) um einen minimalen Zeitpunkt vor A_i eingeplant wurde und deshalb als erste losläuft und wenn alle höherprioren Aktivitäten A_j mit $j \in hp(i)$ gleichzeitig mit der betrachteten Aktivität A_i oder bis zum Beenden der niederprioren Aktivität eingeplant werden. (Beweise siehe Abschnitt 3.5.3.2, vgl. auch Bild 5-12). Existieren mehrere Alternativen für die blockierende Task, so müssen diese alle betrachtet werden.

9. Arbeitet der Vorgängerprozessor $k(A|_{a-1})$ einer Aktivität $A|_a$ mit einem nup-Scheduler, kann auf dem betrachteten Prozessor $k(A|_a)$ der kritische Zeitpunkt auf dem Prozessor dann nicht erreicht werden, wenn weitere Dienste diese beiden Prozessoren ebenfalls gleichzeitig benutzen. Eine Alternative besteht darin, den Vorgängerprozessor als Ausgangspunkt für den kritischen Zeitpunkt zu wählen.

Bild 5-17 zeigt den Algorithmus, mit dessen Hilfe es prinzipiell möglich ist, den Worst-case eines Dienstes zu bestimmen und damit die Forderung nach Gl. 5-3 zu erfüllen. Der Algorithmus muß für jeden Dienst des Systems durchgeführt werden. Ausgangspunkt ist die Bestimmung der lokalen Worst-case-Antwortzeit jeder Aktivität eines Dienstes. Um den größten Beitrag zur Gesamtantwortzeit zu bestimmen, wird das Verhältnis der lokalen Worst-case-Antwortzeit zur Ausführungszeit gebildet und hiervon über alle Prozessoren des Dienstes das Maximum gesucht:

$$\max_{k = 1, \ldots, a_{max}} \left(\frac{r_{s,k}}{C} \right) \tag{5-4}$$

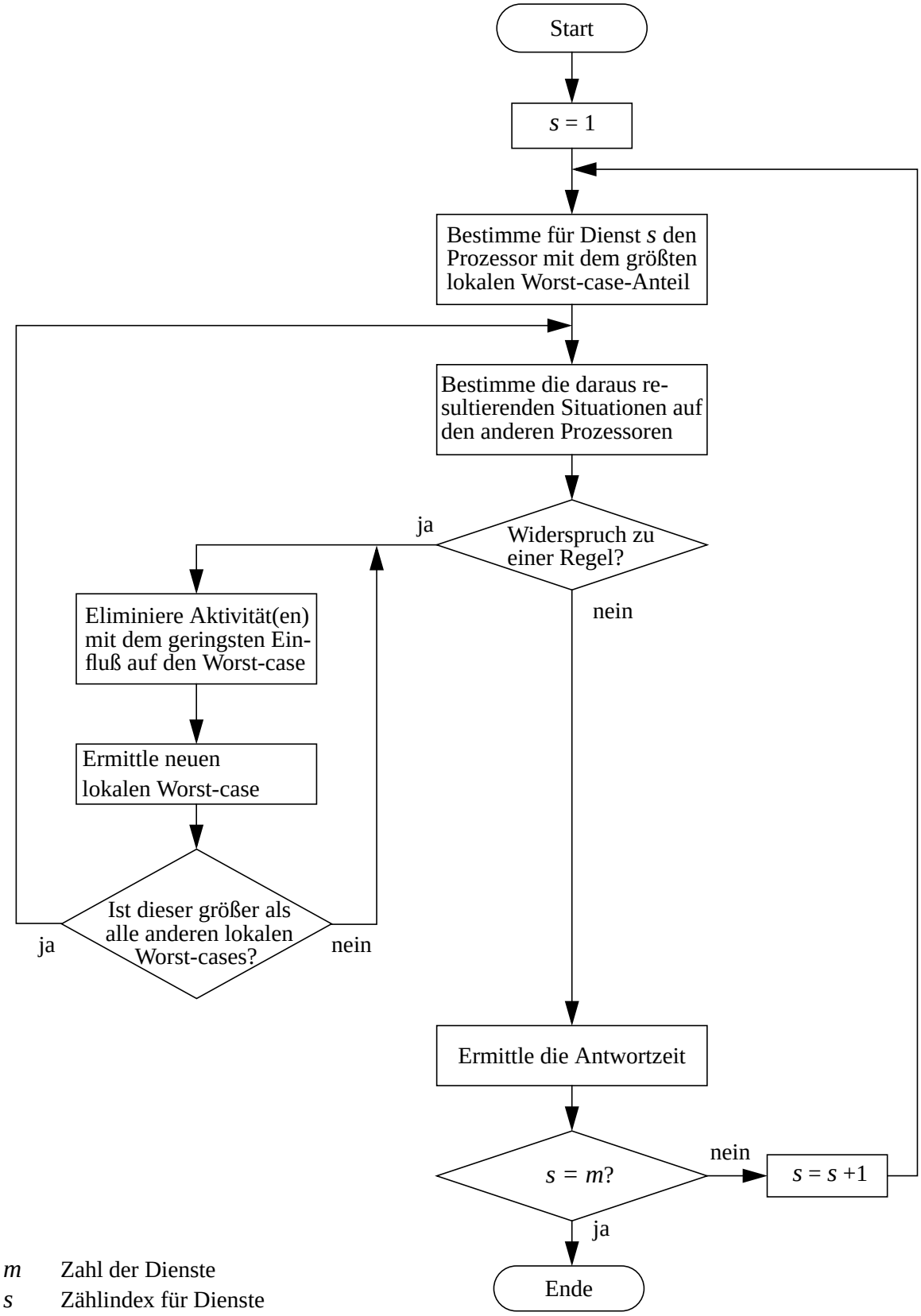
Bei der Einplanung der Aktivitäten auf den Vorgänger- und Nachfolgeprozessoren muß ebenfalls stets versucht werden, den lokalen Worst-case zu erreichen.

Im schlimmsten Fall müssen deshalb alle denkbaren Kombinationen getestet werden. Diese Tatsache macht den Algorithmus für große Systeme, trotz seines exakten Ergebnisses, unbrauchbar.

5.2.2 Der Offset-Algorithmus

Der in diesem Abschnitt vorgestellte Algorithmus zur Bestimmung der Worst-case-Antwortzeit eines Dienstes basiert auf der Separation des beschriebenen Problems, um die Komplexität beherrschbar zu machen. Dabei wird das System wie bisher in seine einzelnen Dienste aufgeteilt. Zusätzlich wird eine dazu orthogonale Sichtweise eingeführt, bei der die Prozessoren getrennt behandelt werden, so daß die Nachfolgerelationen der Aktivitäten aller Dienste des Prozessors nicht mehr vollständig berücksichtigt werden müssen.

Es wird vielmehr folgendes vereinfachtes Modell zugrundegelegt: Jede Aktivität $A_{s,k}|_a$ eines Dienstes *s* auf ihrem Prozessor *k* kann einen Offset *O*, d.h. eine Verzögerung ihres Einplanzeitpunktes, erfahren, der im schlimmsten Fall der Worst-case-Antwortzeit der vorangegangenen



m Zahl der Dienste
s Zählindex für Dienste

Bild 5-17: Ablauf des exakten Algorithmus'

Aktivität $A_s|_{a-1}$ entspricht. Offensichtliche Ausnahme bildet dazu die erste Aktivität eines Dienstes $A_s|_{a=1}$, für die kein Offset durch einen Vorgänger berücksichtigt werden muß.

Für die Berechnung wird es jetzt unerheblich, auf welchem Prozessor *k* diese Vorgängeraktivitäten ablaufen und wie deren jeweilige Konstellation auf diesem Prozessor aussieht. Berücksichtigt werden muß allerdings wiederum die Priorität, mit der die Aktivitäten auf den jeweiligen Prozessoren arbeiten, da diese, wie in Kapitel 3 bereits gesehen, das zeitliche Verhalten der Aktivität steuert. Mit Gl. 3-20 (unterbrechende Systeme) und Gl. 3-24 (nichtunterbrechende Systeme) ergibt sich die Worst-case-Antwortzeit eines Dienstes zu:

$$r_s = r_{a_{max}} = \begin{cases} \max_{q=0,1,2,..}(O_{a_{max}} + P_{a_{max}, q_{amax}} - q_{a_{max}} T_{a_{max}}) & \text{k: up} \\ \max_{q=0,1,2,..}(O_{a_{max}} + W_{a_{max}, q_{amax}} - q_{a_{max}} T_{a_{max}}) & \text{k:nup} \end{cases} \tag{5-5}$$

Zur Berechnung der Aktivitätszeit *P* bzw. der Wartezeit *W* werden die Gleichungen Gl. 3-21 und Gl. 3-25 leicht modifiziert:

$$P_{i,q} = (q+1)C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{P_{i,q}}{T_j} \right\rceil C_j \qquad \text{mit k: up} \tag{5-6}$$

und

$$W_{i,q,nup} = qC_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{W_{i,q}}{T_j} \right\rceil C_j \qquad \text{mit k: nup} \tag{5-7}$$

Das bedeutet, daß gegenüber den Ausgangsgleichungen in beiden Fällen der Offset der höherprioren Aktivitäten bei der Berechnung der Interferenz nicht betrachtet wird. Dies ist ohne Einschränkung zulässig, da ein Offset durch höherpriore Aktivitäten, über die Antwortzeit ihrer Vorgänger hinaus, bedingt durch diesen Ansatz, nicht auftreten kann.

Der Offset O_a einer Aktivität an Position *a* berechnet sich, wie angegeben, aus der Worstcase-Antwortzeit der Vorgängeraktivität ganz allgemein:

$$O_a = r_{a-1}, \forall a \in [2, a_{max}] \tag{5-8}$$

Die allgemeine Rechenvorschrift für r_a lautet somit, in Abhängigkeit des jeweiligen Prozessors *k*:

$$r_a(k) = \max_{q=0,1,2,..}(r_{a-1} + \Re(k)_{a,q_a} - q_a T_a), \forall a \in [1, a_{max}] \tag{5-9}$$

und

$$r_0 = 0. \tag{5-10}$$

$\Re(k)$ ist, abhängig vom Prioritätensystem des jeweiligen Prozessors *k* (up oder nup), entweder die Aktivitätszeit oder die Wartezeit.

Die Berechnung der Antwortzeit mittels dieses Algorithmus erfolgt schrittweise für alle Prozessoren $k(a)$ mit $a \in [1, a_{max}]$ über die einzelnen Aktivitäten des Dienstes, beginnend mit

der Aktivität an Position $a = 1$, die selbst keinen Offset erfährt ($r_0 = 0$). Die berechnete Worst-case-Antwortzeit wird schließlich als Offset für die nachfolgende Aktivität verwendet u.s.w., bis die Aktivität an Position $a = a_{max}$ berechnet wurde. Bild 5-18 zeigt den Ablauf des Offset-Algorithmus‘.

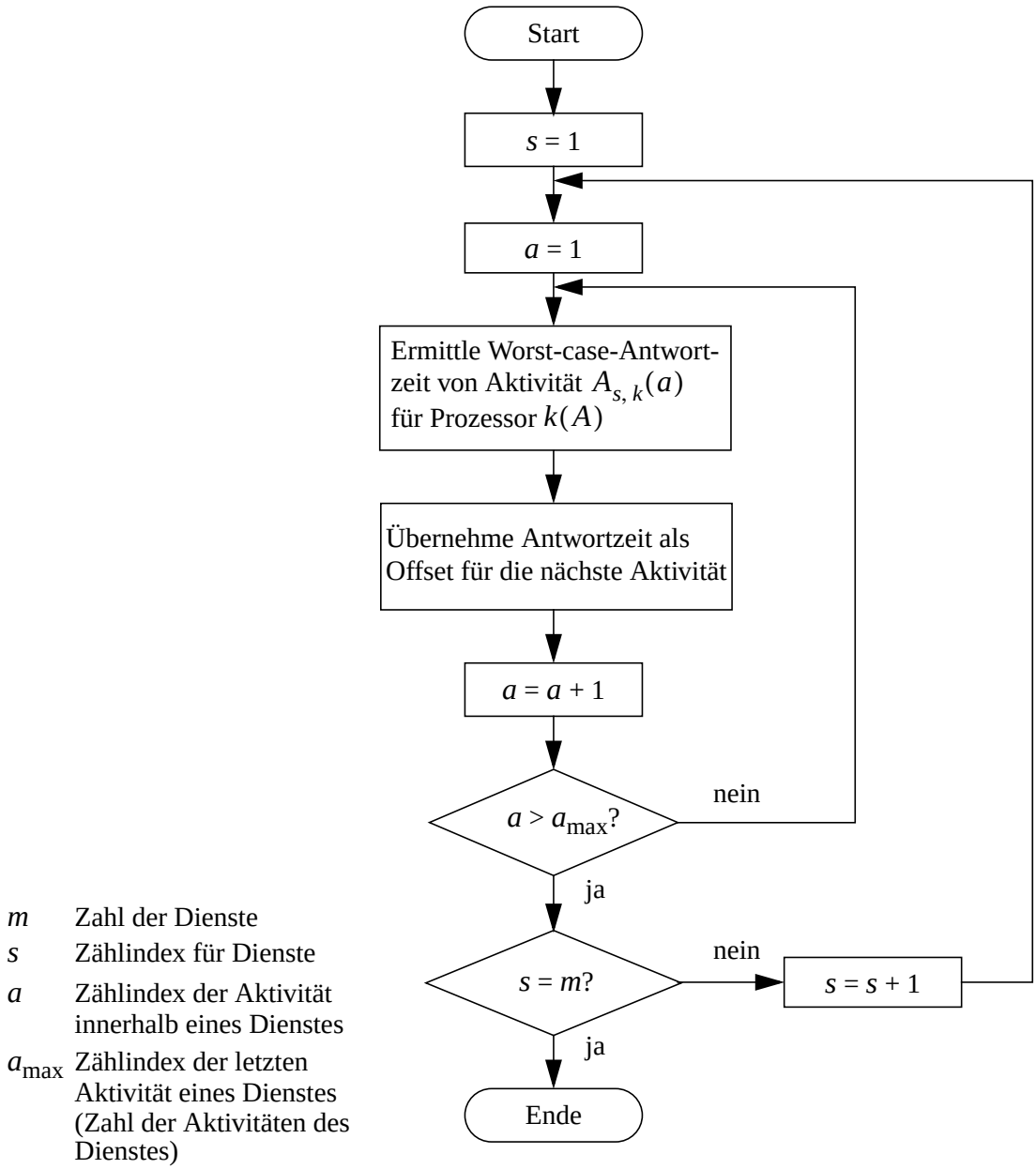


Bild 5-18: Ablauf des Offset-Algorithmus‘

Für das oben vorgestellte Beispiel soll der Algorithmus nun angewendet werden. Die einzelnen Aktivitäten (siehe Bild 5-19) auf einem Prozessor k sind von oben nach unten wieder mit fallender Priorität sortiert. Prozessor $k = 2$ arbeitet mit nichtunterbrechendem, die anderen Prozessoren arbeiten mit unterbrechendem Scheduler. Durch die Separation des Problems auf die einzelnen Prozessoren wird auf allen Prozessoren der Worst-case für alle Dienste angenommen. Im Fall von $s = 3$ heißt das explizit, daß die Sendeaktivität auf Prozessor $k = 3$ aufgrund ihrer niederen Priorität ($p(A_{3,1}) = 2$) im schlimmsten Fall auf die beiden Empfangsaktivitäten $A_{2,1}$ und $A_{1,1}$ warten muß. Dieselbe Situation erfahren die weiteren Aktivitäten des Dienstes

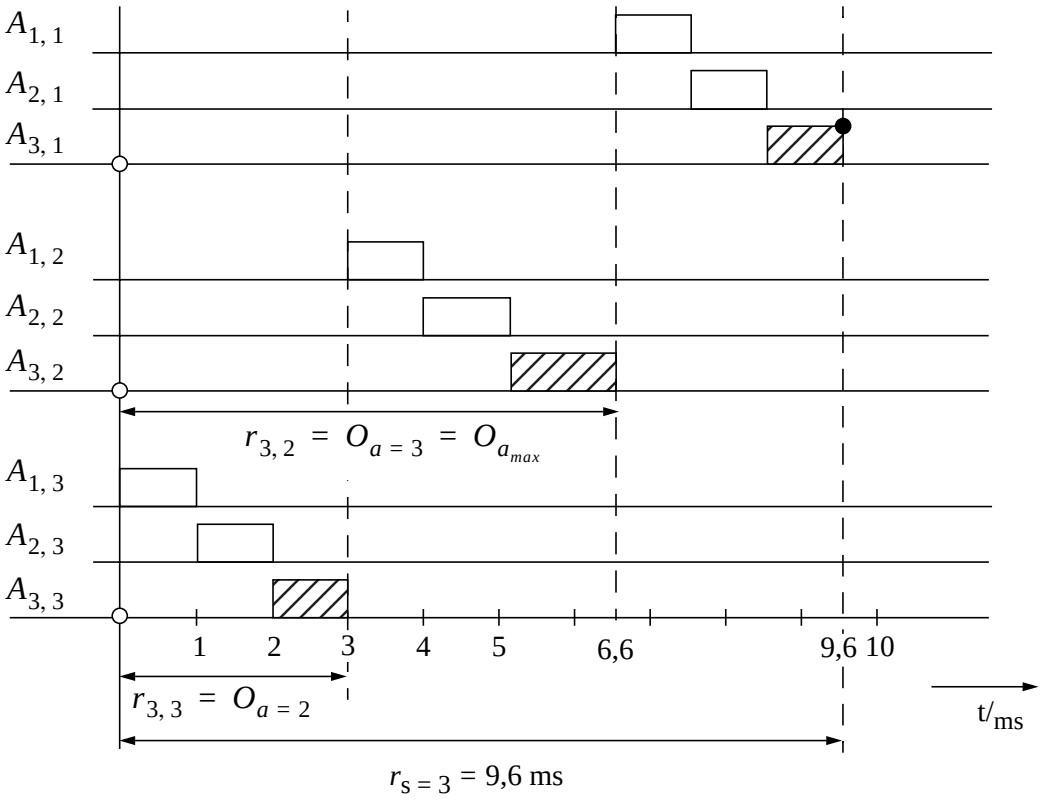


Bild 5-19: Worst-case für den dritten Dienst des Beispielszenarios nach dem Offset-Algorithmus

auf den beiden nachfolgenden Prozessoren, auf denen sie immer erst nach den Aktivitäten der beiden anderen Dienste ablaufen können. Somit ergibt sich eine mit Hilfe von Gl. 5-9 errechnete Antwortzeit des Dienstes von R_s = 9,6 ms.

Tatsächlich aber beträgt die Worst-case-Antwortzeit für Dienst s = 3, wie auch den Simulationsergebnissen in Bild 5-16 entnommen werden kann, nur 5,6 ms.

Tab. 5-3 gibt alle mit dem Offset-Algorithmus analytisch berechneten Werte wieder (weiße Spalten). Außerdem sind die durch den exakten Algorithmus bzw. die Simulation gefundenen Werte aufgeführt (grau hinterlegte Spalten).

3-Dienste-Szenario	$r_{s=1}$ in ms	$r_{s=1}$ in ms	$r_{s=2}$ in ms	$r_{s=2}$ in ms	$r_{s=3}$ in ms	$r_{s=3}$ in ms
nur unterbrechende Scheduler	3	3	6,2	5,2	9,6	5,6
zwei unterbrechende, ein nichtunterbrechender Scheduler (k = 2)	4,4	4,4	7,6	6,6	9,6	5,6
nur nichtunterbrechende Scheduler	6,4	5,2	9,6	5,6	9,6	5,6

Tabelle 5-3: Analyse- und Simulationswerte des 3-Dienste-Szenarios mit dem Offset-Algorithmus

Der Offset-Algorithmus liefert somit eine Aussage über die Worst-case-Antwortzeit eines Dienstes, offensichtlich aber nicht deren untere Schranke, sondern nur eine obere Schranke. Eine Ausnahme bilden die höchstpriorsten Dienste in ausschließlich unterbrechenden Systemen, bei denen auch der Offset-Algorithmus stets die untere Schranke ermittelt. Im folgenden Kapitel wird zu sehen sein, daß die Prioritäten der Aktivitäten eines Dienstes, nach vollzogener Planung, allerdings meist sehr unterschiedliche Prioritäten haben. Dadurch wirkt sich diese Ungenauigkeit zum einen auf alle Dienste des Systems aus und zum anderen wird sie durch die Vermischung der Prioritäten innerhalb eines Dienstes geringer.

5.2.3 Aufwandsabschätzung und Bewertung der Algorithmen

Mit Hilfe der in diesem Kapitel vorgestellten Algorithmen ist man in der Lage, die Worst-case-Antwortzeiten von Diensten, die auf mehrere Prozessoren verteilt sind, zu bestimmen. Um die Einsatzfähigkeit der Algorithmen abschließend bewerten zu können, zählt, außer der Korrektheit des Ergebnisses, auch die Zeit, innerhalb der das Ergebnis ermittelt werden kann (siehe auch Abschnitt 3.2). Betrachtet man vor allem den Zeitaspekt, dann können mit dem optimalen Algorithmus lediglich relativ kleine Systeme analysiert werden. Optimal bedeutet deshalb nur optimal im Bezug auf das ermittelte Ergebnis, nicht aber bezüglich der Zeit, innerhalb der das Ergebnis ermittelt wird. Der Zeitfaktor wiegt noch schwerer, wenn (im Vorgriff auf die Methoden der Prioritätenvergabe im folgenden Kapitel) die Worst-case-Analyse mehrfach durchgeführt werden muß.

Nimmt man die Berechnung der lokalen Worst-case-Antwortzeit einer Aktivität als Basis für die Komplexitätsbetrachtung der Algorithmen, so läßt sich der Aufwand E für den Offset-Algorithmus leicht über die DPA-Matrix als Funktion von Prozessorenzahl n und Dienstezahl m abschätzen[5]:

$$E = E(m \cdot n) \tag{5-11}$$

Das heißt, daß im schlimmsten Fall alle Dienste auf allen Prozessoren vertreten sind. Etwas genauer läßt sich der Aufwand mit Hilfe der Zahl der Aktivitäten pro Prozessor e_k angeben:

$$E = E(\sum_{k=1}^{n} e_k) \tag{5-12}$$

Der Aufwand für den optimalen Algorithmus ist dagegen deutlich höher. Betrachtet man wiederum die Bestimmung der lokalen Worst-case-Antwortzeiten der Aktivitäten, so ergibt sich für E als Funktion der Dienste m, der Zahl der Prozessoren n (Herleitung siehe Anhang A2):

$$E = E\left(mn\left[1 + \sum_{s=0}^{m-1}\binom{m-1}{s}\right]\right) \tag{5-13}$$

Dies alleine macht jedoch nicht alle Probleme aus. Tatsache ist vielmehr, daß dieser Aufwand auch für die Planung der resultierenden Szenarien auf den jeweils benachbarten Prozessoren betrieben werden muß, da für jedes ermittelte Szenario das Überprüfen der aufgestellten Regeln notwendig ist.

5. Dabei muß berücksichtigt werden, daß die Bestimmung der lokalen Worst-case-Antwortzeit selbst um so aufwendiger wird, je mehr Aktivitäten auf einem Prozessor sind und je geringer die Priorität der betrachteten Aktivität ist.

Abschließend kann demnach gesagt werden, daß der Offset-Algorithmus eine sinnvolle und vor allem berechenbare Alternative zur Ermittlung der Worst-case-Antwortzeiten von Diensten eines verteilten Echtzeitprioritätensystems darstellt.

5.3 Diskussion der Methode

Der Ansatz, die Worst-case-Antwortzeit als Grundlage für die Einhaltung der Echtzeitanforderung zu bestimmen, ist notwendig, um eine Garantie für das korrekte Funktionieren eines ereignisgesteuerten Systems geben zu können. Statistische Methoden reichen hier nicht aus, da durch sie ausschließlich eine Wahrscheinlichkeit für das Funktionieren des Systems gegeben werden kann.

Allerdings bleiben durch diese Methode Bandbreite und Systemressourcen ungenutzt. Dies resultiert zum einen aus der notwendigen periodischen Modellierung der Anforderungen und zum anderen aus der Worst-case-Annahme für die Bearbeitungszeit der Aktivitäten. Allerdings gilt dies nur für die in der Planung vorgegebene Menge an harten Echtzeitanforderungen: Existieren weitere Anforderungen im System, für die keine harten Zeitanforderungen gelten, so kann die freie Zeit („slack time“ [149]) auf den Betriebsmitteln/Prozessoren für diese Anforderungen zur Verfügung gestellt werden, ohne daß die Anforderungen mit den harten Randbedingungen davon beeinträchtigt werden. Hat man nichtblockierende unterbrechende Systeme, so müssen keine Randbedingungen betrachtet werden. Bei nichtunterbrechenden Systemen muß die mögliche Blockierung durch eine „weiche“ Anforderung berücksichtigt werden.

Hieraus resultiert ein großer Vorteil der hier betrachteten ereignisgesteuerten TDM-Systeme gegenüber den zeitgesteuerten TDM-Systemen, bei denen bestimmte Zeitscheiben den Anforderungen genau zugeordnet werden. Während zeitgesteuerte TDM-Systeme nicht partiell modifiziert werden können, sondern immer für das komplette System ein neuer Schedule entworfen werden muß, können bei ereignisgesteuerten Systemen Ergänzungen durchgeführt werden, ohne daß andere Komponenten davon betroffen sind. Dabei beschränken sich Ergänzungen nicht nur auf Anforderungen mit weichen Zeitbedingungen. Es können, ohne Beeinträchtigung der bereits vorhandenen Anforderungen, ebenso Dienste mit harten Zeitanforderungen hinzugefügt werden. Allerdings muß vor dem Einsatz der zusätzlichen Anforderungen mittels der hier vorgestellten Methode eine Überprüfung stattfinden.

Kapitel 6

Vergabe und Optimierung von Prioritäten in verteilten Echtzeitsystemen

Mit den im vorangegangenen Kapitel vorgestellten Methoden ist es möglich, ein bestehendes System auf seine Echtzeitfähigkeit hin zu analysieren. Dies alleine bleibt allerdings unbefriedigend, wenn das Ergebnis der Analyse negativ ausfällt. In diesem Kapitel werden deshalb Verfahren entwickelt und vorgestellt, mit denen ein System so modifiziert werden kann, daß es unter bestimmten Umständen die Forderung der Echtzeitfähigkeit erfüllt.

Der zur Planung verfügbare Freiheitsgrad der betrachteten Steuergerätenetze besteht in der Modifikation der den Aktivitäten zugeordneten Prioritäten (sowohl den Tasks als auch den Botschaften auf einem Bus). Wie in Kapitel 3 ausführlich erörtert, stellen gerade die Prioritäten die wesentliche Größe für das zeitliche Systemverhalten dar. Deshalb soll durch geeignete Änderung dieser Prioritäten das gewünschte Systemverhalten erreicht werden. Kann jedoch eine entsprechende Prioritätenverteilung nicht gefunden werden, so muß das System an anderer Stelle optimiert bzw. modifiziert werden, z.B. durch Verwendung leistungsfähigerer Hardware.

Zur Prioritätenvergabe sollen zunächst Heuristiken eingesetzt werden, wie sie bereits in Kapitel 3 vorgestellt wurden. Da Heuristiken unter bestimmten gegebenen Randbedingungen nicht immer befriedigende Ergebnisse liefern, müssen Alternativen zu diesen Verfahren gefunden werden. Der weiterführende und grundlegend neue Ansatz besteht deshalb darin, Optimierungsverfahren zur Vergabe der Prioritäten zu verwenden. Dazu muß zunächst, wie in Kapitel 4 dargelegt, ein Qualitätsmaß für das Gesamtsystem gefunden werden. Hat man ein derartiges Maß, welches möglichst viele Aspekte des Systems berücksichtigt, können die Optimierungsverfahren angewendet werden. Eine weitere Voraussetzung für die Optimierung ist eine geeignete Codierung, insbesondere für die nach naturanalogen Verfahren arbeitenden Methoden. Deren mögliche Anwendung wird am Ende dieses Kapitels diskutiert.

6.1 Heuristische Vergabemethoden

Die in Kapitel 3 vorgestellten Planungsmethoden für Einprozessorsysteme lassen sich mit entsprechender Modifikation auch für ein verteiltes System anwenden. Allerdings liefern sie nicht immer gültige Ergebnisse, insbesondere, wenn reale Randbedingungen (wie z.B. $T_s \neq D_s$, d.h. die Periode T_s eines Dienstes entspricht nicht der Frist innerhalb der er bearbeitet werden muß) beachtet werden müssen. Trotzdem sollen zwei Ansätze präsentiert werden, die auf den

dort vorgestellten Algorithmen beruhen und auf verteilte Systeme erweitert wurden. Diese Verfahren sind interessant, da sie eine einfache, schnell berechenbare Form der Prioritätenvergabe darstellen, die bei Systemen ohne harte Anforderungen ausreichend ist. An einem einfachen Beispiel wird die für reale Systeme begrenzte Einsatzfähigkeit dieser Algorithmen gezeigt.

6.1.1 Das Distributed-Rate-Monotonic-Verfahren

Basierend auf der Modellbildung in Kapitel 5 kann das Rate-Monotonic-Verfahren auch für verteilte Systeme angewendet werden. Das resultierende *Distributed-Rate-Monotonic*-(DRM)-Verfahren macht sich dazu die Propagierung von Diensteigenschaften an ihre Aktivitäten zunutze. Es wird eine prozessorlokale Vergabe der Prioritäten vorgenommen, welche von der übernommenen Periode des Dienstes abhängt. Werden die Dienste mit steigender Periode geordnet, so werden die daraus resultierenden Prioritäten an die Aktivitäten der Dienste auf den lokalen Prozessoren weitergegeben.

6.1.2 Das Distributed-Deadline-Monotonic-Verfahren

Beim *Distributed-Deadline-Monotonic*-(DDM*)-Verfahren* sieht das Vorgehen so aus, daß die vollständigen Dienste entsprechend ihrer Fristen geordnet werden. Der kürzeste Abstand zwischen der Bearbeitungszeit eines Dienstes und seiner Frist führt zum ersten Platz dieses Dienstes in der Ordnung und somit auch zur höchsten Priorität. Diese Ordnung und die damit einhergehenden Prioritäten werden von den Aktivitäten des Dienstes auf den Prozessoren übernommen.

6.1.3 Vergleich der Verfahren

Das in Kapitel 5 bereits dargelegte Beispiel soll die Anwendung der beiden Algorithmen verdeutlichen: Ein Steuergerätenetz bestehe aus drei Diensten $X = \{1, 2, 3\}$, die auf drei Prozessoren $K = \{1, 2, 3\}$ ablaufen (siehe Dienstespezifikation in Tabelle 6-1). Alle Prozessoren arbeiten mit unterbrechenden Schedulern (*k*: up).

Dienst *s*	T_s in ms	D_s in ms
1	10	5
2	20	5
3	30	10

Tabelle 6-1: Dienstespezifikation des Beispielsystems mit Fristen

Tabelle 6-2 gibt, nach erfolgter Transformation der Dienstespezifikation entsprechend Kapitel 5, die Aktivitätenspezifikation (Bearbeitungszeit *C* und Periode *T,* jeweils in ms) aller neun notwendigen Aktivitäten des Systems wieder. Außerdem sind in der Spalte D_s nochmals die Fristen der Dienste aufgeführt. In den nachfolgenden Spaltengruppen werden drei mögliche Prioritätsverteilungen untersucht und ihre Auswirkungen dargestellt. Die erste Spalte *p* gibt die entsprechend des verwendeten Algorithmus ermittelte Priorität der Aktivität wieder. Die Spalte r_s gibt die resultierende Worst-case-Antwortzeit nach dem Offset-Algorithmus wieder. In Klammern stehen dazu außerdem die mit dem optimalen Algorithmus ermittelten

Aktivität	C	T	D_s	*DRM*			*DDM*			*Alternativ*		
				p	r_s	L	p	r_s	L	p	r_s	L
$A_{1,1}$	1,0	10	**5**	0	3 (3)	2 (2)	1	6,2 (5,2)	**-1,2 (-0,2)**	1	5 (5)	0 (0)
$A_{1,2}$	1,0	10		0			1			0		
$A_{1,3}$	1,0	10		0			1			1		
$A_{2,1}$	1,0	20	**5**	1	6,2 (5,2)	**-1,2 (-0,2)**	0	3,2 (3,2)	1,8 (1,8)	0	4,2 (4,2)	0,8 (0,8)
$A_{2,2}$	1,2	20		1			0			1		
$A_{2,3}$	1,0	20		1			0			0		
$A_{3,1}$	1,0	30	10	2	9,6 (5,6)	0,4 (4,4)	2	9,6 (5,6)	0,4 (4,4)	2	9,6 (5,6)	0,4 (4,4)
$A_{3,2}$	1,4	30		2			2			2		
$A_{3,3}$	1,0	30		2			2			2		

Tabelle 6-2: Antwortzeiten bei heuristischer Prioritätenvergabe DRM und DDM

Werte. Die letzte Spalte einer Gruppe, *L (Laxity,* vgl. Abschnitt 6.2.1*)*, gibt die Differenz zwischen Antwortzeit und Frist wieder. Ist diese Differenz negativ, dann überschreitet die Antwortzeit die Frist und der Dienst kann somit die Einhaltung der Echtzeitanforderung nicht garantieren.

Die Gruppe *DRM* legt die Verteilung der Prioritäten unabhängig von der vorgegebenen Frist *D* nach dem DRM-Algorithmus fest.

Die Algorithmen liefern in beiden Fällen Systeme, welche die Anforderungen nicht einhalten können. Bei der Vergabe nach dem DRM-Verfahren kann die Frist $D_{s=2}$ = 5 ms des zweiten Dienstes nicht eingehalten werden; die Laxity des Dienstes ist negativ (-1,2). Die Vergabe nach dem DDM-Verfahren verletzt die Zeitanforderung an den ersten Dienst ($D_{s=1}$ = 5 ms). Auch hier ergibt sich folglich eine negative Laxity. Daß dies nicht an der Verwendung des Offset-Algorithmus' liegt, ergeben die Simulationsergebnisse und die Bestimmung der Worst-case-Antwortzeiten der Dienste mittels optimalem Algorithmus (siehe Anhang für die DRM-Szenarien). Auch die dort gewonnenen Werte überschreiten die vorgegebenen Fristen (vgl. Werte in den Klammern). Mit diesen Ergebnissen kann folgende Aussage formuliert werden:

„Heuristische Scheduling-Algorithmen, die für einen Prozessor eine optimale Verteilung der Prioritäten ermitteln, liefern nicht notwendigerweise einen gültigen Schedule bei ihrer Anwendung in verteilten Systemen."

Werden Aktivitäten A_k auf einem Prozessor *k* nach dem DRM- oder RM-Algorithmus vergeben, so erfahren sie die kürzesten möglichen Antwortzeiten für die zugehörige Aktivitätenmenge auf dem Prozessor. Dies reicht jedoch nicht in allen Fällen für die Zeitanforderungen an einen Dienst *s* aus. Vielmehr kann es notwendig sein, eine Aktivität zugunsten einer anderen Aktivität auf einem Prozessor zu benachteiligen, wenn erst dadurch die beiden dazugehörenden Dienste in der vorgegebenen Zeit bearbeitbar sind. Eine mögliche Verteilung der Prioritäten, bei der alle Anforderungen eingehalten werden, zeigt die letzte Spaltengruppe (*Alternativ*). Bild 6-1 gibt für diese Prioritätenvergabe die Simulationsergebnisse wieder.

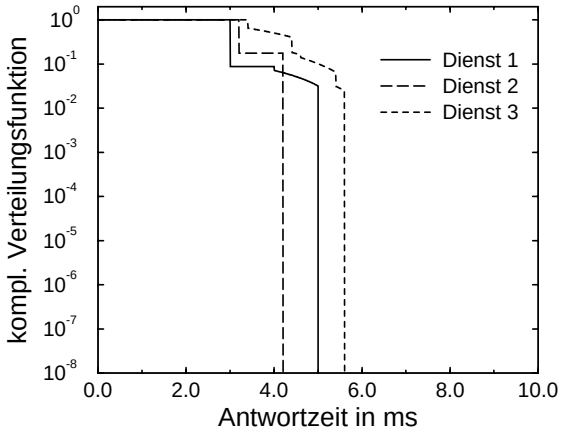


Bild 6-1: Das 3-Dienste-Beispielsystem mit Prioritäten, welche die Zeitbedingungen erfüllen

Zur weiteren Charakterisierung des Systems kann aus Tabelle 6-3 die Belastung der Prozessoren im Beispielsystem entnommen werden. Die Belastung ist allerdings, wie in Kapitel 3 gesehen, unabhängig von der verwendeten Prioritätenverteilung.

Prozessor *k*	1	2	3
Belastung ρ	18,33 %	20,67%	18,33 %

Tabelle 6-3: Belastung der Prozessoren im 3-Dienste-Szenario

Bemerkenswert hierbei ist die geringe Auslastung der Prozessoren, die offensichtlich als Kriterium nicht ausreicht, um die Zeitbedingungen der Dienste erfüllen zu können.

6.2 Prioritätenvergabe mittels Optimierungsmethoden

In diesem Abschnitt werden Optimierungsmethoden und -verfahren dazu verwendet, die Prioritäten in einem Steuergerätenetz zu vergeben. Um diese Methoden einsetzen zu können, sind, wie in Kapitel 4 bereits erläutert, drei Grundbedingungen zu erfüllen. Einmal muß das Optimierungsziel festgelegt werden. Damit einher geht die Erstellung einer Qualitätsfunktion. Schließlich muß sowohl der Optimierungsablauf festgelegt als auch eine geeignete Systemcodierung gewählt werden.

Nach der Vorstellung dieser Komponenten in den nächsten Unterabschnitten werden die Eigenschaften der eingesetzten Optimierer am Ende des Abschnitts anhand eines realen Systems miteinander verglichen.

6.2.1 Die Qualitätsfunktion

Das formale Planungsziel wurde bereits in Kapitel 5 (vgl. Gl. 5-3) formuliert. Daraus läßt sich einfach ein Qualitätsmaß für ein gegebenes System ableiten: „Das System hat eine ausreichende Qualität, wenn alle zeitlichen Randbedingungen erfüllt werden.“ Somit wird nicht nach

einer globalen maximalen Qualität gesucht, sondern nach einer Qualität, welche die gestellten Randbedingungen erfüllt.

Um dieses Qualitätsmaß in eine entsprechende Funktion umzusetzen, sollen allgemeine und die für das Prioritätenoptimierungsproblem notwendigen speziellen Randbedingungen an eine Qualitätsfunktion postuliert werden. Die in Kapitel 4 vorgestellten Optimierungsmethoden verlangen zwar keine explizit aussagekräftigen Funktionen. Es ergibt sich jedoch der Vorteil bei einer derartigen Qualitätsfunktion, schon in einem frühen Stadium zu erkennen, ob man eine sinnvolle Lösung erhalten hat. Die Randbedingungen sind deshalb im einzelnen:

1. Der durch die Funktion ermittelte Qualitätswert soll ein Maß dafür sein, wie „gut“ die Fristen des Gesamtsystems erfüllt werden. Anhand dieses Maßes muß ein Optimierer erkennen können, ob das Optimierungsziel erreicht wurde, oder ob weitere Modifikationen für eine Verbesserung nötig sind.

2. Für jeden Systemzustand muß eine Qualität angebbar sein, d.h. auch für letale Systeme soll ein Qualitätswert existieren. Die dazu notwendige Qualitätsfunktion ist aber nicht notwendigerweise eine stetige Funktion.

3. Werden Fristen verletzt, dann muß der Qualitätswert schlechter sein als bei der schlechtesten Prioritätenkombination, bei der alle Fristen erfüllt werden. Dies stellt die Grundbedingung für das Funktionieren des Echtzeitsystems dar.

4. Eine bessere Prioritätenvergabe soll durch einen höheren Qualitätswert ausgedrückt werden. Diese Anforderung ist notwendig, um nicht mit einem lokalen Maximum vorlieb nehmen zu müssen, sondern vielmehr das globale Maximum erkennen zu können.

Die Grundlage für die Bestimmung der Qualitätsfunktion bildet die bereits erwähnte Laxity L (siehe Bild 6-2) eines Dienstes. Das ist die Zeit, die zwischen dem Beenden des Dienstes und

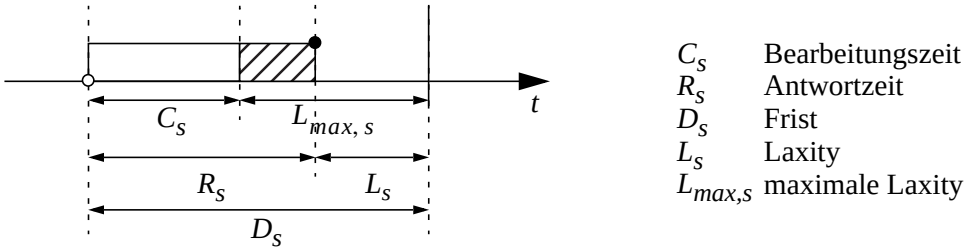


Bild 6-2: Zeitbeziehungen zur Festlegung der Laxity eines Dienstes

dem Erreichen der Frist des Dienstes liegt. Sie kann dabei um den im Bild schraffierten Bereich, der durch die summierten Interferenzen und Blockierungszeiten des Dienstes gebildet wird, schwanken und berechnet sich wie folgt:

$$L_s = D_s - R_s \tag{6-1}$$

Der Maximalwert, den die Laxity idealerweise annehmen kann, ist deshalb die Differenz zwischen Frist und Bearbeitungszeit (d.h. die Antwortzeit des Dienstes entspricht seiner Bearbeitungszeit):

$$L_{max,\,s} = D_s - C_s \tag{6-2}$$

Eine hohe Dienstqualität wird durch eine „relativ“ hohe Laxity gekennzeichnet. Bezieht man alle Dienste *s* des Systems in die Funktion mit ein, indem man den Mittelwert der normierten Laxity verwendet, ergibt sich folgender Grundansatz für die Darstellung der Qualität *Q* über alle *m* Dienste des Systems:

$$Q = \frac{1}{m} \sum_{s=1}^{m} \frac{L_s}{L_{max,\,s}} = \frac{1}{m} \sum_{s=1}^{m} \frac{D_s - R_s}{D_s - C_s} \tag{6-3}$$

Die Änderungen der Antwortzeit R_s in Abhängigkeit von der Prioritätenverteilung in dem für die Optimierung wichtigen Bereich $R = D \pm \delta;\ \delta \ll D$, ist relativ gering. Aus diesem Grund wird die Empfindlichkeit in R_s durch Quadrieren der einzelnen Elemente erhöht:

$$Q = \frac{1}{m} \sum_{s=1}^{m} \frac{D_s^2 - R_s^2}{D_s^2 - C_s^2} \tag{6-4}$$

Die Qualitätsfunktion nach Gl. 6-4 ist relativ einfach und spiegelt nur unzureichend die Komplexität eines realen Systems wider. Deshalb sollen weitere Systemeigenschaften in die Bewertung durch diese Funktion mit einbezogen werden. Eine dieser Eigenschaften ist die Prozessorlast, die eine Aktivität im Verhältnis zur Gesamtauslastung des Prozessors verursacht. Dazu wird die relative Belastung durch eine Aktivität wie folgt festgelegt:

$$\rho_{rel,\,s,\,a} = \frac{\rho_{s,\,a}}{\rho_k} = \frac{C_{s,\,a}}{T_s \rho_k} \tag{6-5}$$

Die Größe $\rho_{rel,\,s,\,a}$ gibt den Anteil an der Prozessorlast ρ_k wieder, der durch die Aktivität $A_{s,\,k}\big|_a$ verursacht wird. *a* gibt dabei die Abfolgeposition der Aktivität innerhalb ihres Dienstes *s* an. Gemittelt über alle Aktivitäten eines Dienstes ergibt sich dann die relative Belastung, die ein Dienst im Gesamtsystem verursacht, zu:

$$\rho_{rel,\,s} = \frac{1}{a_{max}} \sum_{a=1}^{a_{max}} \rho_{rel,\,s,\,a} \tag{6-6}$$

Diese relative Belastung ist relevant bei der Suche nach dem Prozessor, bei dem die Änderung der Prioritäten am vielversprechendsten ist. Verursacht eine Aktivität auf einem Prozessor eine hohe Auslastung, so kann die Antwortzeit der Aktivität und damit des ganzen Dienstes durch Veränderung dieser Aktivität stark beeinflußt werden. Berücksichtigt man diese Eigenschaft bei der Qualitätsberechnung, indem die Dienste mit ihrer relativen Belastung gewichtet werden, so sinkt die Qualität mit fallendem Anteil des Dienstes an der Gesamtrechenzeit des Systems. Gl. 6-4 wird damit erweitert zu:

$$Q = \frac{1}{m} \sum_{s=1}^{m} \rho_{rel,\,s} \frac{D_s^2 - R_s^2}{D_s^2 - C_s^2} \tag{6-7}$$

Randbedingung 3 für die Erstellung von Qualitätsfunktionen verlangt, daß kein Qualitätswert eines nichtfunktionierenden Schedules höher sein darf, als die Qualität einer funktionierenden Prioritätenvergabe. Deshalb wird eine Abwertefunktion $\sigma_s(L)$ eingeführt, welche die Qualität eines Dienstes stärker vermindert, wenn er seine Frist nicht einhält und somit die Laxity nega-

tiv wird. Um garantieren zu können, daß die Abwertefunktion ausreichend abwertet, wird durch sie der theoretisch erreichbare maximale Qualitätswert Q_{max} des Systems abgezogen. Q_{max} tritt ein, wenn die Antwortzeiten aller Dienste ihren Ausführungszeiten entsprechen und berechnet sich deshalb wie folgt:

$$Q_{max} = \frac{1}{m}\sum_{s=1}^{m} \rho_{rel,\,s} \qquad \forall R_s = C_s,\ s = 1, 2, \ldots, m \tag{6-8}$$

Damit ergibt sich für die Penalty-Funktion $\sigma_s(L) \cdot (mQ_{max} - \rho_{rel,\,s})$, d.h. vom Maximalwert wird der eigene Anteil des Dienstes an diesem Wert wieder abgezogen. Damit wird erreicht, daß negative Qualitätswerte grundsätzlich nichtfunktionierende Systeme kennzeichnen.

Die daraus resultierende Qualitätsfunktion hat schließlich das folgende Aussehen:

$$Q = \frac{1}{m}\sum_{s=1}^{m}\left[\rho_{rel,\,s}\frac{D_s^2 - R_s^2}{D_s^2 - C_s^2} - \sigma_s(L)(mQ_{max} - \rho_{rel,\,s})\right] \text{ mit } \sigma_i(L) = \begin{cases} 0 & L \geq 0 \\ 1 & L < 0 \end{cases} \tag{6-9}$$

Mit dieser Qualitätsfunktion kann garantiert werden, daß kein Qualitätswert für ein nichtfunktionierendes System existiert, der größer ist als der Qualitätswert des schlechtesten, funktionierenden Systems (Beweis: siehe Anhang). Außerdem haben funktionierende Systeme immer einen positiven Qualitätswert.

Bild 6-3 zeigt den Verlauf der Qualitätsfunktion des 3-Dienste-Beispielsystems (n = 3 Prozessoren, e_k = 3 Aktivitäten auf jedem Prozessor) aus Kapitel 5. Für dieses einfache System können alle $\prod_{k=1}^{n} e_k! = 216$ Systemqualitäten berechnet werden. Somit repräsentiert jeder Punkt die Qualität einer bestimmten Prioritätenverteilung.

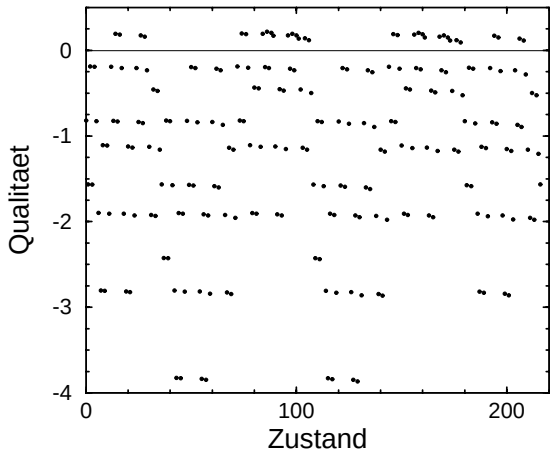


Bild 6-3: Verlauf der Qualitätsfunktion für das 3-Dienste-Beispielsystem

Die Abfolge der Zustände resultiert aus der Permutation aller möglichen Prioritätenverteilungen. Zur Veranschaulichung wurde die Grenzlinie bei Q = 0 eingezogen, so daß alle 32 Punkte, die oberhalb (oder genau auf) dieser Linie liegen und gültige Schedules darstellen, leicht erkennbar sind. Diese Lösungen stellen garantiert funktionierende Systeme dar.

Der Verlauf der Qualitätsfunktion zeigt das typische Aussehen für ein diskretes kombinatorisches Optimierungsproblem. Auffällig sind Qualitäts-„Plateaus“, bei welchen sich die Qualität zwischen den einzelnen Zuständen nicht sehr stark ändert (vgl. dazu auch den Verlauf der Qualitätsfunktion des SAE-Systems in Bild 6-8). Außerdem existieren diese Plateaus nur auf bestimmten diskreten Niveaus, zwischen denen die Funktion springt. Das Wissen über das Aussehen der Funktion kann später dazu verwendet werden, qualitativ höherwertige Lösungen für ein System zu finden: Hat man durch eines der in Abschnitt 6.2.5 vorzustellenden Verfahren ein Plateau erreicht, das eine gültige Lösung für das System darstellt, dann kann u.U. durch die Suche im lokalen Bereich dieser Lösung auf dem Plateau eine noch bessere Lösung für das System gefunden werden. Diese Suche kann solange fortgesetzt werden, bis der Rand des Plateaus erreicht wird und die Qualität der nächsten Lösung wieder stark abnimmt.

6.2.2 Steuerung der Systemmodifikation

Für die nachfolgend untersuchten Optimierer Threshold Accepting (TA) und Simulated Annealing (SA) muß der Systemzustand in den einzelnen Iterationsschritten geeignet verändert werden (vgl. Abschnitt 4.4.1). Dazu wird die Modifikationstiefe δ eingeführt. Sie gibt, in Abhängigkeit der Zahl an Iterationen *i*, die Zahl der zufällig gewählten Aktivitätenpaare mit der Unterbrechungsdistanz „1“ an, die miteinander vertauscht werden. Bei dieser zufälligen Auswahl werden alle Prozessoren des Systems berücksichtigt und die Prioritäten der gewählten Aktivitäten entsprechend gewechselt. Die Modifikationstiefe wird durch folgende Beziehung bestimmt:

$$\delta(i) = \left\lceil \delta_0 \exp(-(\sqrt{i})/c) \right\rceil \tag{6-10}$$

Wobei δ_0 die Anfangstiefe markiert und *i* der Iterationszähler ist. Mit *c* steht ein Steuerparameter zur Verfügung, der eine Anpassung der Funktion an die Größe des Systems und somit an die zu erwartende Iterationenzahl erlaubt. Durch die Abhängigkeit von der Zahl der Iterationen erzielt man damit den gewünschten Effekt von immer kleiner werdenden Schrittweiten zwischen zwei Systemzuständen. Beim Simulated Annealing fließt diese Schrittweite außerdem in die Bestimmung der Akzeptanzwahrscheinlichkeit mit ein. Sie entspricht dort der verringerten Temperatur des Ausglühvorgangs.

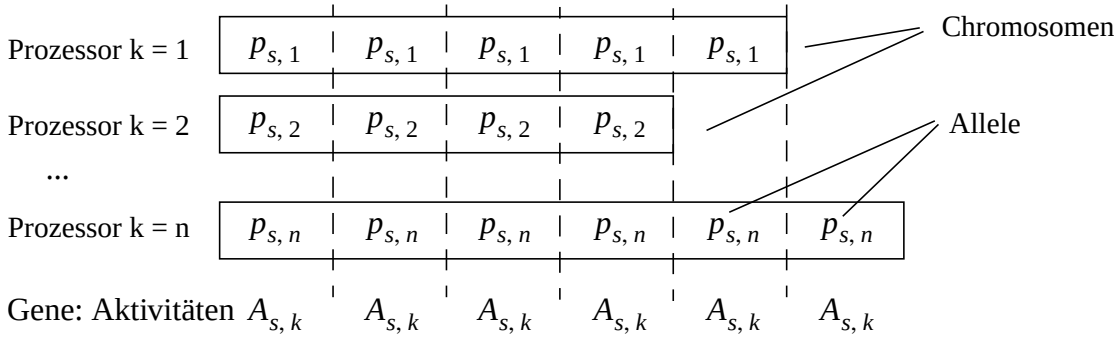
Die Verwendung einer Exponentialfunktion gibt am genauesten die Verhältnisse eines natürlichen Abklingvorgangs wieder. Durch die Verwendung des Dachoperators erreicht man eine gewisse Robustheit der Folge, da der Wert nie unter 1 absinken kann und so auch bei großen Iterationszahlen eine weitergehende Permutation gewährleistet ist. Alternative Möglichkeiten sind Funktionen der Form:

$$\delta(i) = \left\lceil \delta_0 c^i \right\rceil \qquad 0 < c < 1 \tag{6-11}$$

Die Anfangstiefe δ_0 muß, genau wie der Steuerparameter *c*, anhand des zu optimierenden Systems festgelegt werden. Empirische Untersuchungen an mehreren Systemen zeigten, daß für δ_0 die Zahl der im System befindlichen Prozessoren *n* ein sinnvoller Wert ist und *c* so dimensioniert wird, daß nach ca. der Hälfte der zu erwartenden Iterationen (siehe Abschnitt 6.2.5.1) die Modifikationstiefe bei 1 angelangt ist.

6.2.3 Codierung des Systems

Damit Optimierungsalgorithmen (insbesondere die evolutionären Algorithmen) eingesetzt werden können, muß eine Codierung des Kommunikationsnetzes erfolgen. In Analogie zu den in Kapitel 4 vorgestellten natürlichen Codierverfahren wird folgende Dekomposition und anschließende Codierung gewählt: Das Gesamtsystem *S* entspricht einem Individuum der Population Π aller möglichen Systeme. Es setzt sich aus einer Anzahl von *n* Prozessoren und *m* Diensten mit den zugehörigen Aktivitäten zusammen (vgl. DPA-Matrix aus Kapitel 5), die zunächst durch Chromosomen repräsentiert werden müssen. Berücksichtigt man die Randbedingung, daß die Zahl und Zuordnung der Aktivitäten auf Dienste und Prozessoren in einem System fest vorgegeben ist, dann sind nicht alle Codierungsmöglichkeiten gleich sinnvoll. Es wurde deshalb die Codierung „Prozessor –> Chromosom“, „Aktivität –> Gen“ und „Priorität –> Allel“ gewählt. Die Prioritäten der Aktivitäten sind die Allele auf den Chromosomen und damit durch die Mutation veränderbar (siehe Bild 6-4).



$A_{s,k}$ Aktivität (Gen) — s Dienst, mit $1 \le s \le m$

$p_{s,k}$ Priorität der Aktivität (Allel) — k Prozessor, mit $1 \le k \le n$

Bild 6-4: Codierung der Chromosomen

Andere Kombinationen scheiden aufgrund der oben genannten Randbedingungen entweder von vorneherein aus, da durch die evolutionären Operatoren nur letale Individuen erzeugt würden, oder aber die Berechnung ist deutlich aufwendiger. Beispielsweise führt die Abbildung „Prozessor –> Chromosom“, „Priorität –> Gen“ und „Aktivität –> Allel“ dazu, daß eine Rekombination mit Crossover Chromosomen erzeugt wird, auf denen bestimmte Aktivitäten überhaupt nicht mehr vorhanden sind. Der Ausschluß von letalen Individuen, welcher bereits durch die Codierung erreicht werden kann, ist prinzipiell nicht notwendig, erhöht jedoch die Effizienz der Optimierung, da diese Randbedingungen bei der Selektion nicht mehr explizit überprüft werden müssen.

Die Codierung der Prioritäten selbst kann mittels zweier grundsätzlicher Methoden erfolgen: zum einen durch eine Bitcodierung unter Verwendung des in Kapitel 4 vorgestellten Gray-Codes oder zum anderen, indem die ganzzahligen Werte der Prioritäten direkt verwendet und modifiziert werden. Für die erste Methode spricht die direkte Analogie zur Natur, für die zweite Methode allerdings die deutlich einfachere Handhabung, da die Codierungs-/Decodierungsschritte eingespart werden können. Ziel der Codierung muß sein, durch kleine Änderungen (vgl. Punktmutation, Abschnitt 4.4.2.1.2) auch nur kleine Änderungen am Phänotyp zu

bewirken. Dies kann problemlos durch die ganzzahlige Codierung erreicht werden, wenn die Mutationsweite zu „1“ gesetzt wird. Deshalb wurde diese Methode verwendet.

Die Codierung für die Verfahren Simulated Annealing und Threshold Accepting verläuft entsprechend. Die damit ermittelten Systemlösungen stellen eine Untermenge der Population Π dar, und bestehen immer nur aus einem einzigen Individuum.

6.2.4 Der Optimierungsablauf

Der Ablauf der Optimierung ist bei den verwendeten Algorithmen stets iterativ. Nach der Initialisierung des Systems und der Aufstellung einer Anfangskonstellation werden immer die folgenden Schritte durchlaufen: Analyse des Systems, Entscheidung, ob die erzielte Qualität ausreicht, Modifikation des Systems, Analyse, usw. Bild 6-5 zeigt diesen Ablauf und die in jeder Phase verwendeten Methoden in Form eines Flußdiagramms.

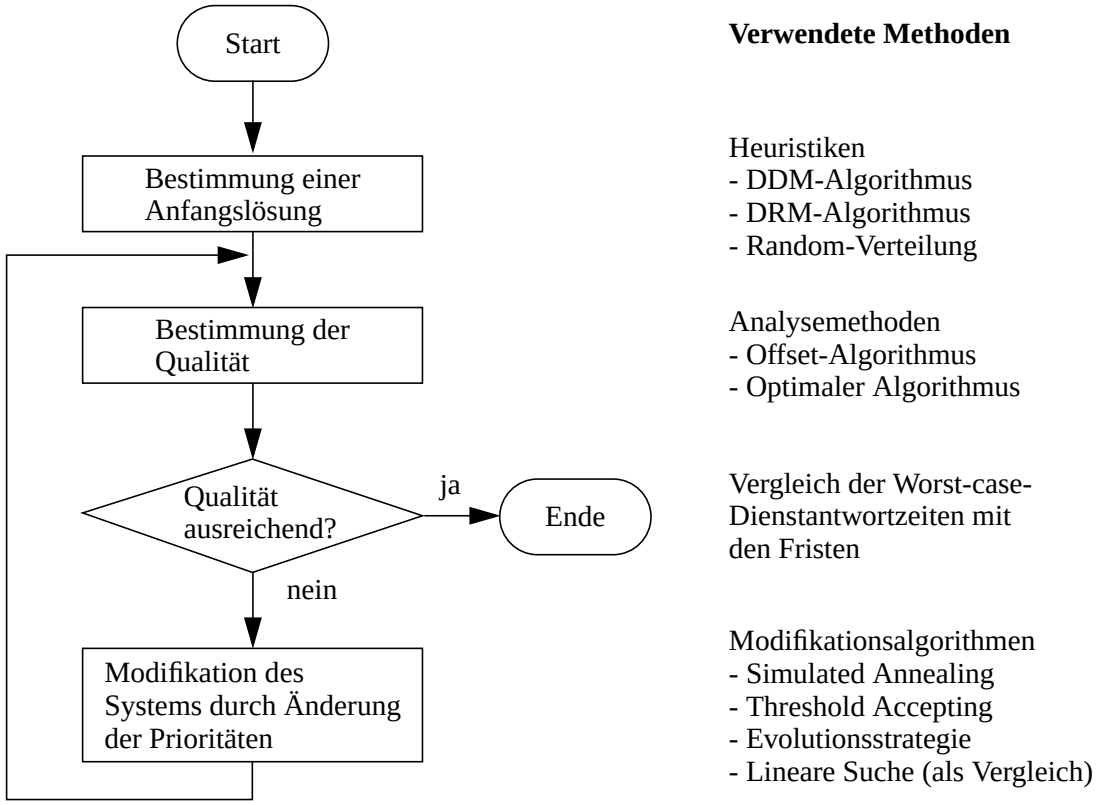


Bild 6-5: Der Optimierungsvorgang und die dafür verwendeten Methoden

Der größte Aufwand des Optimierungsvorgangs steckt in der Qualitätsbestimmung (vgl. Algorithmen aus Kapitel 5). Dagegen ist die Bestimmung eines neuen Individuums durch Prioritätenmodifikation relativ schnell durchgeführt.

6.2.5 Vergleich der Optimierungsverfahren

In den folgenden Unterabschnitten werden verschiedene naturanaloge Optimierungsmethoden dazu verwendet, die Prioritäten in einem Steuergerätenetz so zu verteilen, daß die Echtzeitan-

forderungen eingehalten werden. Die Untersuchung der Leistungsfähigkeit der Algorithmen wurde anhand eines Referenzszenarios durchgeführt, das im folgenden Abschnitt näher vorgestellt wird.

6.2.5.1 Das erweiterte SAE-Referenzszenario

Um Qualität und Leistungsfähigkeit der Optimierungsalgorithmen beurteilen zu können, soll anhand eines größeren, realen Szenarios die Prioritätenvergabe durch diese Algorithmen untersucht werden. Als Grundlage dient dazu das von der SAE aufgestellte Referenzszenario [130] für Class-C Anforderungen (vgl. Abschnitt 2.3.3 und Bild 6-6) mit sieben Steuergeräten, die über einen Bus miteinander verbunden sind.

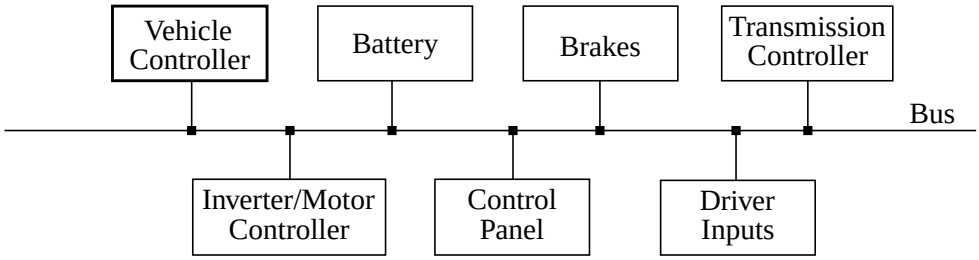


Bild 6-6: SAE-Referenzszenario

Das System beinhaltet ein hervorgehobenes Steuergerät, den „Vehicle Controller“, der sämtliche Information vom Fahrer aufnimmt und an die entsprechenden Komponenten im System weiterleitet. Zur Bewältigung der dafür notwendigen Kommunikationsleistung werden 53 Botschaften über den Bus ausgetauscht. Über die Aktivitäten auf den Steuergeräten werden allerdings keine Aussagen gemacht, so daß hier entsprechende Annahmen getroffen wurden, die dieses System zu einem Steuergerätenetz im Sinne dieser Arbeit erweitern. Jeder Botschaft wird zusätzlich eine Sende- und eine Empfangsaktivität auf dem entsprechenden Steuergerät zugeordnet und somit ein vollständiger Dienst festgelegt. Dazu wurde angenommen, daß alle Prozessoren in den Steuergeräten mit l_k = 400 ticks/ms arbeiten und die Länge C^* der Aktivitäten 50 ticks beträgt. Daraus folgt für die sendenden und empfangenden Aktivitäten auf den Steuergeräten eine Bearbeitungszeit von jeweils C = 0,125 ms. Da für die sporadischen Anforderungen keine Zeiten vorgegeben sind, werden hier jeweils 50 ms als Periode der Dienste angenommen (vgl. [152]). Die Bearbeitungszeiten der Busbotschaften sind durch die Länge der zu übertragenden Nachrichten vorgegeben.[1] Mit dieser Vervollständigung der Spezifikation erlaubt das System, das jetzt aus 53 Diensten mit 159 Aktivitäten gebildet wird (siehe DPA-Matrix in Bild 6-7), insgesamt 1,244 e+176 Möglichkeiten der Prioritätenvergabe. Diese immense Zahl macht es offensichtlich unmöglich, alle Prioritätskombinationen „durchzuprobieren“.

Mit den in Abschnitt 6.1 vorgestellten einfachen Vergabemethoden sind bei diesem System die folgenden Qualitätswerte zu erreichen: Q(DDM) = -0,297 und Q(RDM) = -0,408. Der nach Gl. 6-8 erreichbare theoretische Maximalwert der Qualität liegt bei Q_{max} = 0,0503. Dabei verletzen im ersten Fall (DDM) 6 Dienste ihre Echtzeitanforderungen, während im zweiten Fall (RDM) insgesamt 7 Dienste Antwortzeiten aufweisen, die ihre vorgegebene Frist überschreiten. Bild 6-8 zeigt den Anfang des Qualitätsverlaufs für dieses System (Berechnungsdauer ca.

1. Für die in dieser Arbeit vorgestellte Prioritätenverteilungsmethode ist es im Prinzip unerheblich, wie die genaue Spezifikation des Systems aussieht, da die Methode auf alle spezifizierten Systeme angewendet werden kann.

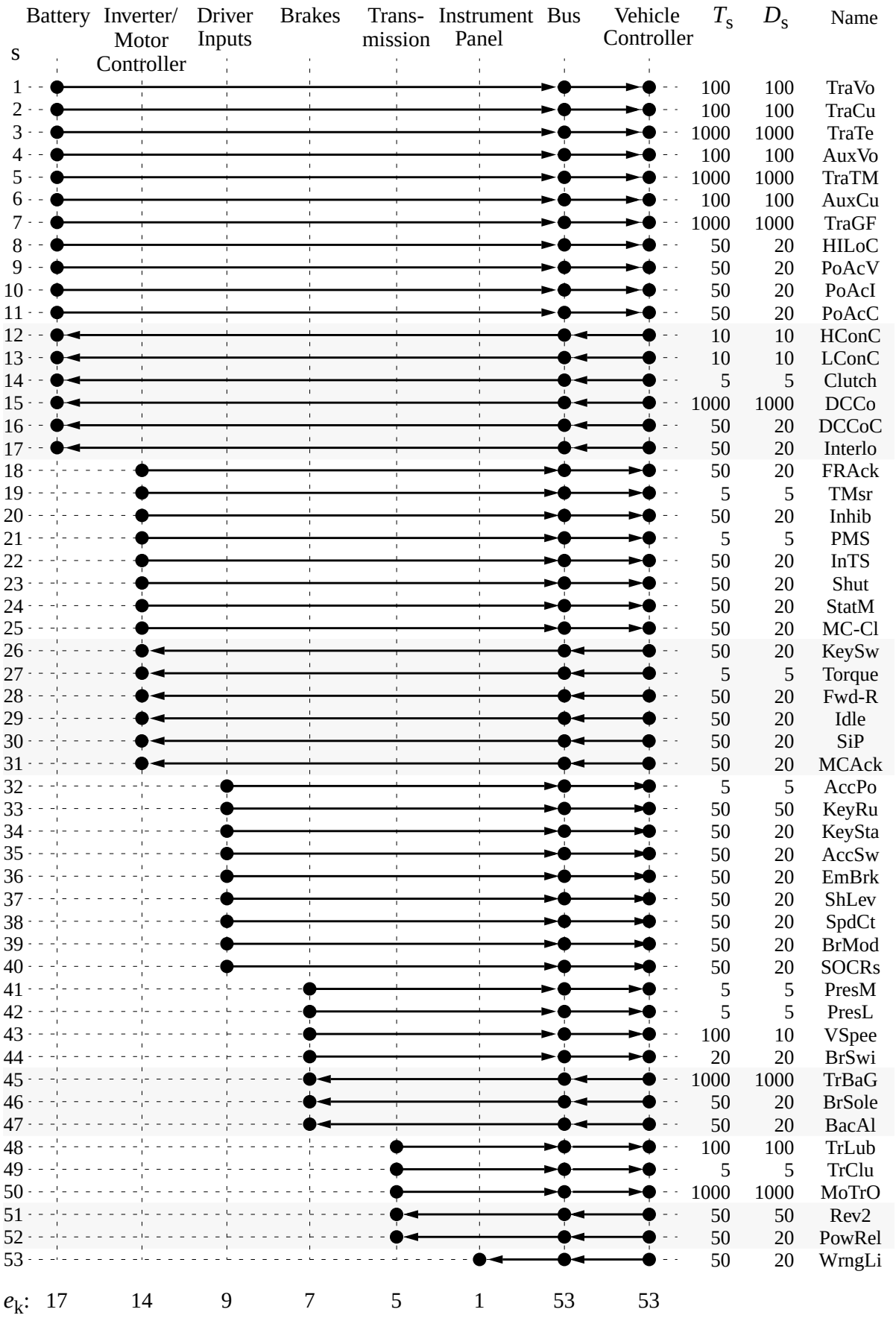


Bild 6-7: DPA-Matrix des erweiterten SAE-Szenarios

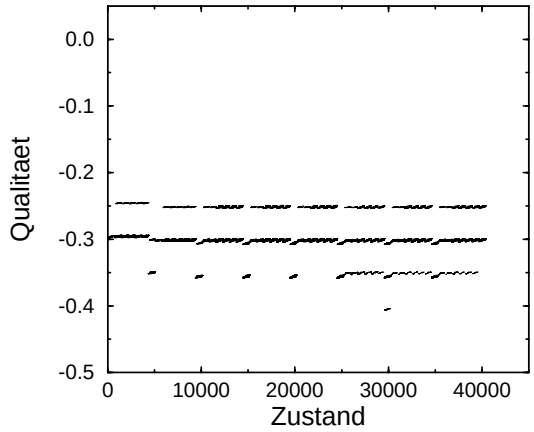


Bild 6-8: Qualitätsverlauf des SAE-Szenarios mit Anfangsverteilung nach DRM

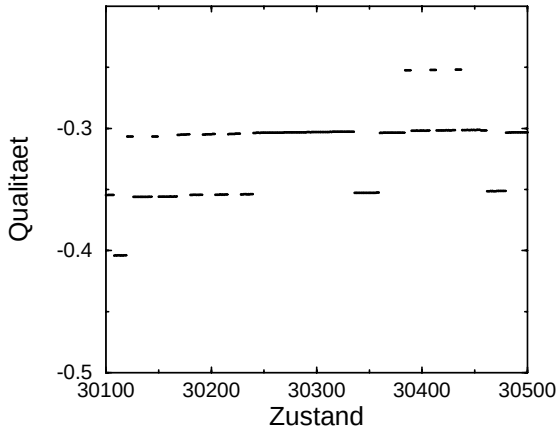


Bild 6-9: Vergrößerter Ausschnitt der Qualitätsfunktion des SAE-Szenarios

7 CPU-Tage!). In der Ausschnittsvergrößerung (Bild 6-9) sind wieder die bereits angesprochenen Qualitäts-Plateaus zu erkennen, die, bedingt durch die Größe des Systems, deutlich ausgedehnter sind als im einführenden Beispiel. Allerdings ist auch erkennbar, daß erwartungsgemäß die durchgeführten Modifikationen, trotz des hohen Rechenaufwands, nie in die Nähe einer funktionierenden Prioritätenverteilung geführt haben.[2]

6.2.5.2 Simulated Annealing

Zur Charakterisierung der untersuchten Vergabestrategien wurde die Systemqualität als Funktion der durchgeführten Iterationen der jeweiligen Optimierer aufgezeichnet. Bild 6-10 zeigt

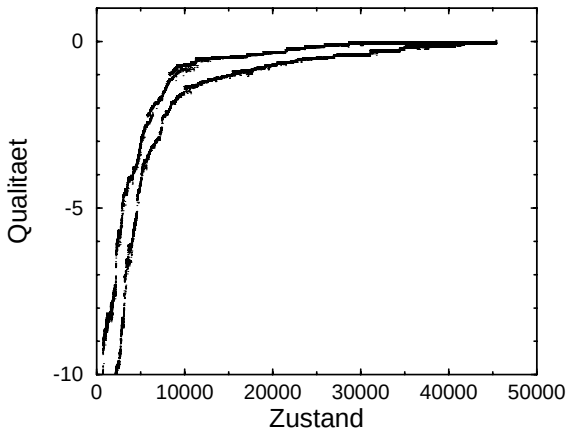


Bild 6-10: Optimierungsverläufe mit Simulated Annealing (SA) und zufälliger Ausgangsverteilung der Prioritäten

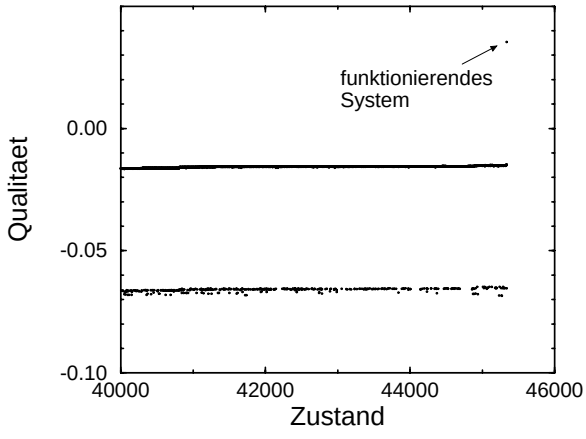


Bild 6-11: Vergrößerter Ausschnitt eines Qualitätsverlaufs

2. Das Ausmaß der Zahl möglicher Zustände kann man sich durch folgenden Vergleich vor Augen führen: Verlängert man den Qualitätsverlauf, wie im Bild dargestellt, so weit, daß das entstehende Diagramm vollständig um die Milchstraße verläuft (Durchmesser: 100.000 Lichtjahre), dann erfaßt man erst 7,5 e+25 Zustände!

den Verlauf zweier Optimierungsläufe, die mit Simulated Annealing (SA) durchgeführt wurden. Das Verfahren ermittelte in beiden gezeigten Beispielen nach ca. 45.000 Iterationen eine gültige Lösung. Die Prioritäten wurden am Anfang per Zufall verteilt. Es läßt sich erkennen, daß das Verfahren im Verlauf der ersten 10.000 Iterationen relativ schnell Zustände findet, die deutlich besser sind als der Anfangszustand. Danach wird der Qualitätsgewinn allerdings geringer. Das liegt daran, daß zum einen bereits relativ gute Werte erreicht wurden und zum anderen nach dieser Zahl an Iterationen die Akzeptanzwahrscheinlichkeit für die Annahme eines schlechteren Wertes schon sehr gering ist. In der Vergrößerung in Bild 6-11 kann man diese zuletzt beschriebenen Effekte daran erkennen, daß die Qualitätswerte zwischen wenigen Qualitätsniveaus hin und her pendeln.

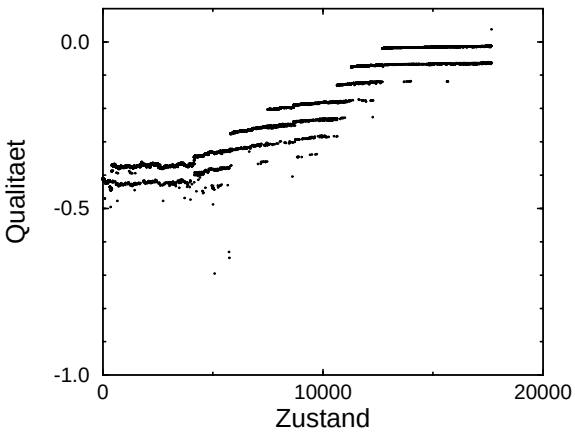


Bild 6-12: Simulated Annealing mit DRM-Anfangsverteilung

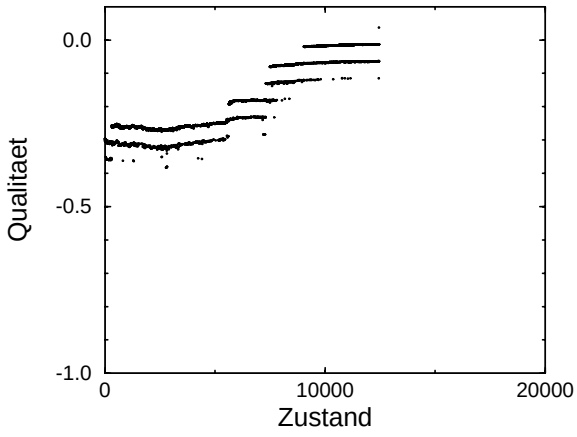


Bild 6-13: Simulated Annealing mit DDM-Anfangsverteilung

Bild 6-12 zeigt den Qualitätsverlauf unter Verwendung einer Anfangsheuristik (DRM). Der Startpunkt für die Optimierung ist um den Faktor 10 besser als ohne Anfangsheuristik. In Bild 6-13 ist der Verlauf der Systemqualität mit einer Anfangsverteilung nach dem DDM-Verfahren zu sehen.

Exemplarisch wurden insgesamt dreißig Vergabeläufe durchgeführt, deren Aufwand in Tabelle 6-4 zusammengefaßt ist. Dabei wurden einerseits zufällige Prioritätsverteilungen angenommen und andererseits Simulated Annealing mit den beiden Heuristiken DRM und DDM kombiniert. Die Tabelle zeigt die minimale, die maximale und die mittlere Zahl der notwendigen Iterationen, bis eine gültige Lösung gefunden wurde. Alle diese Beispiele führen zu sehr ähnlichen Ergebnissen, die sich in der erreichten Qualität nicht wesentlich unterscheiden. Siehe dazu auch Abschnitt 6.2.5.5.

Vergabeläufe mit SA	Zahl der Läufe	min. Iterationen	max. Iterationen	mittlere Iterationenzahl
Zufällige Anfangsverteilung	10	32732	45336	40473
Anfangsverteilung mittels DRM	10	11081	39307	21054
Anfangsverteilung mittels DDM	10	8851	18361	13542

Tabelle 6-4: Anzahl der Iterationen bei Simulated Annealing

Die mittlere Zeit für 1.000 Iterationsschritte beim Simulated Annealing beträgt ca. 130 s, womit die Vergabe der Prioritäten für das SAE-System in rund einer Stunde realisierbar wird. Der weitaus größte Teil dieser Zeit (ca. 97 %) wird dabei allerdings für die Analyse des Systems und somit zur Berechnung der Diensteantwortzeiten verbraucht.[3] Die Implementierung des SA-Algorithmus erfolgt entsprechend Abschnitt 4.4.1.

6.2.5.3 Threshold Accepting

Wird die Prioritätenvergabe mittels Threshold Accepting (TA) durchgeführt, dann zeigt der Verlauf der über den Iterationen erreichten Qualität ein ähnliches Aussehen, wie beim Simulated Annealing.

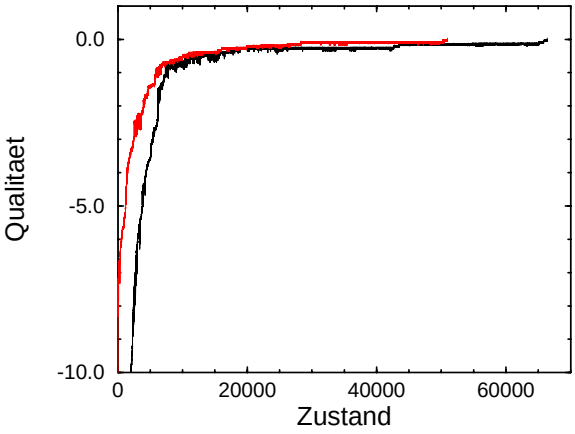


Bild 6-14: Optimierungsverlauf mit Threshold Accepting und zufälliger Ausgangsverteilung der Prioritäten

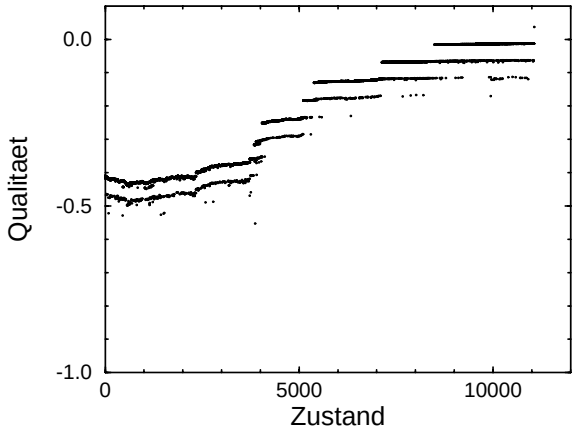


Bild 6-14: Optimierungsverlauf beim TA mit Anfangsheuristik DRM

Der anfängliche Anstieg der Qualität ist allerdings etwas steiler als beim SA. In Tabelle 6-5 ist wiederum der jeweilige Aufwand für die durchgeführten Optimierungsläufe zusammengefaßt.

Vergabeläufe mit TA	Zahl der Läufe	min. Iterationen	max. Iterationen	mittlere Iterationenzahl
Zufällige Anfangsverteilung	10	17997	68764	36404
Anfangsverteilung mittels DRM	10	11063	41045	17622
Anfangsverteilung mittels DDM	10	8159	13621	11287

Tabelle 6-5: Anzahl der Iterationen bei Threshold Accepting

Die mittlere Zeit für 1.000 Iterationsschritte beim Threshold Accepting beträgt ca. 130 s und liegt damit in der gleichen Ordnung wie der Zeitaufwand für SA (Begründung s.o.). Die Implementierung des Algorithmus erfolgte lt. Abschnitt 4.4.1.

3. Diese Zeit sowie alle anderen Zeiten zur Leistungsangabe der Algorithmen wurden über mehrere Läufe hinweg mit dem „time“-Befehl unter UNIX auf einer HP-Workstation mit PA-8000 Prozessor gemessen. Dabei wurde nur der für die tatsächliche Ausführungszeit relevante Teil „user“ berücksichtigt.

6.2.5.4 Evolutionsstrategien

Als dritter Optimierungstyp sind die Evolutionsstrategien untersucht worden. Dabei wurde von der in Abschnitt 4.4.2.3.2 vorgestellten Form etwas abgewichen und eine vereinfachte Version eingesetzt (wohingegen eine vollständige Form zwar implementiert ist, der dafür nötige Rechenaufwand allerdings in keinem Verhältnis zum erreichten Nutzen stand). In den hier vorgestellten Beispielen wurde deshalb auf eine Vererbung (mit entsprechender Mutation) des Strategieparameters Mutationsweite verzichtet. Der Rotationswinkel wurde überhaupt nicht verwendet, da es durch die vorgenommene Codierung in diesen Systemen keine Korrelation zwischen den einzelnen Genen (Prioritäten) eines Chromosoms (Prozessor) gibt.

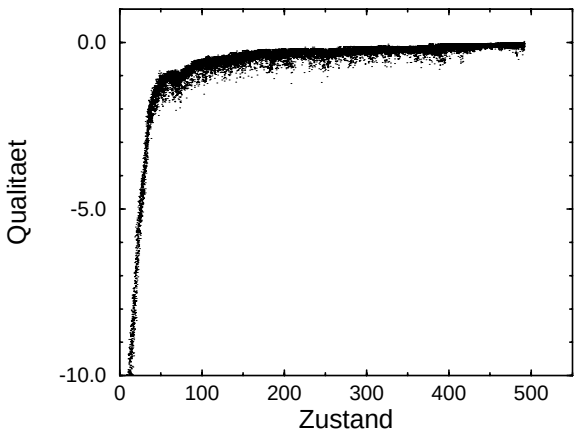


Bild 6-15: Optimierungsverlauf mit Evolutionsstrategien

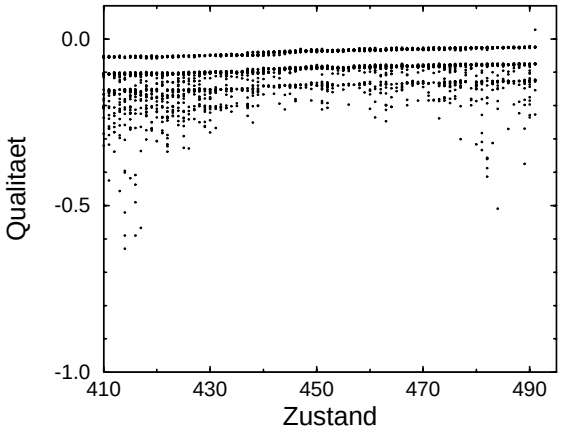


Bild 6-16: Vergrößerter Ausschnitt des Qualitätsverlaufs

In Bild 6-15 ist die Qualitätsentwicklung für eine Population zu sehen, die in jeder Generation zunächst aus 10 Eltern und jeweils 40 Nachkommen (ohne Eltern: Generation Policy). Der vergrößerte Ausschnitt in Bild 6-16 läßt die 40 Individuen mit ihren Qualitäten in einer Generation erkennen.

Tabelle 6-6 faßt die durchgeführten Prioritätenvergabeläufe zusammen und zeigt die Zahl der notwendigen Iterationen, bis eine gültige Lösung gefunden wurde. (Eine Generation wurde mit 40 Iterationsschritten gezählt.) Die mittlere Bearbeitungsdauer für 25 Generationen (entspricht bei 40 Nachkommen 1000 Iterationsschritten) beträgt für die Evolutionsstrategien ca. 250 s.

Vergabeläufe mit ES	Zahl der Läufe	min. Iterationen	max. Iterationen	mittlere Iterationenzahl
Zufällige Anfangsverteilung	10	11960	43840	21520
Anfangsverteilung mittels DRM	10	5800	15920	8820
Anfangsverteilung mittels DDM	10	2880	5720	4420

Tabelle 6-6: Anzahl der Iterationen bei Evolutionsstrategien

6.2.5.5 Vergleich der Algorithmen

Das Hauptkriterium für den Vergleich der vorgestellten Algorithmen ist die Anzahl der Iterationen bis eine funktionierende Lösung gefunden wurde. Durch die Vorgabe einer Qualitätsschwelle, deren Erreichen für die Verfahren ausreicht, ist die tatsächliche Qualität einer ermittelten Lösung weniger entscheidend. Allerdings können, ausgehend von einer gefundenen Lösung, in sehr kurzer Zeit weitere gültige Lösungen bestimmt werden, da die Verfahren auf den Qualitätsniveaus relativ schnell weitere Lösungen finden.

Für einen Vergleich der untersuchten Optimierungsmethoden wurden deren Ergebnisse als Funktion der Anzahl der benötigten Iterationsschritte aufgetragen (siehe Bild 6-17).

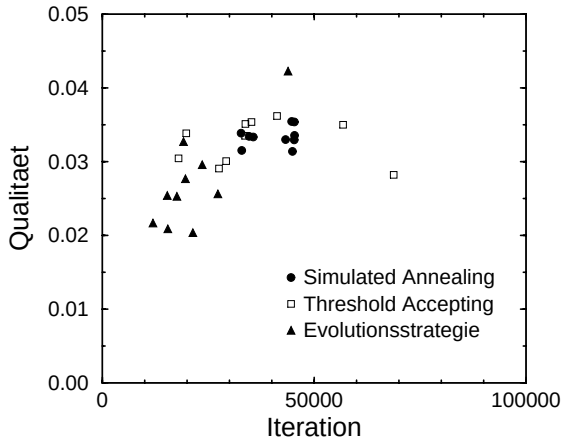


Bild 6-17: Erreichte Qualität nach der Optimierung als Funktion der Iterationen bei zufälliger Anfangsverteilung der Prioritäten

Dabei schnitten die Evolutionsstrategien im Mittel am besten ab. Simulated Annealing zeigt relativ konstante Ergebnisse, während Threshold Accepting breit gestreute Ergebnisse liefert, dabei jedoch auch sehr schnell Lösungen findet. Der Vorsprung der Evolutionsstrategien scheint von der Verwendung der Rekombination mit Crossover herzurühren. Während SA und TA nur „Mutationen“ zulassen, werden durch die Evolutionsstrategien auch Änderungen in der Prioritätenvergabe durch die Crossover erzielt. Allerdings wirken sich die Crossover gegen Ende des Optimierungsverlaufs prinzipiell eher störend aus, da sie die Konvergenz des Verfahrens durch starkes Ändern der zugeordneten Prioritäten wieder zunichte machen können. Dieses Problem umgehen die Evolutionsstrategien durch die Mutation der Strategieparameter.

Die beiden nächsten Bilder (6-18 und 6-19) zeigen die drei Optimierungsmethoden in Verbindung mit einer Anfangsvergabestrategie.

Abschließend kann gesagt werden, daß die Unterschiede zwischen den Verfahren nicht besonders gravierend sind (wie auch aus den prinzipiell ähnlich verlaufenden Qualitätskurven erkennbar ist). Dies resultiert hauptsächlich aus der grundsätzlichen Ähnlichkeit dieser Verfahren. Trotzdem scheinen die Evolutionsstrategien aus den oben genannten Gründen in den gezeigten Beispielen stets etwas schneller zu sein, als die beiden anderen Verfahren.

Einen offensichtlichen Vorteil stellt in jedem Fall die Kombination eines dieser Verfahren mit einer Anfangsheuristik dar. Im vorgestellten Beispiel schnitten dabei die Kombinationen mit dem DDM-Verfahren, die durch dessen Anfangsverteilung bereits eine höhere Startqualität zur Verfügung hatten, besser ab, als die Verbindung DRM + naturanaloger Optimierer.

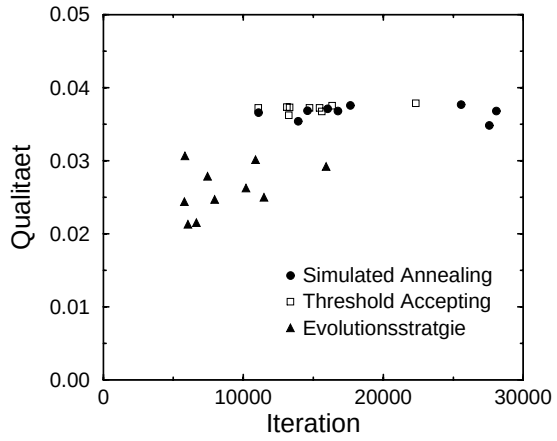


Bild 6-18: Erreichtes Optimierungsziel als Funktion der Iterationen, Anfangsverteilung mittels DRM-Verfahren

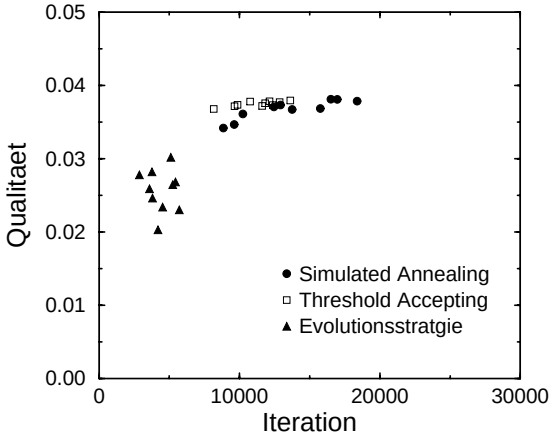


Bild 6-19: Optimierungsziel als Funktion der Iterationen, Anfangsverteilung mittels DDM-Verfahren

Hat man durch das eben beschriebene Verfahren ein Qualitätsplateau mit einem funktionierenden System erreicht, so können (falls vorhanden) schnell weitere Lösungen besserer Qualität gefunden werden. Für diese weitere Suche ist Threshold Accepting ideal, da mit diesem Verfahren durch die Einstellung der Toleranzschwelle lediglich bessere Werte zugelassen werden können und mit einer kleinen Modifikationstiefe explizit die nahe Umgebung abgesucht werden kann.

Insgesamt wurden über 150 verschiedene Läufe mit unterschiedlichen Parametern durchgeführt, die alle zu einer gültigen Lösung führten. Die Spannweite der notwendigen Iterationen, bis diese Lösung gefunden wurde, reichte dabei von 2880 bis zu 99876 Iterationen. Damit kann festgehalten werden, daß die Wahl der Systemparameter zwar die Zahl der notwendigen Iterationen beeinflußt, die Verfahren allerdings relativ robust gegenüber diesen Parametereinstellungen reagieren.

6.3 Diskussion der Methode

Mit Hilfe der vorgestellten Methode der Prioritätenvergabe mittels Optimierungsverfahren ist es möglich, gültige Verteilungen der Prioritäten zu erhalten, die alle Echtzeitanforderungen erfüllen. Dies gelingt in relativ kurzer Zeit im Vergleich zu einer vollständigen Suche im möglichen Lösungsraum der Prioritätenverteilungen eines Steuergerätenetzes. Für reale Systeme kann eine derartige Suche durch die kombinatorische Vielzahl an Möglichkeiten unmöglich sein.

Unbefriedigend bleibt die Tatsache, daß bei der Anwendung der Optimierungsverfahren keine Gewißheit darüber herrscht, ob überhaupt eine Lösung gefunden werden kann. Weiterhin hat man auch keine Gewißheit darüber, wie lange es dauert, bis eine tatsächlich existierende Lösung erkannt wird. In den durchgeführten Untersuchungen konvergierten zwar alle drei Verfahren in sehr kurzer Zeit (innerhalb von Minuten, bzw. wenigen Stunden), allerdings hat man dafür jedoch keine Garantie.

Desweiteren werden die aufgeführten Verfahren durch eine Vielzahl von Parametern bestimmt. Auf eine ausführliche Diskussion der Auswirkung dieser Parameter wurde hier verzichtet, da

deren Einstellung stark von dem zu optimierenden System abhängt und deshalb keine allgemeingültigen Aussagen gemacht werden können. Es zeigte sich außerdem, daß sich die Optimierungsverfahren relativ robust gegenüber Parameteränderungen verhielten. Letztendlich kann somit die Behauptung untermauert werden, daß sich derartige Verfahren sehr gut für die Lösung des gestellten Problems eignen.

Eine gemeinsame Eigenschaft der naturanalogen Optimierungsverfahren ist, daß keine Kenntnis über den Ort des globalen Maximums eines Systems existiert. Deshalb kann mit diesen Methoden eine Aufgabenstellung, wie z.B. „Ermittle das kostengünstigste System!“, nicht beantwortet werden. Hier muß die Aufgabe/Frage immer lauten: „Gibt es ein System, das die Qualitätsanforderung *Q* erfüllt?“. Dies ist für reale, technische Probleme vollkommen ausreichend, da bei entsprechender und vollständiger Planung immer von einer zu erreichenden Sollvorgabe ausgegangen werden kann.

Um das *NP*-vollständige Problem der Prioritätenvergabe lösen zu können, steht hiermit ein nichtdeterministischer Algorithmus zur Vergabe der Prioritäten zur Verfügung. Die gefundene Vergabe kann schließlich über den in Kapitel 5 gefundenen Algorithmus in polynomialer Zeit bewertet werden.

Kapitel 7

Werkzeugunterstützte Planung

Die in den vorhergehenden Kapiteln vorgestellten Methoden zur Analyse und Planung von Steuergerätenetzen auf der Basis von Echtzeitprioritätensystemen wurden im Rahmen dieser Arbeit prototypisch in einer Werkzeugumgebung realisiert. Eine derartige Werkzeugumgebung ermöglicht im realen industriellen Einsatz sowohl die Erstellung als auch die notwendige Überprüfung (und ggf. Modifikation) einer Spezifikation für ein derartiges System und erlaubt damit die frühzeitige Entdeckung von Problemen und Designfehlern. Wurde eine erfolgreiche Planung durchgeführt, dann kann das Funktionieren des Systems in jedem Fall garantiert und funktionierender Code für die Zielplattformen generiert werden.

In diesem Kapitel soll, aufbauend auf dem Planungsablauf der Echtzeiteigenschaften eines Steuergerätenetzes und den daraus resultierenden Anforderungen, ein Überblick über die Architektur und den Aufbau dieser Werkzeugumgebung gegeben werden. Der Schwerpunkt liegt dabei in der Darstellung der Architektur des Analyse- und Optimierungswerkzeugs.

7.1 Anforderungen an die Planungsunterstützung

Die Anforderung an die Planungsunterstützung leiten sich aus dem notwendigen Planungsablauf ab und können in zwei Kategorien eingeteilt werden: Einmal die offensichtlichen Anforderungen bezüglich der Unterstützung des Anwenders bei den einzelnen Schritten der Planung selbst, zum anderen aber auch die softwaretechnischen Anforderungen an die Werkzeuge, wie beispielsweise deren Erweiterbarkeit. Nach der Vorstellung des Planungsablaufes sollen deshalb diese beiden Punkte in den folgenden Abschnitten näher untersucht werden.

7.1.1 Der strukturierte Planungsablauf

Der Planungsablauf erfolgt in wohlstrukturierten Schritten[1]. Bild 7-1 gibt diese Schritte, eingeteilt in die Phasen I-IV, wieder:

- Phase I legt die Systemeigenschaften anhand verschiedener Spezifikationen fest. Hier wird angegeben, welche Hardware-Komponenten zur Verfügung stehen und wie diese miteinander vernetzt werden. Außerdem werden die Dienste mit den notwendigen zeitlichen Angaben spezifiziert.

1. Die Anforderungen, die aus der Zusammenarbeit zwischen Fahrzeughersteller und Zulieferer resultieren, sollen hier nicht näher betrachtet werden (siehe dazu [144]).

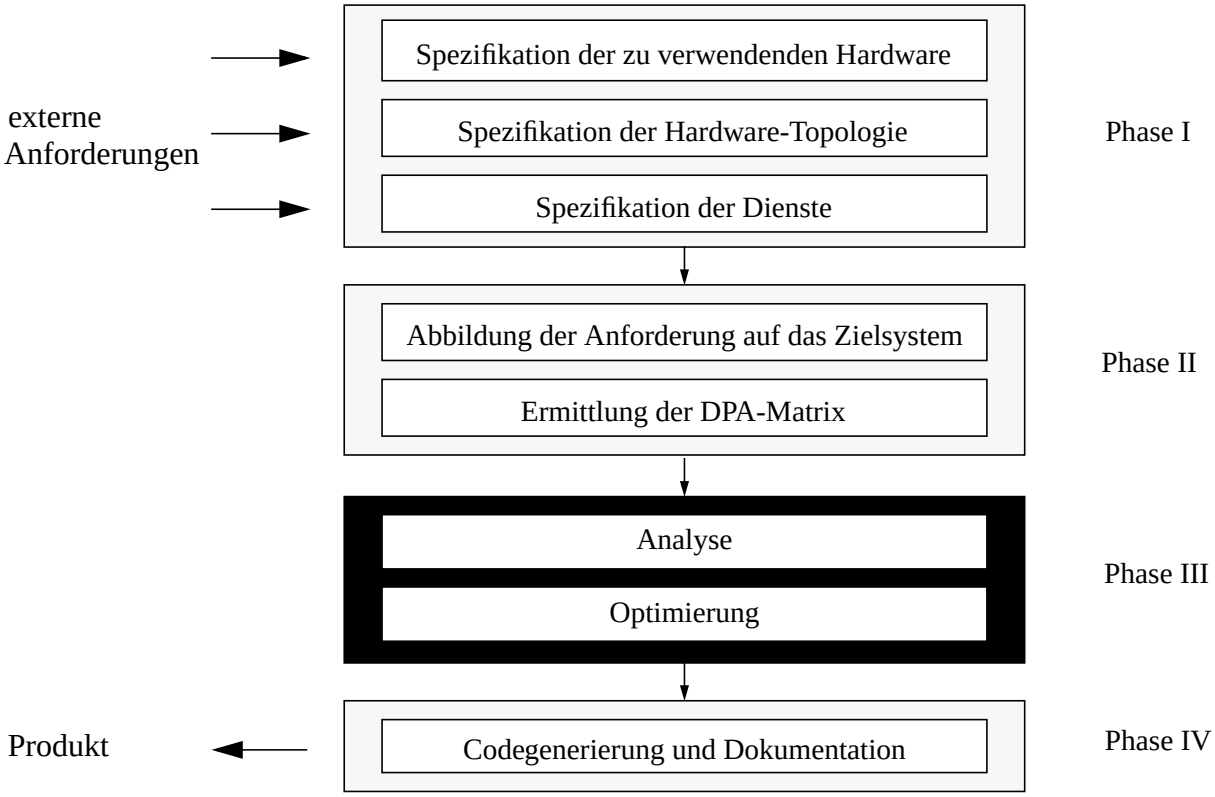


Bild 7-1: Planungsablauf der zeitlichen Anforderungen eines Steuergerätenetzes

- In Phase II findet die Transformation der Dienstespezifikation auf die Hardware-Anforderungen statt. Das Ergebnis ist die in Kapitel 5 vorgestellte DPA-Matrix.
- Phase III erlaubt dann die Analyse der in Phase II gefundenen Konfiguration. Sind alle Randbedingungen erfüllt, dann kann diese Phase verlassen werden. Tauchen jedoch Widersprüche zur gegebenen Spezifikation auf, so muß durch eine geeignete Modifikation der Systemparameter (der Prioritäten) versucht werden, die Randbedingungen einhalten zu können. Diese Modifikation wird durch die Unterphase „Optimierung“ erreicht, mit deren Hilfe die Prioritätenverteilung innerhalb des Systems modifiziert wird. Kann keine Konfiguration gefunden werden, die alle Anforderungen erfüllt, wird dies geeignet gemeldet und der Planungsvorgang abgebrochen.
- Die letzte Phase dient zur Produktion des gewünschten Codes auf den Zielplattformen. Dazu müssen Codegeneratoren zur Verfügung stehen, welche die ermittelte Konfiguration auf die Prozessoren abbilden (i.d.R. durch Code in der Sprache „C“). Außerdem muß eine Dokumentation der gefundenen Konstellation erfolgen, um das Ergebnis nachvollziehbar zu machen.

Die letzte Phase hat aus der Sicht der Korrektheit des Produkts einen zentralen Stellenwert. Wird die ermittelte Konfiguration und Parametrierung automatisch in Zielcode umgesetzt, dann kann, korrektes Verhalten der Generatoren vorausgesetzt, von einem garantiert fehlerfrei arbeitenden System ausgegangen werden. Diese Fehlerfreiheit läßt sich, wie in den vorangegangenen Kapiteln gesehen, auch nachweisen.[2]

2. Es sei an dieser Stelle nochmals darauf hingewiesen, daß Fragen der Ausfallsicherheit von Systemen in vorliegender Arbeit nicht betrachtet werden.

7.1.2 Anforderungen an die Werkzeugumgebung

Die Aufgabe der Werkzeugunterstützung ist es, den Entwickler bei der Erstellung eines funktionierenden Steuergerätenetzes in allen vorgestellten Planungs- und Produktionsphasen zu unterstützen. Nach einer erfolgten funktionalen Spezifikation muß, wie in Kapitel 5 vorgestellt, eine Abbildung dieser Spezifikation auf die Hardware des zu planenden Systems stattfinden. Diese Abbildung wird mit den vorgeschlagenen Methoden analysiert und, bei negativem Analyseergebnis, modifiziert. Werden schließlich alle Randbedingungen erfüllt, indem entweder diese Modifikationen ausreichten oder z.B. sinnvolle Alternativen aufgezeigt wurden, dann wird mittels automatischer Codegenerierung der Code für die unterschiedlichen Zielplattformen erstellt. Dies ist wichtig, da sich typischerweise unterschiedliche Prozessortypen in einem Steuergerätenetz befinden.

Bild 7-2 zeigt den prinzipiellen Aufbau einer integrierten Werkzeugumgebung [68], wobei jede der ermittelten Entwicklungsphasen durch ein eigenes, spezielles Werkzeug realisiert wird.

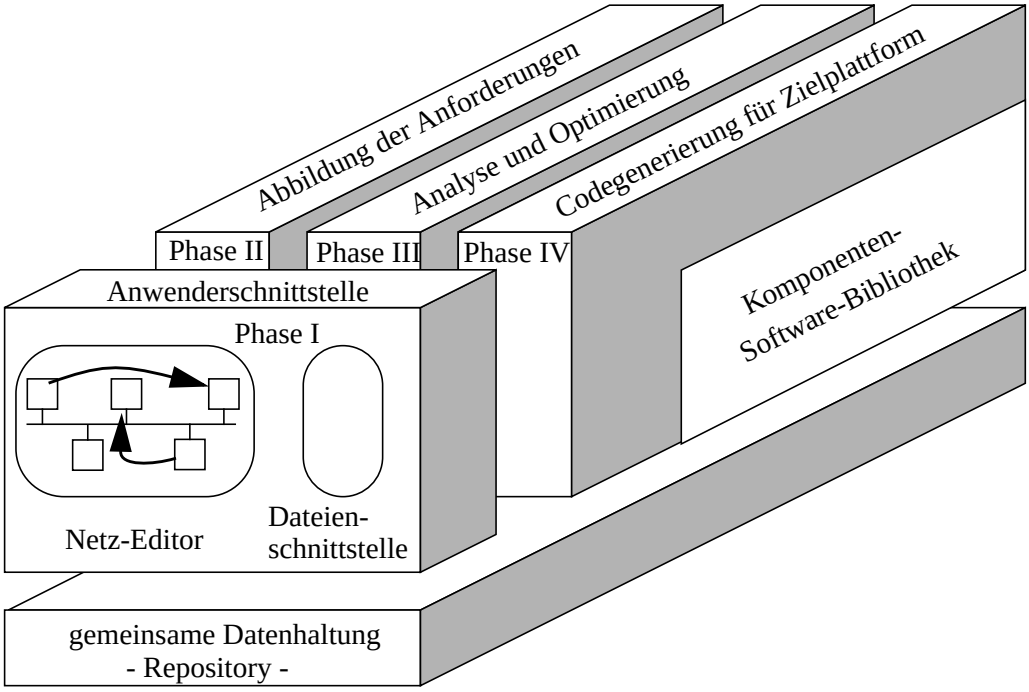


Bild 7-2: Werkzeugumgebung für die Planung von Steuergerätenetzen

Der modulare Aufbau erleichtert es zum einen, Teilwerkzeuge einfach gegen neuere Werkzeuge auszutauschen. Zum anderen besteht ein Vorteil darin, daß die Teilwerkzeuge getrennt voneinander entwickelt werden können. Hinzu kommt, daß mit einer derartigen offenen Architektur auch weitere Planungsaufgaben, die das Steuergerätenetz betreffen, integriert werden könnten. Hierunter fallen z.B. der Applikationsentwurf oder die Kostenermittlung des Systems.

Über eine Anwenderschnittstelle, die sowohl durch einen graphischen Netzeditor als auch eine einfache Dateienschnittstelle gebildet werden kann, wird das zu planende System spezifiziert und in die Werkzeugumgebung eingegeben. Das bedeutet im einzelnen den Entwurf der Netztopologie und die Festlegung der darauf ablaufenden Dienste mittels eines Netzeditors (oder auf Dateibasis mit Hilfe einer geeigneten Beschreibungssprache).

In der nächsten Phase findet die Abbildung der Dienste auf das Zielsystem statt (die Ermittlung der DPA-Matrix), bevor dann in Phase III die Analyse und Optimierung durchlaufen wird. Treten hierbei keine Probleme auf, so kann in Phase IV die Umsetzung der Spezifikation in den Zielcode stattfinden. Dazu benötigt diese dritte Phase zusätzliche Information über die verwendeten Hardware-Plattformen, um geeigneten Code zu generieren. Dies kann beispielsweise mittels einer zusätzlichen Software-Bibliothek realisiert werden.

Alle Teile der Umgebung verwenden außerdem eine gemeinsame Datenbasis. Die Steuerung der einzelnen Komponenten kann entweder zentral über die Anwenderschnittstelle erfolgen oder, deutlich aufwendiger in der Realisierung, indem jedes Werkzeug das ihm logisch folgende Werkzeug aufruft.

Kann in angemessener Zeit durch die Änderung der Prioritäten mittels Optimierung kein funktionierendes System gefunden werden, dann müssen andere Parameter des Systems verändert werden. Dieser Fall kann, bedingt durch die Verwendung der naturanalogen Optimierungsverfahren, durchaus auftreten, da man nie einen direkten Hinweis darauf erhält, ob überhaupt eine Lösung für das vorliegende Planungsproblem existiert. Das Werkzeug muß deshalb in der Lage sein, Hinweise auf Engpässe geben zu können. Der Entwickler hat damit die Möglichkeit, entsprechende Alternativen zu wählen. Das Ziel dieser Alternativen, die Reduktion der Bearbeitungszeit von Aktivitäten, um die Echtzeitanforderungen einhalten zu können, kann auf zwei Wegen erreicht werden:

- Modifikation und Optimierung der verwendeten Software (Steuerungs- und Regelungsalgorithmen, Betriebssystemfunktionen), um eine kürzere Bearbeitungsdauer zu erreichen.
- Modifikation der verwendeten Hardware durch die Wahl schnellerer Prozessoren.

Welche der Alternativen günstiger ist (Aufwand an Entwicklungsarbeit für die Software gegenüber höheren Preisen für schnellere Prozessoren), ist wiederum eine eigene Optimierungsaufgabe.

7.1.3 Anforderungen an die Architektur der Werkzeuge

Die zentrale Anforderung, die ganz allgemein für Software-Systeme gilt, ist die Erweiterbarkeit. Der äußere Qualitätsfaktor[3] „Korrektheit“ wird hier als gegeben vorausgesetzt. Der innere Qualitätsfaktor[4] „Erweiterbarkeit“ bezieht sich in diesem Zusammenhang sowohl auf die Werkzeugumgebung als auch die einzelnen Werkzeuge selbst. Die Erweiterbarkeit des Werkzeugs wurde bereits kurz angesprochen, wird aber hier nicht weiter vertieft. Die Erweiterbarkeit der Einzelwerkzeuge hingegen soll im folgenden Abschnitt berücksichtigt werden. Sie wird wesentlich durch eine einer objektorientierten Analyse folgenden Modellierung und Implementierung erreicht. Exemplarisch wird deshalb die Architektur des Analyse- und Optimierungswerkzeugs dargestellt und anhand spezieller Aspekte des Werkzeugs die Realisierung dieser Anforderung demonstriert.

3. Anforderungen aus der Sicht des Anwenders
4. Anforderungen aus der Sicht des Software-Ingenieurs

7.2 Architektur des Analyse- und Optimierungswerkzeugs

Zunächst soll die Grundarchitektur des Analyse- und Optimierungswerkzeugs vorgestellt werden. Bild 7-3 zeigt das entsprechende, in seiner eigentlichen Komplexität etwas reduzierte, Klassendiagramm[5]. Die hier aufgeführten zentralen Klassen haben die folgenden Aufgaben:

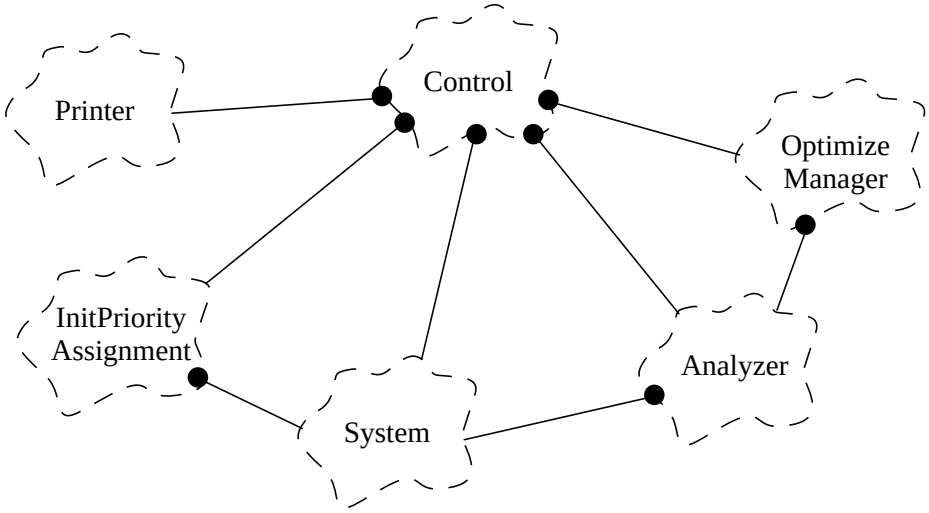


Bild 7-3: Grundarchitektur des Analyse- und Optimierungswerkzeugs

- `Control` ist die steuernde Instanz innerhalb des Werkzeugs. Sie hat Zugang zu allen anderen Hauptklassen und ist für den Ablauf der Aktionen innerhalb des Werkzeugs zuständig (zum Ablauf vgl. Bild 6-5).
- In der Klasse `System` wird die lokale Datenhaltung des Werkzeugs realisiert.
- `InitPriorityAssignment` sorgt für die Herstellung einer Ausgangslösung für das System durch Bereitstellen verschiedener Zuteilungsstrategien.
- Der `OptimizeManager` ist die kontrollierende Instanz für den Optimierungsteil. Er stellt eine virtuelle Methode `Run()` zur Verfügung, die von Control aus aufgerufen wird.
- Die Klasse `Analyzer` dient zur Realisierung der Systemanalyse. Ihre Methode `AnalyzeSystem` kann ebenfalls von `Control` aus aufgerufen werden.
- Mit der `Printer`-Klasse können die Ergebnisse von Analyse und Optimierung ausgegeben und weiteren Werkzeugen zur Verfügung gestellt werden

In den nachfolgenden Abschnitten werden einige dieser Grundklassen kurz vorgestellt.

7.2.1 Datenhaltung für das Werkzeug

Für die prototypische Realisierung des Werkzeugs wurde eine eigene, lokale Datenhaltung realisiert. Die Schnittstelle zu den Daten wird durch die Klasse `System` gebildet. Durch sie werden die Strukturen einer objektorientierten Datenbank gekapselt („Repository“). Bei den Objekte, die in diesem Repository abgelegt sind, handelt es sich z.B. um die Aktivitäten, die Prozessoren und die Dienste eines Systems. Die Modellierung und Implementierung dieser

5. Für die Darstellung wurde die Notation nach Booch verwendet. Siehe auch Anhang A4.

Objekte resultiert wiederum aus dem objektorientierten Vorgehen, welches es erlaubt, die realen Verhältnisse ohne große Modifikation direkt in Software abzubilden.

7.2.2 Bestimmung einer Anfangslösung

Die abstrakte Klasse `InitPriorityAssignment` ist für die Bereitstellung einer Anfangslösung der Prioritätenvergabe verantwortlich. Sie stellt dazu die virtuelle Methode `AssignInitialPriorities` zur Verfügung, die von den abgeleiteten Klassen implementiert werden muß und dort die jeweilige Vergabe ermöglicht. Bild 7-4 gibt einen Überblick über die realisierten Strategien.

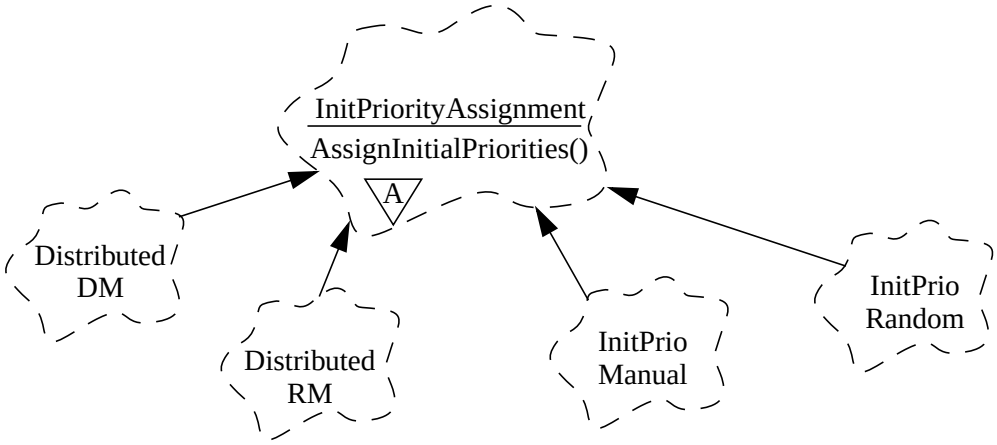


Bild 7-4: Klassendiagramm der Prioritätenvergabestrategien

Die Strategien Distributed-Deadline-Monotonic (DDM) und Distributed-Rate-Monotonic (DRM) wurden bereits in Kapitel 6 ausführlich vorgestellt (siehe Abschnitt 6.1). Mit der Klasse `InitPrioManual` wird eine einfache Strategie realisiert, welche die Prioritäten aus der Beschreibungsdatei ohne Modifikation übernimmt. Die Klasse `InitPrioRandom` ermöglicht es schließlich, Prioritäten zufällig zu vergeben.

7.2.3 Systemanalyse

Die Systemanalyse hat die Aufgabe, die Antwortzeiten der Aktivitäten (und davon abgeleitet der Dienste) im System zu bestimmen. Dazu existiert zur Basisklasse `Analyzer` eine davon abgeleitete Klasse `ResponseTimeCalculator`, die diese Aufgabe übernimmt (Bild 7-5).

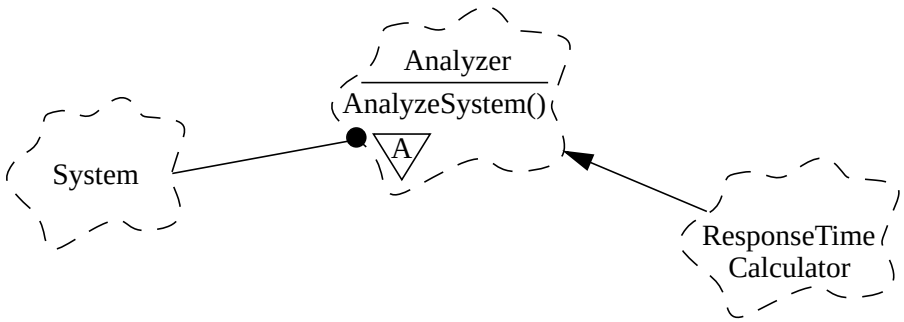


Bild 7-5: Beziehungen der Analyzer-Klasse

In dieser Klasse ist der in Kapitel 5 vorgestellte Offset-Algorithmus umgesetzt. Sollten effizientere Algorithmen zur Bestimmung der Antwortzeit gefunden werden, so können sie problemlos in das Werkzeug integriert werden, indem eine weitere Klasse von Analyzer abgeleitet wird, die diesen neuen Algorithmus enthält. Dabei bleibt der restliche Code des Werkzeugs, dank der objektorientierten Implementierung, von dieser Änderung unberührt.

Die Methode `AnalyzeSystem()` hat zur Erfüllung ihrer Aufgabe Zugriff auf die Daten der Repositoryklasse `System`.

7.2.4 Der Optimierer

Mit Hilfe des Optimierers soll eine Verteilung der Prioritäten gefunden werden, die alle zeitlichen Anforderungen an das System erfüllt. Bild 7-6 zeigt die an dieser Aufgabe beteiligten Klassen und ihre Verbindungen zueinander.

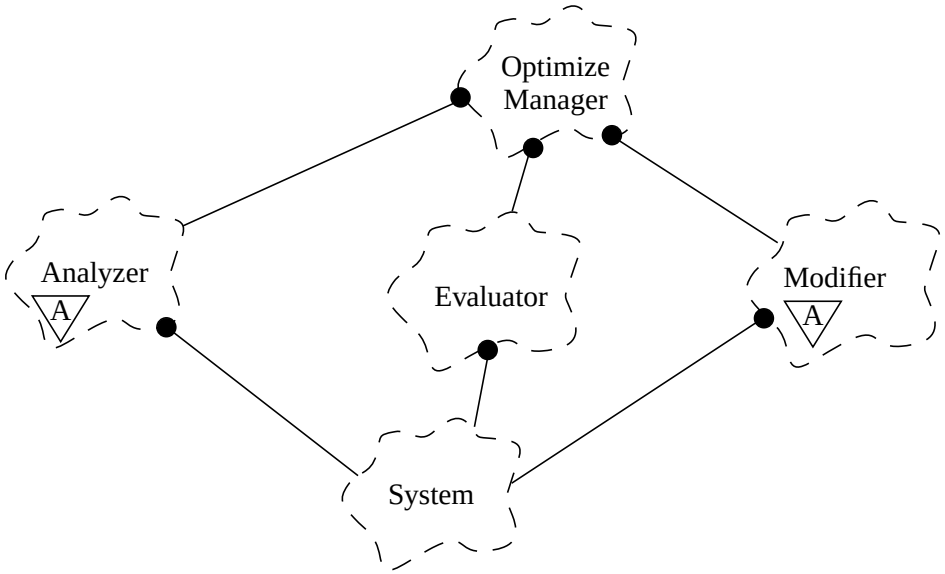


Bild 7-6: Klassendiagramm des Optimierermoduls

Gesteuert wird der Optimierer durch die Klasse `OptimizeManager`. Sie hat Zugriff auf die drei weiteren Klassen, die bei der Optimierung involviert sind. Die Klasse `Analyzer` taucht hier nochmals im Diagramm auf, da sie bei den einzelnen Iterationen des Optimierungsvorgangs wieder zur Berechnung der Antwortzeiten verwendet wird. Die Klasse `Evaluator` stellt eine Methode zur Verfügung, mit der abgeprüft wird, ob die gestellten Randbedingungen an das System erfüllt sind. Ein weiteres Abbruchkriterium, das in dieser Klasse realisiert wurde, ist die Überschreitung einer vorgebbaren, maximalen Zahl an Iterationen, die der Optimierer durchlaufen darf. Die abstrakte `Modifier`-Klasse schließlich gibt die Schnittstelle für unterschiedliche Modifikationsverfahren vor (s.u.).

Die Klassen `Analyzer`, `Evaluator` und `Modifier` stellen, dank der objektorientierten Modellierung und Implementierung, wiederum ein genaues Abbild des vorgestellten Optimierungsalgorithmus dar.

Wie in den vorangegangenen Kapiteln zu sehen war, ist die geeignete Modifikation, neben der Analyse eines Systems, der zentrale Punkt auf der Suche nach einer gültigen Prioritätenvertei-

lung. Deshalb wurden mehrere Modifikationsklassen entworfen, die von ihrer Basisklasse `Modifier` die virtuelle Methode `ModifySystem()` erben.[6] Entsprechend ihrer Ausprägung werden daraufhin die Prioritäten geändert.

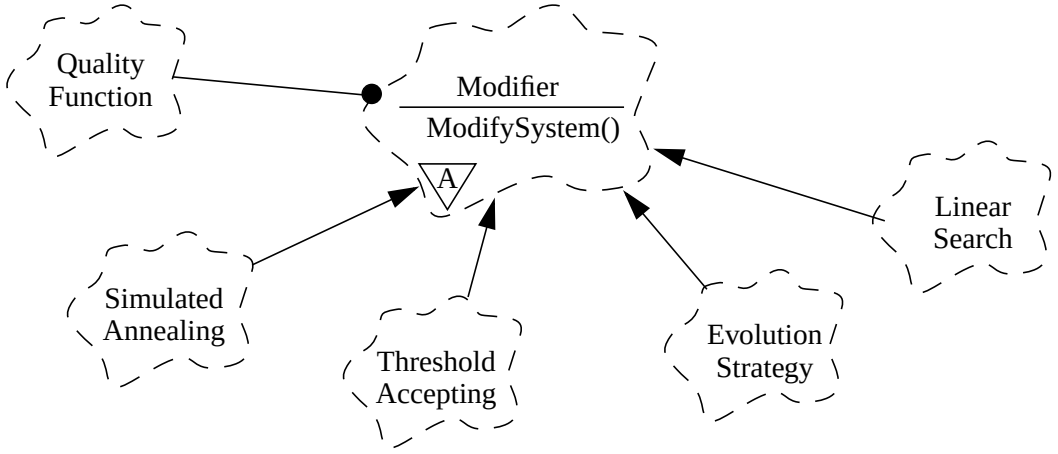


Bild 7-7: Klassenhierarchie der Modifier-Klasse

Die `Modifier`-Klasse hat außerdem Zugriff auf den Algorithmus zur Berechnung der Systemqualität nach Kapitel 6. Dieser ist in der Klasse `QualityFunction` realisiert. Ein vereinfachtes Interaktionsdiagramm soll das Zusammenspiel der beteiligten Objekte veranschaulichen. Die instanziierten Klassen (Objekte) erhalten zur Kennzeichnung den Klassennamen und das Präfix „the“.

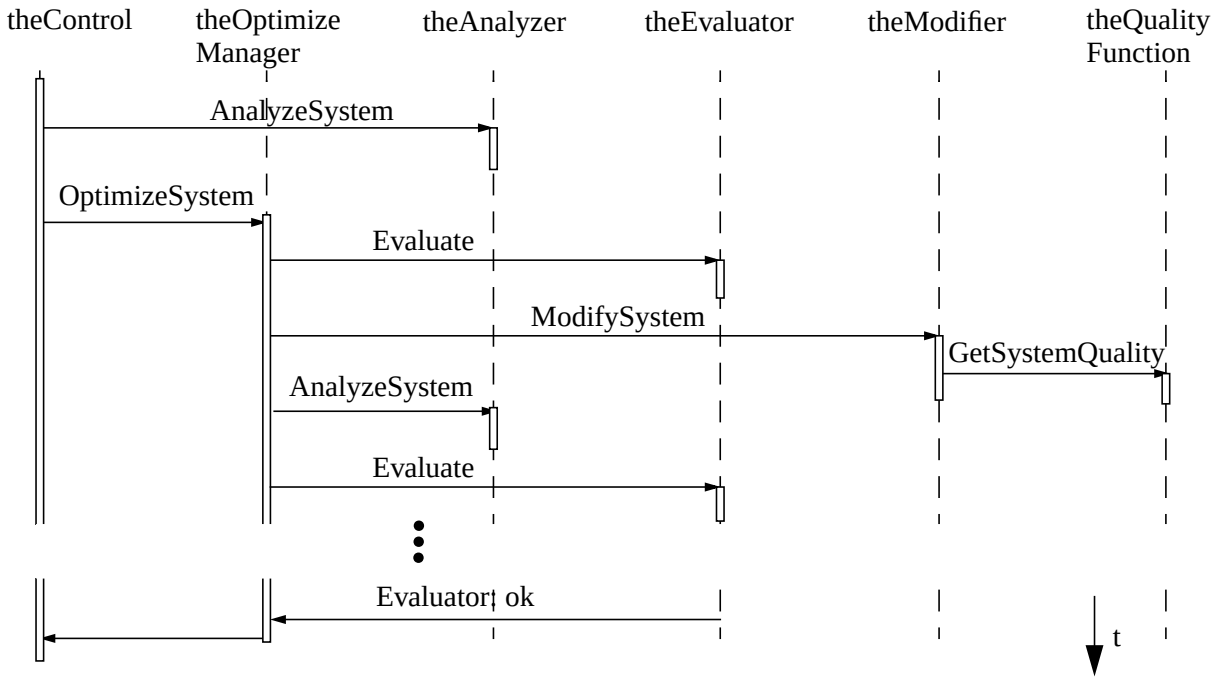


Bild 7-8: Interaktionsdiagramm des Optimierers

6. Dieser Entwurf kann in den Grundzügen mit dem „Strategy“-Design Pattern aus [41] verglichen werden. Dessen Ziel ist die Kapselung von verschiedenen Algorithmen, um sie gegeneinander austauschbar zu machen.

7.2.5 Allgemeine Implementierungsaspekte und Einsatz des Werkzeugs

Das in den vorangegangenen Abschnitten vorgestellte Analyse- und Planungswerkzeug wurde nach objektorientierten Gesichtspunkten analysiert, modelliert und implementiert. Dies erlaubt die eingangs geforderte Erweiterbarkeit, insbesondere im Bezug auf die Analyse und die Optimierung. Von den entsprechenden abstrakten Basisklassen (`Analyzer`, `Modifier`) können problemlos weitere Klassen abgeleitet und mit der entsprechenden Funktionalität versehen werden.

Ein weiterer Vorteil der objektorientierten Vorgehensweise ist die damit verbundene direkte Abbildung des Problembereichs (Werkzeug zur Analyse und Optimierung mit entsprechenden Algorithmen) in eine Programmiersprache, wodurch das Verständnis für den erzeugten Code signifikant erhöht wird.

Die Implementierung erfolgte plattformunabhängig in der Sprache C++ und umfaßt rund 85 Klassen bei ca. 15.000 Zeilen Code. Zur Erstellung des Repositorys wurden zusätzlich die Container-Klassen einer kommerziellen Klassenbibliothek verwendet. Der Parser zum Einlesen der Spezifikation in das Repository des Werkzeugs ist mit Hilfe der Compiler-Tools LEX (für die lexikalische Analyse eines Datenstroms) und YACC (für die syntaktische Analyse der von LEX gelieferten Token) realisiert worden.

Als Ausgabe und Ergebnis dieses Werkzeugs erhält der Anwender Auskunft über alle wichtigen Systemeigenschaften. Dies sind im einzelnen:

- die Antwortzeiten jeder einzelnen Aktivität im System (aufgeteilt in Bearbeitungs-, Blockierungs- und Interferenzzeiten),
- die Antwortzeiten aller Dienste im System,
- das Verhältnis zwischen Antwortzeit und Frist der Dienste,
- die Auslastung der einzelnen Prozessoren im System,
- die Prioritätenverteilung der Aktivitäten, welche die höchste Qualität ergeben hat und
- eine Auflistung der Dienste, die ihre Frist nicht einhalten konnten.

Anhand dieser Information erhält der Entwickler ein detailliertes Bild über das Verhalten des spezifizierten Systems, das ihm auch Aufschluß darüber gibt, wo Modifikationen der Hardware sinnvoll sind, falls die Optimierer keine gültige Lösung finden.

Um den Einsatz des Werkzeugs zu demonstrieren, soll die Vorgehensweise für die Optimierung des SAE-Referenzszenarios aus Kapitel 6 dargestellt werden. Alle dort präsentierten Ergebnisse wurden mit diesem Werkzeug ermittelt.

Es müssen dazu die in Abschnitt 7.1.1 vorgestellten Phasen durchlaufen werden. Der aufwendigste Teil besteht darin, die externen Anforderungen in das System einzugeben. Dies kann über einen ASCII-Editor in einem vorgegebenen Datenformat durchgeführt werden. Alternativ dazu besteht die Möglichkeit, einen Netzeditor zu verwenden, der aus der graphischen Darstellung des Netzes ebenfalls die ASCII-Beschreibung generieren kann. Bild 7-9 zeigt einen Ausschnitt aus der Spezifikation des SAE-Systems. Dabei ist zu erkennen, daß die Spezifikationsdatei aus zwei Abschnitten besteht. Im ersten Abschnitt werden die Analyse- und Optimierungsalgorithmen (in diesem Fall Simulated Annealing) angegeben. Hier können auch die Parameter der einzelnen Optimierungsalgorithmen festgelegt werden. Der zweite

Abschnitt umfaßt die Systembeschreibung, die sich wiederum in drei Teile untergliedert. Im ersten Teil erfolgt die Beschreibung der Prozessoren des Systems: `TransRate` oder `ClockSpeed` geben die Bearbeitungsgeschwindigkeit an, mit `Scheduler` kann zwischen up und nup gewählt werden. Im zweiten Teil der Systembeschreibung werden die Aktivitäten (`Task`) spezifiziert. Für sie muß die auf ticks normierte Bearbeitungszeit sowie der Prozessor angegeben werden, auf dem sie ablaufen. Optional können auch Prioritäten manuell vergeben werden. Schließlich folgen die Dienste (`Service`) des Systems. Sie sind gekennzeichnet durch die Periode (`Period`), die Frist (`Deadline`) und die Folge von Aktivitäten (`TaskSequence`), die diesen Dienst repräsentieren. Der Informationsgehalt, der in dieses Werkzeug eingegeben wird, entspricht somit der Information der DPA-Matrix. Die Spezifikation des SAE-Systems umfaßt in der gezeigten, dreispaltigen Form insgesamt 6 Seiten.

```
Control  SAE-Test-System
{

InitPriorityAssignment = Ran-
dom;

Operation = Optimize;

Analyzer = CalcResponseTime;

OptimizeManager Default
{

Evaluator Default
{
   MaxIteration = 200000;
}

Modifier SimulatedAnnealing
{
   StartStepWidth = 8;
   StepWidthCtrl = 70;
   AccProbabCtrl = 0.08;
   RestoreFromFile = 0;
}

QualityFunction
NormWeightedLaxity
{
   MissedFactor1 = 1.2;
}

}

ServicePrinter
{
  ServiceList = MissedDead-
line;
// ServiceList = All;
}

}  // end of ctrl description
```

```
System  SAE
{

////////////////////////////
// PROCESSORS
////////////////////////////

Proc CANBus
{
   TransRate = 210;
   Scheduler NonPreemptive
}

Proc VehicleCtrl
{
   ClockSpeed = 400;
   Scheduler Preemptive;
}

Proc Battery
{
   ClockSpeed = 400;
   Scheduler Preemptive;
}

...

////////////////////////////
// TASKS/ACTIVITIES
////////////////////////////

Task  TracBattVoltageMsg1
{
  Ticks = 55; // 47 + 8 bit
  Location = CANBus;
}

Task  TracBattCurrentMsg2
{
   Ticks = 55;  // 47 + 8 bit
   Location = CANBus;
}
```

```
Task   ReceiveBattMsg2
{
   Ticks = 50;
   Location = VehicleCtrl;
}


...


////////////////////////////
// SERVICES
////////////////////////////

Service   EmergencyBrake
{
   Period = 50; // sporadic
   Deadline = 20;
   TaskSequence (
      SendDriverMsg19,
      EmergencyBrakeMsg19,
      ReceiveDriverMsg19
   );
}

Service   SpeedCtrl
{
   Period = 50; // sporadic
   Deadline = 20;
   TaskSequence (
      SendDriverMsg22,
      SpeedCtrlMsg22,
      ReceiveDriverMsg22
   );
}


...

} //end of system description
```

Bild 7-9: Ausschnitt aus der Spezifikation des SAE-Beispielszenarios

Jetzt kann ein erster Analyselauf erfolgen, der die im vorangegangenen Abschnitt genannten Ergebnisse liefert. Werden Fristen verletzt, so muß mittels Optimierungsalgorithmen eine neue Prioritätenverteilung gesucht werden. Diese Verteilung wird am Ende der Optimierung ebenfalls ausgegeben (oder die beste gefundene, falls die Zahl der erlaubten Iterationen als Abbruchkriterium festgesetzt wurde). Konvergieren die Optimierer nicht in überschaubarer Zeit in der gezeigten Weise, dann kann durch Änderung ihrer Parameter versucht werden, daß dieser Vorgang beschleunigt wird.

7.3 Bewertung des Werkzeugs

Dem Entwickler von Steuergerätenetzen steht mit der Möglichkeit der Analyse und Optimierung verteilter Echtzeitprioritätensysteme ein wertvolles Werkzeug zur Erstellung derartiger Systeme zur Verfügung. Werden bei der Analyse eines spezifizierten Systems einzelne Verletzungen der Randbedingungen erkannt, so kann eine Parametermodifikation, d.h. die Änderung der Prioritäten, mittels Optimierungsverfahren durchgeführt werden. (Dies führt allerdings nur zum Erfolg, sofern nicht alle Dienste ihre Fristen verletzen.) Tritt der bereits angesprochene Fall ein, daß bei Diensten das Antwortzeit/Frist-Verhältnis trotz Optimierung immer noch größer als 1 ist, muß aus den ermittelten Werten ein Hinweis darauf gewonnen werden, an welcher Stelle das System mit dem geringsten Kostenaufwand modifiziert werden kann. Diese Aufgabe ist nicht trivial, da lokale Änderungen, wie in Kapitel 6 gesehen, Auswirkungen auf das Verhalten des gesamten Systems haben. Folgende Möglichkeiten stehen zur Auswahl:

- *Schnellere Prozessoren*. Potentielle Stellen für Änderungen sind dort zu suchen, wo die Interferenz auf die Aktivitäten des betrachteten Dienstes durch einzelne Aktivitäten anderer Dienste sehr hoch wird. Hier kann eine Beschleunigung der Prozessorgeschwindigkeit nutzbringend sein, weil dadurch diese Interferenz und die eigene Bearbeitungsdauer vermindert wird.
- *Effizientere Software*. Der scheinbar einfachste Weg ist die Beschleunigung der implementierten Software. Dies kann z.B. durch manuelle Code-Optimierung geschehen, da Hochsprachencompiler die zur Verfügung stehende Hardware nicht immer optimal ausnutzen. Dieses Vorgehen erzeugt jedoch Kosten durch den nötigen Zeitaufwand.
- Andere *Scheduling-Strategie*. Aufwendiger ist die Vermeidung von Blockierungen, indem die Scheduling-Strategie verändert und keine nichtunterbrechenden Systeme zugelassen werden. Dies kann natürlich nur zum Erfolg führen, sofern eine derartige Alternative überhaupt besteht. Ist das verwendete Kommunikationsmedium ein prioritätsgesteuertes Bussystem, wie in Abschnitt 2.3.1 vorgestellt, hat man keine Alternative und muß anders modifizieren. Außerdem stehen nicht immer beide Betriebssystemvarianten für die Steuergeräte zur Verfügung.

Ist eine Modifikation der Hardware (schnellerer Prozessor) oder der Software (kürzere Bearbeitungszeit durch schneller bearbeitbaren Code, anderes Scheduling-Verfahren) durchgeführt worden, so wird das Werkzeug wieder gestartet, um die Auswirkungen der Modifikation zu beurteilen. Man erreicht auf diese Weise ein iteratives Vorgehensmodell.

Die Integration dieses Werkzeugs konnte soweit geführt werden, daß die Topologie eines Netzes über einen (plattformunabhängigen) graphischen Netzeditor festgelegt wird. Die Spezifikation der Diensteeigenschaften erfolgt über das vorgestellte formale Beschreibungsformat in Form einer ASCII-Datei. Anschließend an die Analyse- und Optimierungsphase findet die automatische Codegenerierung statt, wie sie in der Darstellung der Anforderungen vorgestellt wurde. Sie ist ebenfalls in einem separaten Werkzeug realisiert, welches die Ergebnisse der Analyse- und Optimierungsphase in „C“- und Assemblercode für die gewünschten Zielplattformen umsetzt. Die gemeinsame Datenbasis wird in diesem Prototyp außerdem durch den Austausch von ASCII-Dateien realisiert. Ergänzend sei erwähnt, daß aus der vollständigen Spezifikation eines derartigen Systems mit diesem Werkzeug automatisch ein Simulationsmodell generiert werden kann, das mittels ereignisgesteuerter, zeitdiskreter Simulation ebenfalls erste Aufschlüsse über das Zeitverhalten des Systems liefert. Die Simulationsergebnisse alleine reichen allerdings, aufgrund ihres statistischen Charakters, nicht für die Funktionsgarantie aus.

Kapitel 8

Zusammenfassung und Ausblick

Die Auswirkung der Prioritätenvergabe in prioritätsbasierten Steuergerätenetzen bildet den zentralen Aspekt der vorliegenden Arbeit. Insbesondere, wenn Dienste mit harten Echtzeitanforderungen berücksichtigt werden müssen, ist eine analytisch begründete Vergabe der Prioritäten (oder Identifier auf den Bussystemen) die einzige Möglichkeit, das korrekte Funktionieren des Systems zu garantieren. Diese Garantie kann durch die in dieser Arbeit entwickelte Methode erreicht werden. Mit ihrer Hilfe ist es dem Planer und Entwickler von derartigen verteilten Echtzeitprioritätensystemen möglich, eine Prioritätsverteilung zu finden, die dazu führt, daß die Antwortzeiten, der im System vorhandenen Echtzeitdienste, ihre vorgegebenen Fristen nicht überschreiten.

Die vorgestellte Methode basiert auf drei grundlegenden, ausführlich diskutierten, Teilphasen:

- Einer Modellierungsphase, in der das zu planende System strukturiert wird und außerdem eine Dekomposition in die wesentlichen Bestandteile stattfindet.
- Einer analytischen Phase, mit der die Einhaltung der Echtzeitfähigkeit eines spezifizierten Systems nachgerechnet werden kann.
- Einer Phase, die mittels Optimierungsverfahren Lösungsvorschläge für neue Systemkonfigurationen erbringt, sofern die Analyse die Echtzeitfähigkeit nicht bestätigen konnte.

Um eine Analyse der Echtzeitfähigkeit möglich zu machen, wurde ein Modellierungsvorschlag für verteilte Prioritätensysteme erarbeitet. Grundlage für diese Modellierung war die Erkenntnis, daß sich sowohl Tasks auf den betrachteten Steuergeräten als auch Botschaften, die über ein prioritätengesteuertes Bussystem ausgetauscht werden, analytisch betrachtet, genau gleich verhalten und deshalb allgemein als Aktivitäten bezeichnet werden können. Dieser Schritt war notwendig, da die Kommunikationsdienste nicht isoliert von den Anwendungen im System betrachtet werden können, sondern ein alle Komponenten umfassendes Modell erstellt werden muß. Mit der aus dieser Modellierung resultierenden Dienste-Prozessoren-Aktivitäten-Matrix kann die für die Analyse notwendige Information übersichtlich dargestellt werden.

Basis der Analyse ist ein Algorithmus zur Ermittlung der Worst-case-Antwortzeiten von Aktivitäten auf einem Prozessor. Aufbauend auf diesem Algorithmus wurde ein Verfahren vorgestellt, welches es prinzipiell erlaubt, die Worst-case-Antwortzeit eines Dienstes zu ermitteln. Da dieses Verfahren für größere Systeme nicht in sinnvoller Zeit durchführbar ist, wurde ein weiterer, approximativer Algorithmus, der sog. Offset-Algorithmus, entwickelt, der die Berechnung einer oberen Schranke der Worst-case-Antwortzeit eines Dienstes in vertretbarer

Zeit zuläßt. Simulative Methoden, wie sie in der Verkehrstheorie oft Verwendung finden, können bei der Ermittlung dieser Worst-case-Antwortzeiten nicht eingesetzt werden, da sie, aufgrund ihrer statistischen Ergebnisse, keine Garantie darüber liefern können, ob tatsächlich alle möglichen Systemzustände berücksichtigt wurden.

Erbringt die Analyse das Ergebnis, daß Antwortzeiten der Dienste ihre Fristen verletzen, muß nach alternativen Prioritätenverteilungen gesucht werden. Die Grundüberlegung für dieses Vorgehen ist, Dienste, die ihre zeitlichen Anforderungen problemlos einhalten konnten, durch Veränderung der Prioritäten ihrer Aktivitäten in ihrer Antwortzeit so zu verlängern, daß sie ihre Anforderung gerade noch erfüllen. Die Verlängerung der Antwortzeiten dieser Dienste führt gleichzeitig zu einer Verkürzung der Antwortzeiten von Diensten, die bisher ihre zeitlichen Randbedingungen nicht einhalten konnten. Die Modifikation der Prioritäten wird, aufgrund der *NP*-Vollständigkeit dieses Problems, durch den Einsatz naturanaloger Optimierungsverfahren erreicht.

Von den untersuchten Optimierungsverfahren zeigte dabei Simulated Annealing sehr konstante Eigenschaften, da die Zahl der benötigten Iterationen keine große Streuung aufwies. Threshold Accepting hingegen weist eine hohe Streuung bei der Anzahl an notwendigen Iterationen auf, kann jedoch schneller Ergebnisse liefern als Simulated Annealing. Die Evolutionsstrategien schließlich zeigten in der eingesetzten, gegenüber der Originalversion reduzierten Form die besten Eigenschaften der untersuchten Optimierer. Für den Einsatz im hier vorliegenden Problem schnitt eine Kombination dieser Verfahren am besten ab: Ermittelt man eine Anfangslösung mit dem vorgestellten heuristischen Distributed-Deadline-Monotonic-Verfahren und wendet auf diese Prioritätenverteilung Simulated Annealing oder die Evolutionsstrategien an, so erhält man in relativ kurzer Zeit ein Ergebnis, sofern gültige Lösungen für das spezifizierte System existieren. (Dieses Ergebnis konnte auch an weiteren Beispielen, die hier allerdings nicht dokumentiert sind, bestätigt werden.) Anschließend können durch den Einsatz von Threshold Accepting relativ schnell weitere Lösungen gefunden werden, da sich dieser Optimierungsalgorithmus zunächst vorrangig auf dem gefundenen Qualitätsplateau weiterbewegt.

Wichtig für den gerade beschriebenen Einsatz von Optimierungsverfahren war die Aufstellung einer Qualitätsfunktion, die dem Optimierer als „Sensor“ in das System dient und ihm die Veränderungen der durchgeführten Optimierung zurückmeldet. Anhand der Qualitätsfunktion konnte auch der, für diese Art von diskreten kombinatorischen Optimierungsproblemen typische, Verlauf der Qualität in Form von Qualitätsniveaus festgestellt werden.

Durch die in dieser Arbeit entwickelte Methode zur Garantie von Worst-case-Antwortzeiten findet eine Transformation des ereignisgesteuerten Systems in ein „quasi“-zeitgesteuertes TDM-System statt. Dabei werden Ereignisse, die zu bestimmten Zeitpunkten gleichzeitig Betriebsmittel beanspruchen, durch die vergebenen Prioritäten in die gewünschte Reihenfolge gebracht. Die Prioritäten sorgen somit für das entsprechende Zeitverhalten. Die (angestrebte) Garantie der Einhaltung des Worst-cases geht allerdings auf Kosten einer relativ geringen Systemauslastung.

Diese prinzipbedingten und notwendigen freien Systemressourcen können jedoch problemlos von zusätzlichen Anforderungen und Diensten genutzt werden, für die keine harten Fristen vorgegeben sind, solange sie keine Interferenz auf andere Dienste produzieren. Dies gilt uneingeschränkt für unterbrechende Systeme, sofern diese zusätzlichen Dienste niedrigere Prioritäten haben und Semaphoren nicht zugelassen werden. In nichtunterbrechenden Systemen muß die mögliche Blockierung (insbesondere des Dienstes mit der bisher niedrigsten Priorität) durch die neu hinzugekommenen Dienste berücksichtigt werden. Wird ein System erweitert,

ohne daß sämtliche Komponenten umkonfiguriert werden sollen, so ist diese Möglichkeit der Ausnutzung freier Systemressourcen ein großer Vorteil gegenüber den reinen zeitgesteuerten TDM-Systemen, da hier der freie Platz nicht genutzt werden kann. In derartigen TDM-Systemen müssen bei einer unerwarteten Erweiterung sämtliche Komponenten überarbeitet werden.

Mit dem abschließend vorgestellten Werkzeug besteht für einen Netzplaner die Möglichkeit, bereits in der Designphase das System so auszulegen, daß die geforderten Randbedingungen eingehalten werden. Damit muß man sich u.U. von den bisher eingesetzten Vergabemethoden trennen, die nicht immer die systemweiten Auswirkungen der Prioritätenvergabe berücksichtigt haben. Ist es zunächst nicht möglich, das Einhalten der Randbedingungen durch Anwendung der präsentierten Methode zu erreichen, dann kann in dieser frühen Phase bereits eine Korrektur durchgeführt und die verwendete Hardware und/oder Software modifiziert werden. Ist die Korrektur vorgenommen, muß mit Hilfe des Werkzeugs ihre Auswirkung überprüft werden. Durch die Garantie der Echtzeiteigenschaften ist der Einsatz verteilter Prioritätensysteme somit auch für sicherheitskritische Systeme (wie z.B. X-by-wire-Anwendungen) problemlos möglich.

Die Anwendung dieser Methode ist natürlich nicht alleine auf Fahrzeugnetze beschränkt, sondern kann für alle Arten von verteilten Echtzeitprioritätensysteme zur Anwendung kommen.

Läßt man Detailverbesserungen sowohl am vorgestellten Werkzeug als auch an den verwendeten Algorithmen, die sich während des Einsatzes ergeben, außer acht, so existieren außerdem drei interessante Richtungen, in welche die präsentierten Verfahren weitergeführt werden können:

- Zum einen die Weiterentwicklung des Analysealgorithmus, um mit dessen Ergebnis näher an die untere Schranke des Worst-case zu gelangen.
- Zum anderen ist eine Verfeinerung der Optimierungsverfahren denkbar. Hierzu sind vor allem Parameterstudien von Interesse, die weitere Hinweise auf die Wahl der Optimierungsparameter geben, um die Zahl an notwendigen Iterationen so klein wie möglich zu halten. Dies kann, trotz des relativ robusten Verhaltens der Optimierungsalgorithmen gegenüber den eingestellten Parametern, zu einer Reduktion der Bearbeitungszeit führen. Außerdem wäre eine Heuristik interessant, die Aufschluß darüber gibt, wie die bestehenden Echtzeitverhältnisse aussehen müßten, damit sich der Einsatz der Optimierungsverfahren überhaupt lohnt.
- Schließlich ist eine vollständige Werkzeugintegration erstrebenswert. Wird hier eine Komponenten- und Softwarebibliothek mit Daten über verfügbare Prozessoren und Softwarebausteine eingebunden, so kann man sich Entwicklungssysteme vorstellen, die, abhängig von eingestellten Präferenzen (Kosten, bestimmte Hersteller, etc.), eigenständig Dimensionierungs- und Bauteilvorschläge unterbreiten.

Im Einleitungskapitel dieser Arbeit wurde eine von Conway formulierte Warnung vor der Schwierigkeit des Ressourcenplanungsproblems in verteilten Systemen aufgeführt. Zu behaupten, daß in der vorliegenden Arbeit alle Probleme des Shop-Schedulings gelöst wurden, wäre vermessen. Nichtsdestotrotz wurde eine äußerst effiziente Methode dargestellt, mit deren Hilfe die Planung und Realisierung von verteilten Prioritätensystemen mit harten Echtzeitanforderungen überhaupt erst ermöglicht sowie signifikant unterstützt wird. Nicht zuletzt die Garantie der zeitlichen Anforderungen im zu planenden System macht diese Methode so wertvoll.

Literaturverzeichnis

Die aufgeführten Literaturstellen entsprechen dem Wissensstand bei der Einreichung der Arbeit im Januar 1997.

[1] Abdelzaher, T.F.; Shin, K.G.: „Optimal Combined Task and Message Scheduling in Distributed Real-Time Systems“, *Proceedings of IEEE Real-Time Systems Symposium*, Pisa, S. 162 ff, 1995

[2] Ablay, P.: „Optimieren mit Evolutionsstrategien“, *Spektrum der Wissenschaft*, S. 104 ff, Juli 1987

[3] Audsley, N.C.: „Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times“, *Internal Report YCS164*, Dept. of Computer Science, University of York, November 1991

[4] Audsley, N.C.; Burns, A. et al.: „Hard Real-Time Scheduling: The Deadline Monotonic Approach“, *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, 1991

[5] Audsley, N.; Burns, A. et al.: „Applying New Scheduling Theory to Static Priority Preemptive Scheduling“, *Software Engineering Journal*, Vol. 8, No. 5, S. 284–292, Sept. 1993

[6] Audsley, N.C.: „Deadline Monotonic Scheduling“, Internal Report YCS146, Dept. of Computer Science, University of York, 1990

[7] Audsley, N.C.; Burns, A. et al.: „Fixed Priority Pre-Emptive Scheduling: An Historical Perspective“, *Real-Time Systems*, Vol. 8, S. 173–198, 1995

[8] Auzins, J; Wilhelm R.V.: „Automotive Electronics-Getting in Gear for the 90s and Beyond“, *IEEE Circuits & Devices*, S. 14 ff, Januar 1994

[9] Bäck, T.; Hoffmeister, F.; Schwefel, H.-P.: „A Survey of Evolution Strategies“, *Proc. of the 4th Intl. Conf. on Genetic Algorithms*, 1991

[10] Bäck, T.; Schwefel, H.-P.: „An Overview of Evolutionary Algorithms for Parameter Optimization“, *Evolutionary Computing*, Vol 1, No.1, S.1 ff., 1993

[11] Barth, J.: *Simulation von Steuergerätenetzen für Fahrzeuge*, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1996

[12] Beike, A.: *ABUS Applikations-Handbuch*, Volkswagen, V. 1.1, 1992

[13] Bettati, R.: *End-to-end scheduling to meet deadlines in distributed systems*, Dissertation, Dept. of Computer Science, University of Illinois, Urbana, 1994

[14] Böndel, B.: „Highway im Auto“, *Wirtschaftswoche*, Nr.41, S. 134 ff., Oktober 1994

[15] Booch, G.: *Object-Oriented Analysis and Design with Applications*, Benjamin Cummings, Second Edition, 1994

[16] Bosch, M.: *Kopplung von Kommunikationsnetzen: Architekturen, Leistungsuntersuchungen und eine Beispielrealisierung*, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1992

[17] Brenner, J; Lesky, P: *Mathematik für Ingenieure und Naturwissenschaftler I-IV*, 2. Auflage, Akademische Verlagsgesellschaft Wiesbaden, 1982

[18] Bresch, C.; Hausmann, R.: *Klassische und molekulare Genetik*, Springer Verlag, Berlin, 1972

[19] Burchard, A.; Liebeherr, J. et al.: „New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems“, *IEEE Transactions on Computers*, Vol. 44, No. 12, Dezember 1995, S. 1429 ff

[20] Burns, A.: „Scheduling Hard Real-Time Systems: A Review“, *Software Engineering Journal*, Vol. 6, No. 3, S. 116–128, 1991

[21] Burns, A.; Tindell, K.; Wellings, A.: „Effective Analysis for Engineering Real-Time Fixed Priority Schedulers“, *IEEE Transactions on Software Engineering*, Vol. 21, No. 5, S. 475 ff., Mai 1995

[22] Casavant, T. L.; Kuhl, J.G.: „A Taxonomy of Scheduling in General Purpose Distributed Computing Systems“, *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, S. 141–154, Februar 1988

[23] Charzinski, J.: *Bewertung der Fehlersicherheit im CAN-Protokoll*, Semesterarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1991

[24] Ciocan, C.: „The Domestic Digital Bus System (D2B) A Maximum of Control Convenience in Audio Video“, *IEEE Trans. on Consumer Electronics*, Vol. 36, No. 3, S. 619–622, August 1990

[25] Conway, R.W.; Maxwell, W.L.; Miller, L.W.: *Theory of Scheduling*, Addison-Wesley, 1967

[26] Czerwinski, M.: *Untersuchung von Algorithmen zur Optimierung von Systemen mit festen Prioritäten*, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1996

[27] Darwin, C.: *Die Entstehung der Arten*, (Reprint von „The Origin of Species by Means of Natural Selection“, 1859), Reclam-Verlag, Stuttgart, 1963

[28] Davis, R.; Wellings, A.: „Dual Priority Scheduling“, *Proc. IEEE Real-Time Systems Symposium*, Pisa, S. 100 ff, 1995

[29] Davis, L. (Ed.): *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991

[30] DiNatale, M.; Stankovic, J.P.: „Applicability of Simulated Annealing Methods to Real-Time Scheduling and Jitter Control“, *Proc. IEEE Real-Time Systems Symposium*, Pisa, S. 190 ff, 1995

[31] DIN 44300, *Realzeitverarbeitung*, Nr 9.2.11, Oktober 1985

[32] Ditfurth, H.v.: *Im Anfang war der Wasserstoff*, Hoffmann und Campe Verlag, Hamburg, 1972

[33] Dueck, G.; Scheurer, T; Wallmeier, H.-M.: „Toleranzschwelle und Sintflut: neue Ideen zur Optimierung“, *Spektrum der Wissenschaft*, S. 42–51, März 1993

[34] Duppel, M.: *Entwurf und Implementierung eines Modells zur Simulation von Kommunikations-Software in verteilten Systemen*, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1995

[35] Färber, G.: „Feldbus-Technik heute und morgen“, *atp*, Oldenbourg Verlag, November 1994

[36] Filho, J.L.R.; Treleaven, P.C.; Alippi, C.: „Genetic-Algorithm Programming Environments“, *IEEE Computer*, Vol. 27, No. 6, S. 28 ff, Juni 1994

[37] Fogel, L.J.; Owens, A.J.; Walsh, M.J.: *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966

[38] Ford Motor Company: *Standard Corporate Protocol*, Specification Version 1.0, April, 1989

[39] Friedrichsohn, O.: *Bestimmung der Restfehlerwarscheinlichkeit der Datenbusse VAN und J1850*, Semesterarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1993

[40] Friedrichsohn, O.: *Modellierung und Simulation von CAN-Systemen mit Hilfe einer objektorientierten Simulationsbibliothek*, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994

[41] Gamma, E.; Helm, R. et al.: *Design Patterns*, Addison-Wesley, 1995

[42] Garey, M.R.; Johnson, D.S.: *Computers and Intractability - A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979

[43] Gemkow, U.; Kocher, U.: „A network Architecture for the Intra-Car Communication“, *Proc. of the 3rd PROMETHEUS Workshop*, S. 389 ff., April 1990

[44] Gerber, R.; Hong, S; Saksena, M.: „Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes“, *Proc. IEEE Real-Time Systems Symposium*, S. 192 ff., 1994

[45] Gerber, R.; Hong, S; Saksena, M.: „Guaranteeing Real-Time Requirements With Resource-Based Calibration of Periodic Processes“, *IEEE Trans. on Software Engineering*, Vol. 21, No. 7, S. 579–592, Juli 1995

[46] Germer, F.: *Untersuchung von topologischen Entwurfsverfahren für Kommunikationsnetze*, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1993

[47] Gillies, D.W.; Liu, J.W.S.: „Scheduling Tasks with AND/OR Precedence Constraints“, *SIAM Journ. on Computing*, Vol. 24, No. 4, S. 797–810, August 1995

[48] Goldberg, D.E.: *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, Reprint, 1989

[49] Goldberg, D.E.: „Genetic and Evolutionary Algorithms Come of Age“, *Communications of the ACM*, Vol. 37, No.3, S. 113 ff, März 1994

[50] Gray, F.: *Pulse Code Communication*, U.S. Patent 2 632 058, März 1953

[51] Gregory, J.W.: *Linear Programming FAQ*, sci.op-research, ftp://rtfm.mit.edu/pub/usenet/sci.answers/linear-programming-faq, Mai 1996

[52] Gregory, J.W.: *Nonlinear Programming FAQ*, sci.op-research, ftp://rtfm.mit.edu/pub/usenet/sci.answers/nonlinear-programming-faq, Mai 1996

[53] Gremmelmaier, U.: *Methoden zur Planung von hierarchischen Hochgeschwindigkeitsnetzen (MAN)*, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1995

[54] Ha, R.; Liu, J.W.S.: *Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems*, Technical Report UIUCDCS-R-93-1833, Dept. of Computer Science, University of Illinois, Urbana, Oktober 1993

[55] Halang, W.A.; Hommel, G.; Lauber, R.: „Perspektiven der Informatik in der Echtzeitverarbeitung“, *Informatik-Spektrum*, Band 16, Heft 6, S. 357–360, Dezember 1993

[56] Han, C.-C.; Lin, K.-J.; Hou, C-J.: „Distance-Constrained Scheduling and Its Applications to Real-Time Systems“, *IEEE Trans. on Computers*, Vol. 45, No. 7, S. 814–825, Juli 1996

[57] Härbour, M. G.; Klein, M. H.; Lehoczky, J. P.: „Timing Analysis for Fixed Priority Scheduling of Hard Real-Time Systems“, *IEEE Trans. on Software Engineering*, Vol. 20, No. 1, S. 13 ff., Januar 1994

[58] Harter, P. K.: „Response Times in Level-Structured Systems“, *ACM Transactions on Computer Systems*, Vol. 5, No. 3, S. 232–248, August 1987

[59] Heberle, K.: „So schnell wie nötig, so langsam wie möglich - Intelligenter Eindrahtbus für Automobilanwendungen“, *Elektronik*, Heft 19, S. 78 ff, 1992

[60] Heitkötter, J.; Beasley, D.: *The Hitch-Hiker’s Guide to Evolutionary Computing*, FAQ for comp.ai.genetic, Version 3.4, posted Dezember 1995

[61] Herrmann, M.: *Spezifikation und Implementierung des OSEK-Netzmanagements für D2B*, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1996

[62] Herrtwich, R. G.: „Betriebsmittelvergabe unter Echtzeitgesichtspunkten“, *Informatik Spektrum*, 14, S. 123–136, 1991

[63] Herzog, U.: „Preemption-Distance Priorities in Real-Time Computer Systems“, *ntz*, Heft 4, S. 201 ff., 1972

[64] Herzog, U.: „Optimal Scheduling Strategies for Real-Time Computers“, *IBM Journal of Research and Development*, Vol. 19, No. 5, S. 494 ff., September 1975

[65] Hillier, F. S.; Liebermann, G. J.: *Operations Research*, R. Oldenbourg Verlag, München, 4. Auflage, 1988

[66] Holland, J.H.: „Outline for a logical theory of adaptive systems“, *Journal of the ACM*, No. 3, S. 297 ff., 1962

[67] Holland, J.H.: *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, 1975

[68] Hoyler, G.: *Entwurf und Implementierung einer offenen Werkzeugumgebung*, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994

[69] Hsueh, C.; Lin,K.-J.; Fan, N.: „Distributed Pinwheel scheduling with End-to-End Timing Constraints“, *Proc. of the IEEE Real-Time Systems Symposium*, Pisa, S. 172 ff., 1995

[70] ISO 7498: *Information Processing Systems – Open Systems Interconnection – Basic Reference Model*, 1984

[71] ISO 8824: *Information Processing Systems – Open System Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*, Dezember, 1987

[72] ISO 11519-1: *Road vehicles - Low speed serial data communication - Part 1: Controller Area Network (CAN)*, ISO-DIS 11898 *Road vehicles - High speed serial data communication* (CAN), 1992

[73] ISO 11519-2: *Road vehicles - Low speed serial data communication - Part 2: Vehicle Area Network* (VAN), 1992

[74] ISO 11519-3: *Road vehicles - Low speed serial data communication - Part 3: Class B data communication network interface (J 1850)*, 1992

[75] Jähnert, J. „Untersuchung der Sicherheit des Time Triggered Protocol (TTP)“, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverabeitung, Universität Stuttgart, 1996

[76] Jaiswal, N.K.: *Priority Queues*, Academic Press, 1968

[77] Johnson, D.S.; Aragon, C.R.; et al.: „Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning“, *Operations Research*, Vol. 37, No. 6, S. 865–892, November-Dezember 1989

[78] Johnson, D.S.; Aragon, C.R.; et al.: „Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning“, *Operations Research*, Vol. 39, No. 3, S. 378 ff., Mai–Juni 1991

[79] Joseph, M.; Pandya, P.: „Finding Response Times in a Real-Time System“, *Computer Journal* (BCS), Vol. 29, No. 5, S. 390–395, Oktober 1986

[80] Jungnickel, D.: *Graphen, Netzwerke und Algorithmen*, BI-Wiss. Verlag, Mannheim, 3. Auflage, 1994

[81] Katcher, I.; Sathaye, S.S.; Strosnider, J.K.: „Fixed Priority Scheduling with Limited Priority Levels“, *IEEE Trans. on Computers*, Vol.44, No.9, S. 1140 ff., September 1995

[82] Kern, R.: „Prozeßauswahl und Ablaufplanung in Echtzeit-Sytemen“, *Elektronik*, No. 14, 1992, S. 26 ff.

[83] Kettler, K.A.; Lehoczky, J.P.; Strosnider, J.K.: „Modeling Bus Scheduling Policies for Real-time Systems“, *Proc. of the IEEE Real-Time Systems Symposium*, Pisa, S. 242 ff, 1995

[84] Khuri, S.; Bäck, T.; Heitkötter, J.: „An Evolutionary Approach to Combinatorial Optimization Problems“, *Proc. of the CSC*, Phoenix, März 1994

[85] Kiencke, U.: „Controller Area Network - from Concept to Reality“, *Proceedings of the 1. CAN Conference*, S. 0-11 ff., Sept. 1994

[86] Klein, M.H.; Lehoczky, J.P.; Rajkumar, R.: „Rate-Monotonic Analysis for Real-Time Industrial Computing“, *IEEE Computer*, Vol. 27, No.1, S. 24 ff, Januar 1994

[87] Klein, M.H.; Ralya, T. et al.: *A Practioner’s Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers, Boston, 1993

[88] Klipstein, D.L.: „Automobile brauchen mehr elektronische Intelligenz“, *VDI Nachrichten*, Nr. 41, Oktober 1996

[89] Knippers, R.: *Molekulare Genetik*, Georg Thieme Verlag, Stuttgart, 6. Auflage, 1995

[90] Kocher, H.: *Entwurf und Implementierung einer Simulationsbibliothek unter Anwendung objektorientierter Methoden*, Dissertation, Insitut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994

[91] Kopetz, H.; Damm, A.; et al.: „Distributed Fault-Tolerant Real-Time Systems: The MARS Approach“, *IEEE Micro*, Vol. 9, No. 1; S. 25–40, Februar 1989

[92] Kopetz, H.; Grünsteidl, G.: „TTP - A Protocol for Fault-Tolerant Real-Time Systems“, *IEEE Computer*, Vol. 27, No.1, S. 14 ff., Januar 1994

[93] Kopetz, H.: „A Solution to an Automotive Control System Benchmark“, *IEEE Proceedings of the Real-Time Systems Symposium*, S. 154 ff., 1994

[94] Kühn, P. J.: „*Über die Berechnung der Wartezeiten in Vermittlungs- und Rechensystemen*“, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1972

[95] Kühn, P.J.; Gemkow, U., et al: Interworking between Intra-Car and Extra-Car Communication Networks, *Proc. of the 3rd PROMETHEUS Workshop*, S. 471, April 1990

[96] Kühn, P.J.: *Nachrichtenvermittlung I+II*, Script zur Vorlesung, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1995

[97] Kühn, P.J.: *Datenverarbeitung I*, Script zur Vorlesung, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994

[98] Kühn, P.J.: *Hochgeschwindigkeits-Kommunikationsnetze*, Script zur Vorlesung, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1995

[99] Kühner, T.; Zimmermann, C.; et. al.: „Modulare Vernetzungsstrukturen - Netztopologien, Strukturanforderungen, Montagebelange - “, *Elektronik im Kraftfahrzeug*, VDI-Bericht 1152, S. 101 ff., 1994

[100] Kull, U.; Knodel, H.: *Genetik und Molekularbiologie*, J.B. Metzlersche Verlagsbuchhandlung Stuttgart, 2. Auflage, 1980

[101] Kuntz, W.; Mores, R.: „CAN Operated on an Optical Double Ring at Improved Fault-Tolerance“, *ETT,* Vol. 3, No. 4, S. 465–470, July–August 1993

[102] Lampson, B.W.; Redell, D.D.: „Experiences with Processes and Monitors in Mesa“, *Communications of the ACM*, Vol. 23, No. 2, S. 105–117, Februar 1980

[103] Laraqui, K.; Nazari, et al.: „Communication Systems Architecture - A Case Study“, *Proc. of the 3rd PROMETHEUS Workshop*, S. 472, April 1990

[104] Lauber, R.: *Prozeßautomatisierung*, Band 1, Springer-Verlag, 2. Auflage, 1989

[105] Lawrenz, W.: „Vernetzte Systeme im Automobil der Zukunft“, *Informatik Spektrum*, S. 107–109, September 1993

[106] Lee, J.; Lee, S.; Kim, H.: „Scheduling Soft Aperiodic Tasks in Adaptable Fixed-Priority Systems“, *ACM Operating Systems Review*, Vol. 30, No. 4, S. 17–28, Oktober 1996

[107] Lehoczky, J.P.; Sha, L.; Strosnider, J.K.: „Enhanced Aperiodic Scheduling in Hard Real-Time Environments“, *Proc. of the IEEE Real-Time Systems Symposium*, Los Alamitos, S. 261–270, 1987

[108] Lehoczky, J.P.: „Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines“, *Proc. of the IEEE Real-Time Systems Symposium*, S. 1–30, 1990

[109] Leinbaugh, D.W.: „Guaranteed Response Times in Hard-Real-Time Environment“, *IEEE Trans. on Software Engineering*, Vol. SE-6, No. 1, S. 85–91, Januar 1980

[110] Leung, J.Y.T.; Whitehead, J.: „On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks“, *Performance Evaluation*, Vol. 2, No. 4, S. 237–250, Dezember 1982

[111] Liu, C.L.; Layland, J. W.: „Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment“, *Journal of the Association for Computing Machinery*, Vol. 20, No. 1, S. 46–61, Januar 1973

[112] Liu, C.L.: „Fundamentals of Real-Time Scheduling“, *Real-Time Computing*, (Halang, ed.), NATO ASI Series, 1994

[113] Liu, C.Y.; Bulfin, R.L.: „Scheduling Open Shops with Unit Execution Times to Mimimize Functions of Due Dates“, *Operations Research*, Vol. 36, No. 4, S. 553 ff., July–August 1988

[114] Locke, C. D.: „Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives“, *Journal of Real-Time Systems*, 4, S. 37 ff., Juni 1992

[115] Masur, K.-D.: „GSM-Datenübertragung bei kleinen, mittleren und sehr großen Fahrgeschwindigkeiten“, *ntz*, Heft 8, S. 16–23, 1996

[116] Metropolis, N.; Rosenbluth, M.; Teller, E. et al.: „Equation of State Calcualtion by Fast Computing Machines“, *Journal of Chemistry and Physics*, No. 21, S. 1087–1092, 1953

[117] Meyer, H.; Daumer, K.: *Evolution*, bsv, 1985

[118] Mok, A.K.: *Fundamental design problems of distributed Systems for the hard-real-time environment*, Dissertation, Dept. of Electrical Engineering and Computer Science, Mass. Institute of Technology, Cambridge, 1983

[119] NTG Empfehlung 0902 „Nachrichtenvermittlungstechnik Begriffe“, *ntz*, Bd. 35 Heft 8, S. 549 ff., 1982

[120] NTG Empfehlung 0903 „Nachrichtenverkehrstheorie Begriffe“, *ntz*, Bd. 37 Heft 7, S. 465 ff., 1984

[121] O’Brien, S.J. (Hrsg.): *Genetic Maps*, 6th ed., Cold Spring Harbor Lab Press, Cold Spring Harbor, 1993

[122] Parashkevov, A. „Distributed Real-Time Computer Systems“, Internal Report, Dept.of Computer Science, Univ. of Adelaide, 1995

[123] Park, K. S.; Yun, D. K.: „Optimal Scheduling of Periodic Activities“, *Operations Research*, Vol. 33, No. 3, S. 690 ff., Mai–Juni 1985

[124] Puschner, P.; Koza, C.: „Calculating the Maximum Execution Time of Real-Time Programs“, *Journal of Real-Time Systems*, Vol.1, S. 159–176, 1989

[125] Raisch, A.: *Spezifikation und Implementierung von Modellen zur objektorientierten Simulation von Steuergerätenetzen unter der Verwendung lokaler Scheduler*, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1996

[126] Raith, T.; Thurner, T.; Bogner, A.: „Netzwerke zur Integration von Systemfunktionen der Kraftfahrzeug-Elektronik“, *Informationstechnik und Technische Informatik (it+ti)*, Jahrg. 37, Nr. 6, S. 28 ff, Juni 1995

[127] Raji, R.S.: „Smart networks for control“, *IEEE Spectrum*, No. 6, S. 49–55, June 1994

[128] Rajkumar, R.; Sha, L.; Lehoczky, J.P.: „Real-Time Synchronisation Protocols for Multiprocessors“, *Proc. of the IEEE Real-Time Systems Symposium*, Huntsville, S. 259–269, 1988

[129] Rzehak, H.: „Echtzeit-Datenverarbeitung - Grundlagen und Methoden für die Praxis“, *Elektronik*, Nr. 5, 1991, S. 202 ff

[130] SAE: *Class C Application Requirement Considerations - SAE J2056/1 Jun93, Handbook 1994*, Volume 2: Parts and Components, S. 23.366 ff, 1994

[131] Schmitt, W.: *Verkehrsanalyse von Warteschlangennetzen mit Prioritäten*, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1984

[132] Schneider, M.: „*Optimierung von Systeme mit festen Prioritäten*“, Diplomarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1995

[133] Schödel, W.: *Kopplung von DQDB-Regionalnetzen mit ATM-Weitverkehrsnetzen: Architektur, Steuerstrategien und Leistungsverhalten*, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994

[134] Schwefel, H.-P.: „*Collective phenomena in evolutionary systems*“, Preprints of the 31st annual meeting of the int’l soc. for general system research, Budapest, S. 1025 ff., 1987

[135] Sedgewick, R.: *Algorithmen in C++*, Addison-Wesley, 1994

[136] Sha, L.; Lehoczky, J.P.; Rajkumar, R.: „Solutions for Some Practical Problems in Priotised Preemptive Scheduling“, *Proc. of IEEE Real-Time Symposium*, S. 181–191, 1986

[137] Sha, L.; Rajkumar, R.; Lehoczky, J. P.: „Priority Inheritance Protocols: An Approach to Real-Time Synchronisation“, *IEEE Transactions on Computers*, Vol. 39, No. 9, S. 1175 ff, Sept. 1990

[138] Shepard, T.; Gagné, J.A.M.: „A Pre-Run-Time Scheduling Algorithm For Hard Real-Time Systems“, *IEEE Trans. on Software Engineering*, Vol.17, No 7, S. 669–677, Juli 1991

[139] Spuri, M.; Stankovic, J. A.: „How to Integrate Precedence Constraints and Shared Resourcees in Real-Time Scheduling“, *IEEE Trans on Computers*, Vol. 43, No. 12, S. 1407 ff.,Dez. 1994

[140] Srinivas, M; Patnaik, L.M.: „Genetic Algorithms: A Survey“, *IEEE Computer*, Vol. 27, No. 6, S. 17 ff., Juni 1994

[141] Staub, M.: „Mit dem Worst Case kalkuliert - Zugriffsgarantie bei CAN-Netzen“, *Elektronik*, Jahrg. 44, Nr. 12, 1995

[142] Stümpfle, M.: „*Grundzüge der biologischen Informationscodierung*“, Interner Bericht, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1993

[143] Stümpfle, M.: „Ober sticht Unter - Prioritätenvergabe in Echtzeit-Systemen“, *Elektronik*, Jahrg. 44, Nr. 22, S. 102–108, Oktober 1995

[144] Stümpfle, M.; Eberspächer, M.; Kocher, H.: „Design und Realisierung eines offenen Fahrzeug-Kommunikationssystems“, *Interner Bericht*, 1996

[145] Tanenbaum, A.S.: *Computer-Netzwerke*, Wolfram’s Fachverlag, 2. Auflage, 1992

[146] Tanenbaum, A.S.: *Computer-Networks*, Prentice-Hall, 3rd Ed., 1996

[147] Tanenbaum, A.S.: *Moderne Betriebssysteme*, Hanser Verlag, 2. Auflage, 1995

[148] Tangemann, M.: *Modellierung und Analyse von lokalen Hochgeschwindigkeitsnetzen mit zeitgesteuerten Token Passing-Medienzugriffsprotokollen*, Dissertation, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994

[149] Thuel, S.R.; Lehoczky, J.P.: „Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Priority Systems using Slack Stealing“, *Proc. of IEEE Real-Time Symposium*, S. 22–34, 1994

[150] Tindell, K.W.: *Fixed Priority scheduling of Hard Real-Time Systems*, Dissertation, Dept. of Computer Science, University of York, 1993

[151] Tindell, K. W.; Burns, A.; Wellings, A.J.: „Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy“, *Journal of Real-Time Systems* Vol. 4, No. 2, S. 145 ff, Juni 1992

[152] Tindell, K.; Burns, A.: „Guaranteeing Message Latencies on Control Area Network“, *Proc. 1st. Int’l CAN Conference*, Mainz, S. 1-2 ff., Oktober 1994

[153] Triller, M.: „Dynamischer Lastausgleich in verteilten Echtzeit-Systemen“, *Informationstechnik und Technische Informatik*, Jahrg. 35, Heft 5, S. 35 ff, Mai 1993

[154] Vestal, S.: „Fixed-Priority Sensitivity Analysis for Linear Compute Time Models“, *IEEE Trans. on Software Engineeering*, Vol. 20, No. 4, S. 308–317, April 1994

[155] Vogt, S.: *Ermittlung der Phasenlaufzeiten von Echtzeit-Betriebssystemmodulen für Simulationsmodelle*, Studienarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1994

[156] Wagner, A.; Krammer, J.; Paulini, M.: „Bussysteme in der neuen 7-Serie von BMW“, *Elektronik im Kraftfahrzeug*, VDI-Bericht 1152, S. 123–144, 1994

[157] Walke, B.; Decker, P.: „Mobile Datenkommunikation - Eine Übersicht“, *Informationstechnik und Technische Informatik*, Jahrgang 35, Heft 5, S. 12–25, Mai 1993

[158] Whitley, D.: *A Genetic Algorithm Tutorial*, Technical Report CS-93-103, Dept. of Computer Science, Colorado State University, November 1993

[159] Xu, J.; Parnas, D.L.: „Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations“, *IEEE Trans. on Software Engineering*, Vol 16, No.3. S. 360–369, März 1990

[160] Xu, J.: „Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence and Exclusion Relations“, *IEEE Trans. on Software Engineering*, Vol 19, No.2. S. 139–154, Februar 1993

[161] Zuberi, K.M.; Shin, K.G.: „Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications“, *Proceedings of Real-Time Technology and Applications Symposium*, S. 240–249, Mai 1995

[162] Zhao, W.; Ramamritham, K.; Stankovic, J.: „Preemptive Scheduling Under Time and Resource Constraints“, *IEEE Trans. on Computers*, Vol. C-36, No. 8, S. 949–960, August 1987

Anhang

A1 Simulationsergebnisse für das 3-Dienste-Beispielszenario

Die Simulationsergebnisse in den Bildern A-1 und A-2 zeigen das System aus Kapitel 5, diesmal mit ausschließlich unterbrechenden Schedulern bzw. ausschließlich nichtunterbrechenden Schedulern. Außerdem werden im Anschluß die jeweiligen Worst-cases abermals in Form von auf die einzelnen Dienste „aufgespreizten“ Gantt-Diagrammen dargestellt. Die Prioritätenverteilung folgt dem DRM-Verfahren, d.h. alle Aktivitäten von Dienst 1 haben die Priorität 0, alle Aktivitäten von Dienst 2 die Priorität 1 und die Aktivitäten von Dienst 3 haben die Priorität 2.

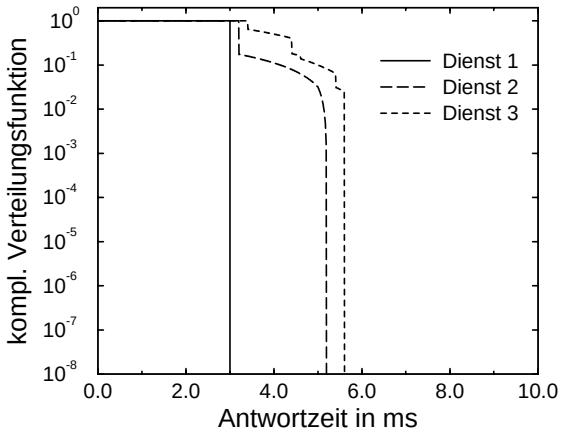


Bild A-1: Simulationsergebnisse des 3-Dienste Systems mit ausschließlich unterbrechenden Schedulern

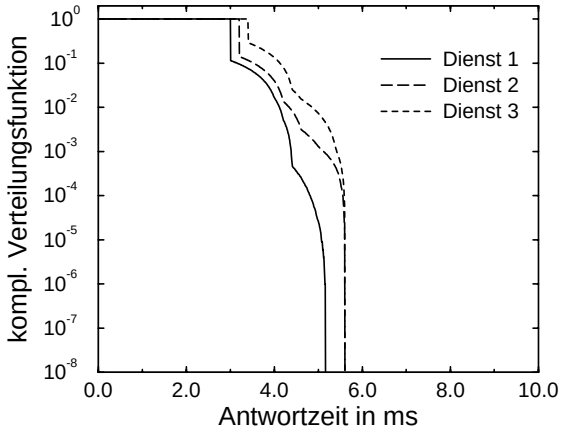
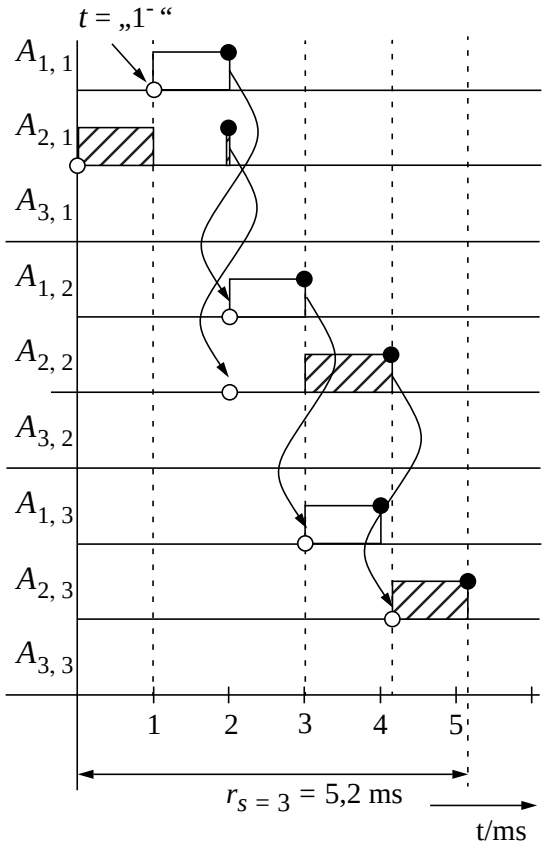
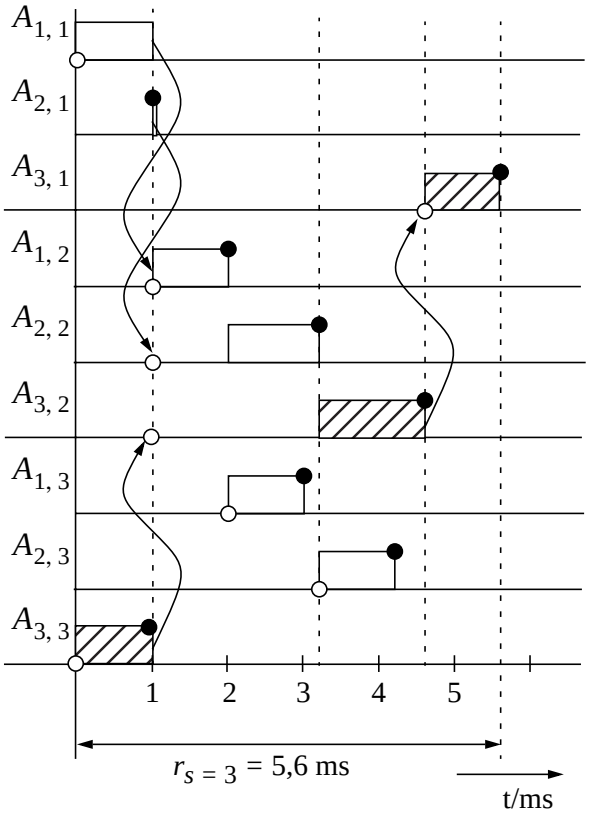


Bild A-2: Simulationsergebnisse des 3-Dienste Systems mit ausschließlich nichtunterbrechenden Schedulern

a) Ausschließlich unterbrechende Scheduler

Für Dienst $s = 1$ ist der Fall trivial, da seine Aktivitäten stets sofort zur Bearbeitung kommen. Seine Antwortzeit entspricht also immer der Summe der Bearbeitungszeiten seiner Einzelaktivitäten. Dies kann auch anhand der Simulationsergebnisse in Bild A-1 belegt werden, die zeigen, daß keine anderen Antwortzeiten als genau diese Summe gemessen wurden (Kurve geht senkrecht nach unten). Bild A-3 zeigt die Verhältnisse für Dienst $s = 2$. Es läßt sich erkennen, daß Dienst $s = 2$ nur von Dienst $s = 1$ unterbrochen werden kann. Dieser Fall tritt allerdings maximal auf zwei Prozessoren auf. Dienst $s = 3$ spielt hier wiederum keine Rolle, da er keinen Beitrag zur Interferenz leisten und Blockierungen nicht auftreten können. Für Dienst $s = 3$ führt der lokale Worst-case auf Prozessor $k = 2$ zum Worst-case (siehe Bild A-4).





○ Einplanung einer Aktivität
● Ende einer Aktivität

Aktivitäten von Dienst $s = 3$

Bild A-3: Worst-case-Szenario für $s = 2$, unterbrechende Scheduler

Bild A-4: Worst-case-Szenario für $s = 3$, unterbrechende Scheduler

b) Ausschließlich nichtunterbrechende Scheduler

Arbeiten alle drei Prozessoren mit nichtunterbrechenden Schedulern, dann besteht der Worst-case für den ersten Dienst darin, daß seine Aktivitäten von den niederprioren Aktivitäten blokkiert werden. Der schlimmste Fall wird jedoch nicht durch die vermeintlich längste Blockierung durch die längste Aktivität $A_{3,2}$ hervorgerufen, sondern besteht aus einer Kombination an Blockierungen der Aktivitäten der beiden niederpriorsten Dienste (siehe Bild A-5). In Bild A-6 ist der Worst-case für Dienst $s = 2$ aufgeführt, der ausschließlich aus der maximal möglichen Blockierung und Interferenz auf dem zweiten Prozessor herrührt.

Der Worst-case für Dienst 3 resultiert, ebenso wie bei Dienst 2, aus den Verhältnissen auf dem zweiten Prozessor ($k = 2$), wodurch er seine maximale Interferenz erfährt. Damit hat Dienst 3 dieselbe maximale Antwortzeit wie Dienst 2. Der Beweis für die Allgemeingültigkeit dieses Sachverhalts, kann dem folgenden Abschnitt (Anhang A 2) entnommen werden.

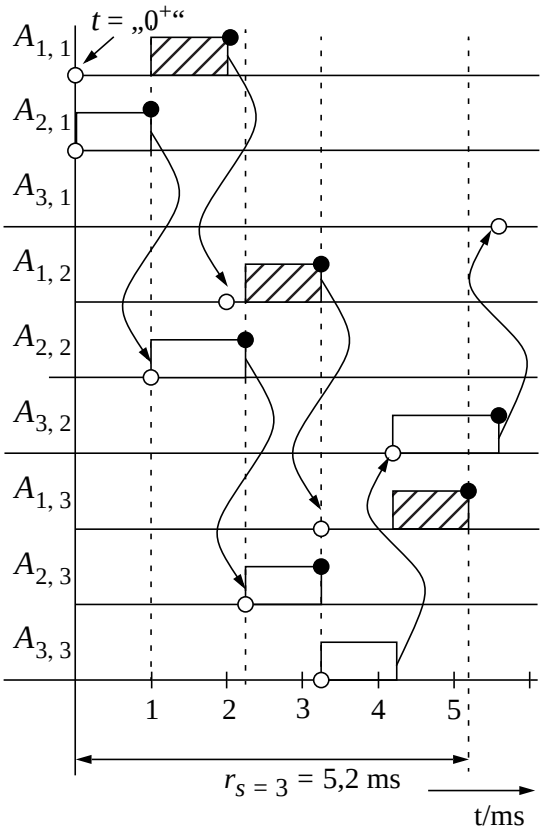


Bild A-5: Worst-case-Szenario für $s = 1$, nichtunterbrechende Scheduler

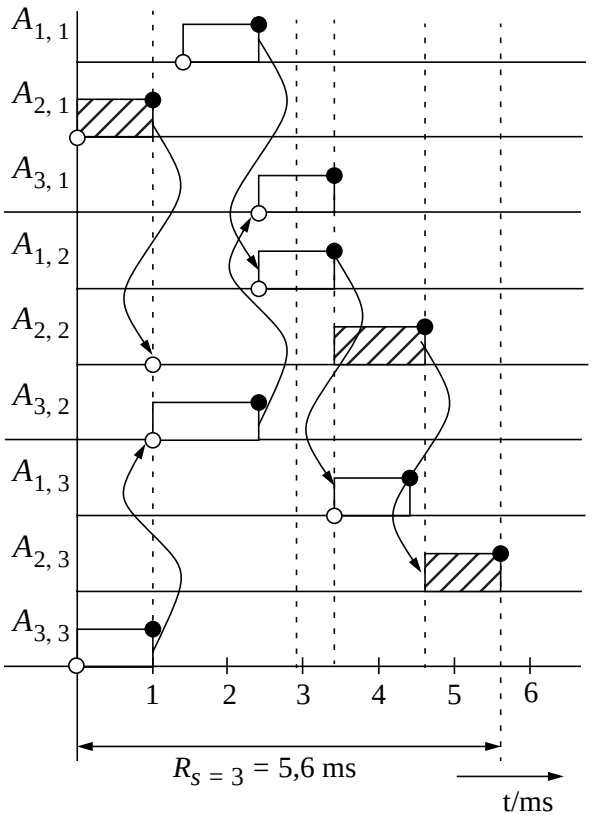


Bild A-6: Worst-case-Szenario für $s = 2$, nichtunterbrechende Scheduler

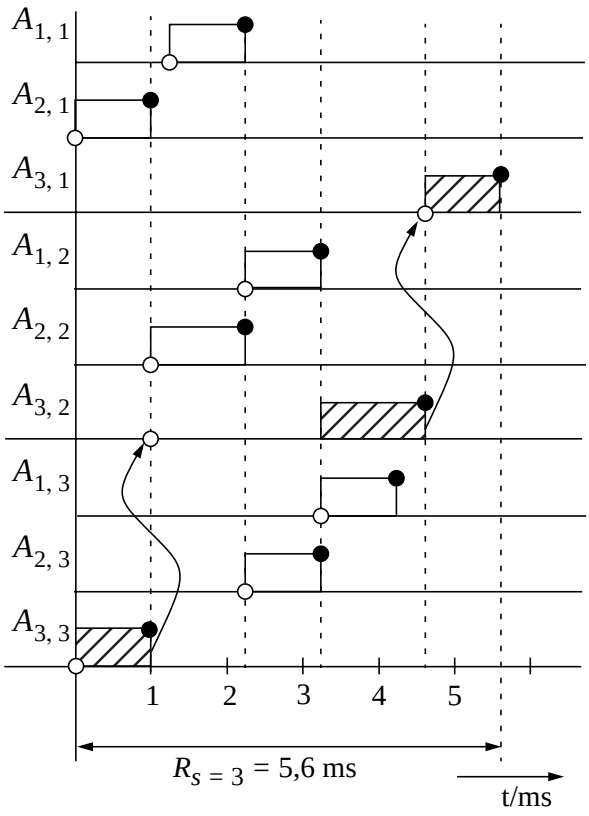
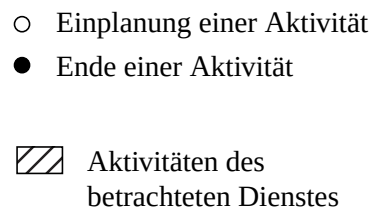


Bild A-7: Worst-case-Szenario für $s = 3$, nicht-unterbrechende Scheduler



A2 Sätze und Beweise für die Analyse und die Optimierung

Beweis zu Kapitel 5: Begrenzung des Anforderungsabstands

Satz: Der Abstand $T(A_s|_a)$ zwischen zwei Anforderungen $A_s|_a$ eines Dienstes *s* auf einem Prozessor und der Position *a* innerhalb des Dienstes ist begrenzt durch:

$$T(A_s|_{a-1}) + C(A_s|_{a-1}) - r(A_s|_{a-1}) \le T(A_s|_a) \le T(A_s|_{a-1}) + r(A_s|_{a-1}) - C(A_s|_{a-1})$$

Wobei über *a*-1 der Prozessor gekennzeichnet ist, auf dem die Vorgängeraktivität $A_s|_{a-1}$ des Dienstes abläuft.

Beweis: Zum Beweis soll folgendes Szenario eines Dienstes dienen, der insgesamt auf drei Prozessoren abläuft (siehe Bild A-8).

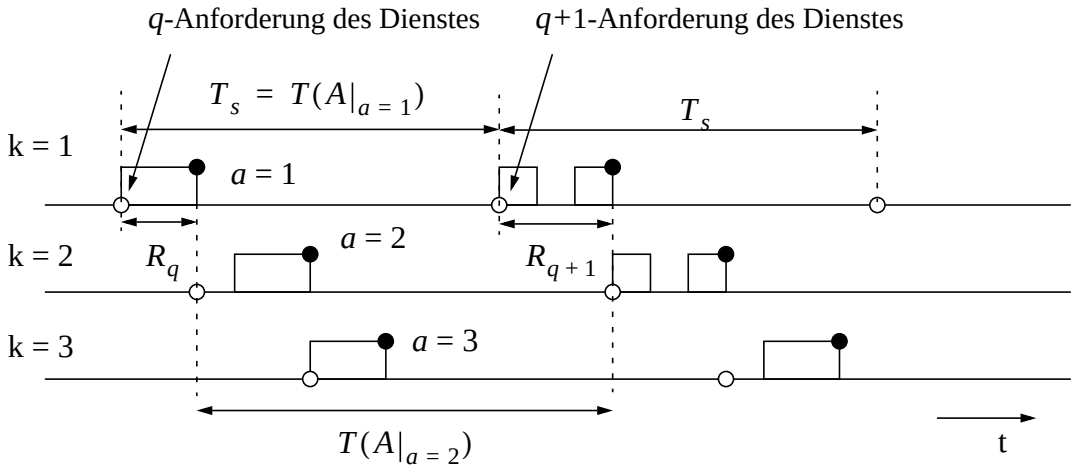


Bild A-8: Anforderungsabstand bei Diensten

Aus dem Bild läßt sich die Zeit für den Anforderungsabstand *T* ablesen:

$$T(A|_{a=2}) = T(A|_{a=1}) + R_{q+1}(A|_{a=1}) - R_q(A|_{a=1})$$

Der rechte Term wird maximal für $R_{q+1}(A|_{a=1}) = r$ und $R_q(A|_{a=1}) = C$. Entsprechend wird er minimal für $R_{q+1} = C$ und $R_q = r$. Damit folgt allgemein, unabhängig von *q*, aber in Abhängigkeit von a:

$$T(A_s|_{a-1}) + C(A_s|_{a-1}) - r(A_s|_{a-1}) \le T(A_s|_a) \le T(A_s|_{a-1}) + r(A_s|_{a-1}) - C(A_s|_{a-1})$$

d. h. der Abstand zwischen zwei Anforderungen kann durch eine obere und eine untere Schranke begrenzt werden.

Beweise zu Kapitel 5: Gleiche Worst-case-Antwortzeit der beiden niederpriorsten Aktivitäten in einem nup-System unter bestimmten Voraussetzungen

Satz: In einem nup-System haben immer die beiden niederpriorsten Aktivitäten dieselbe Worst-case-Antwortzeit, unabhängig von der Zahl an höherprioren Aktivitäten, wenn die Perioden der höherprioren Anforderungen größer sind als diese Worst-case-Antwortzeit. Bei e_k Aktivitäten auf einem Prozessor folgt somit für die Aktivitäten mit den beiden niedrigsten Prioritäten, die den Diensten $s = x$ und $s = y$ angehören, ($p(A_{s=x}) = e_k - 1$ und $p(A_{s=y}) = e_k - 2$):

$$r_x = r_y$$

wenn die zugehörige Wartezeitzeit W der Aktivitäten kleiner ist als die Periodendauern der höherprioren Anforderungen.

Beweis: Die Worst-case-Antwortzeit in einem nup System berechnet sich nach Gl. 3-24 (mit $O_i = 0, q = 0$) zu $r_i = C_i + W_i$, wobei der Index i für die bei 0 beginnende, sortierte Priorität $p(A_k)$ einer Aktivität auf dem Prozessor k steht. Sind die Perioden der höherprioren Aktivitäten länger als die betrachteten Wartezeiten, d.h. es gilt $T_j \geq W_i, \forall j \in hp(i)$, dann reduziert sich die Beziehung zur Berechnung der Wartezeit im Worst-case zu:

$$W_i = B_i + \sum_j C_j$$

Mit $B_y = C_x$ und $B_x = 0$, folgt aus $r_x = r_y$:

$$C_x + 0 + \sum_j C_j = C_y + C_x + \sum_k C_k \qquad j \in hp(x), k \in hp(y)$$

Die Interferenz die Aktivität A_x erfährt, ist diesselbe wie A_y wobei zusätzlich noch deren Bearbeitungsdauer C_y hinzukommt. Es gilt also:

$$\sum_j C_j = C_y + \sum_k C_k$$

und somit

$$C_x + 0 + C_y + \sum_k C_k = C_y + C_x + \sum_k C_k \qquad \forall k \in hp(y)$$

q.e.d.

Beweise zu Kapitel 5: Regeln für die Ermittlung des Worst-case

Zu Regel 2: Ist eine Aktivität auf einem Prozessor eingeplant, so dürfen später keine Zustände entstehen, die dieser Einplanung und dem Ablauf der Aktivität widersprechen.

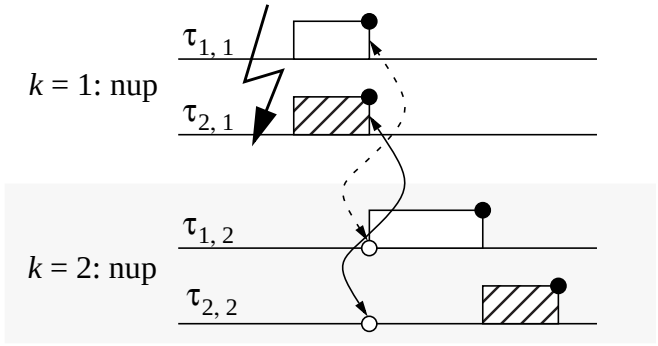
Diese Regel ist notwendig, da die verplanten Aktivitäten die zeitliche Grundlage für die nachfolgenden Aktivitäten darstellen.

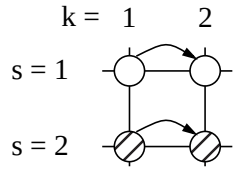
Zu Regel 5: Das verteilte System, das in seiner ursprünglichen Form ein lose gekoppeltes System darstellt, wird durch die Modellierung zu einem eng gekoppelten System, in dem alle Kommunikationsbeziehungen ebenfalls als Aktivitäten mitberücksichtigt werden. Somit

bewirkt das Ende einer Aktivität zur „gleichen“ Zeit das Einplanen der Folgeaktivität auf dem nachfolgenden Prozessor. Dabei kann das zeitliche Verhalten des Betriebssystems durchaus mitberücksichtigt und in gleicher Weise als Aktivität miteingeplant werden. Die Granularität der Betrachtung hängt dabei wesentlich von den Verhältnissen zwischen Betreibssystemaktivitäten und Anwendungsaktivitäten ab. Für sicherheitsrelevanten Aussagen ist es allerdings unerläßlich, alle beteiligten Komponenten in die Betrachtung einzubeziehen.

Zu Regel 6: Befindet sich eine Aktivität auf einem nup-Prozessor, so hat sie, von ihrem Endzeitpunkt aus zeitlich rückwärts betrachtet, ihren Prozessor über die Bearbeitungszeit C belegt, da sie ja nicht unterbrochen werden konnte. Ausgehend von einer Folgeaktivität kann damit die unmittelbare Belegung auf dem nup-Vorgängerprozessor dieser Folgeaktivität ermittelt werden.

Zu Regel 9: Arbeitet der Vorgängerprozessor eines Dienstes mit einem nup-Scheduler, dann kann der kritische Zeitpunkt nur dann auftreten, wenn keine weiteren Dienste diese beiden Prozessoren benutzen. Im Beispiel in Bild A-9 ist dieser Ausschlußfall dargestellt.





Ausgangssituation: angenommener kritischer Zeitpunkt auf Prozessor k = 2

Bild A-9: Verschiebung des kritischen Zeitpunktes bei nichtunterbrechenden Schedulern

Geht man vom Worst-case auf Prozessor k = 2 aus und versucht dann die Aktivitäten auf dem Vorgängerprozessor k = 1 einzuplanen, dann erhält man den Widerspruch, daß beide Aktivitäten gleichzeitig beendet sein müßten, um auf k = 2 gleichzeitig eingeplant zu werden.

Aufwandsabschätzung des optimalen Algorithmus in Kapitel 5

Betrachtet man die Zahl der zu berechnenden lokalen Worst-case-Szenarien als ein Maß E für die Komplexität des Algorithmus, dann berechnet sich dieses aus der Zahl an Prozessoren n und der Zahl der Dienste m im System nach

$$E = E\left(\left(n + \left[1 + \sum_{s=1}^{m-1}\binom{m-1}{s}\right]n\right)m\right)$$

Dabei stammt der erste Summand n aus der initialen Ermittlung aller lokalen Worst-case-Antwortzeiten der Aktivitäten eines Dienstes. Die Binomialsumme resultiert aus den möglichen Kombinationen der Aktivitäten, die auf einem Prozessor berücksichtigt werden müssen, wenn ein ermitteltes Szenario nicht funktioniert und deshalb eine neue Kombination betrachtet wird. Diese Binomialsumme muß im schlimmsten Fall über alle n Prozessoren des Systems gebildet

werden. Schließlich wird die Gesamtsumme noch über alle *m* Dienste des Systems gebildet. Durch einfache Umformung erhält man:

$$E = E\left(mn\left[1 + \sum_{s=0}^{m-1}\binom{m-1}{2}\right]\right)$$

Beweis zu Kapitel 6: Qualitätsfunktion

Satz: Für die Qualitätsfunktion nach Gl. 6-9 wird garantiert, daß es keinen Qualitätswert eines nichtfunktionierenden Systems gibt, der höher ist als der Qualitätswert des schlechtesten, funktionierenden Systems.

Beweis: Im kritischsten Fall verpaßt ein Dienst $s = 1$ seine Deadline minimal um ε ($R_1 = D_1 + \varepsilon$ mit $|\varepsilon| \ll D_1$), wobei alle anderen Dienste des Systems eine Antwortzeit haben, die genau ihrer Bearbeitungszeit entspricht, d.h. $D_s = R_s \;\forall s = 2, \ldots, m$. Laut Anforderung (4) an die Qualitätsfunktion muß gelten:

$$Q_{\varepsilon > 0} < Q_{\varepsilon \le 0}$$

Die Gesamtqualität des Systems berechnet sich nach

$$Q = \frac{1}{m}Q_1 + \frac{1}{m}\sum_{s=2}^{m} Q_s$$

Die Qualität des Dienstes $s = 1$, Q_1, ergibt sich zu:

$$Q_1 = \begin{cases} -\varepsilon\rho_{rel,1}\dfrac{2D_1 + \varepsilon}{D_1^2 - C_1^2} - \sum_{s=2}^{m}\rho_{rel,s} & \varepsilon > 0 \\ -\varepsilon\rho_{rel,1}\dfrac{2D_1 + \varepsilon}{D_1^2 - C_1^2} & \varepsilon \le 0 \end{cases}$$

Die Qualität der übrigen Dienste des Systems berechnet sich zu:

$$Q_s = \rho_{rel,s} \qquad \forall s = 2, \ldots, m.$$

Dieser Qualitätswert muß nicht weiter betrachtet werden, da er in beiden betrachteten Fällen gleich groß ist. Setzt man diese Werte ein, dann erhält man für den Fall, daß die Deadline nicht eingehalten wird ($\varepsilon > 0$):

$$Q_{\varepsilon > 0} = -\frac{\varepsilon\rho_{rel,1}}{m}\frac{2D_1 + \varepsilon}{D_1^2 - C_1^2} < 0 \qquad \forall \varepsilon > 0$$

Für den Fall, daß die Deadline des Dienstes eingehalten wird, gilt:

$$Q_{\varepsilon \le 0} = \sum_{s=2}^{m}\rho_{rel,s} - \frac{\varepsilon\rho_{rel,1}}{m}\frac{2D_1 + \varepsilon}{D_1^2 - C_1^2} > 0 \qquad \forall \varepsilon \le 0$$

Damit ergibt sich: $Q_{\varepsilon > 0} < Q_{\varepsilon \le 0}$

q.e.d.

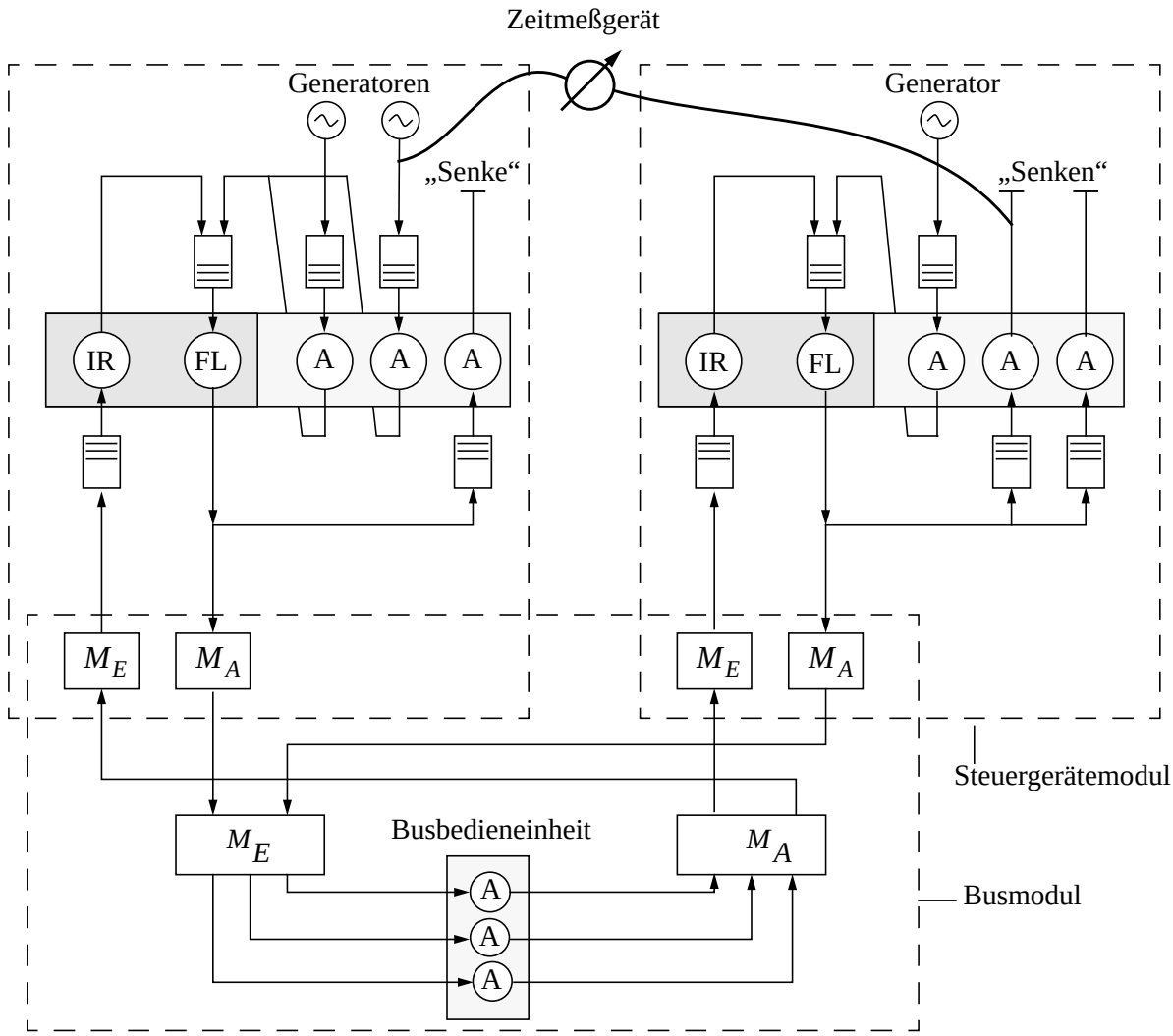
A3 Das Modell zur Steuergerätenetzsimulation

Zur Validierung der Analyse in Kapitel 5 wurden Modelle von Steuergerätenetzen simuliert. Bild A-10 gibt den Aufbau des Systems in Form eines Warteschlangensystems wieder, das in diesem Kapitel untersucht wurde. Zur Messung wurde eine ereignisgesteuerte, zeitdiskrete Simulation durchgeführt und als Ergebnis die komplementären Verteilungsfunktion der Antwortzeiten der beteiligten Dienste bestimmt (vgl. z.B. Bilder A-1 und A-2).

Simuliert werden können alle Netze, die über die in Kapitel 7 erwähnte Spezifikationssprache beschrieben sind, da die Erstellung des Simulationsmodells aus dieser Spezifikation automatisch erfolgt.

Einige Besonderheiten, die dieses Modell betreffen, sind:

- Die Prozessormodelle für die Steuergeräte besitzen mehrere Bereiche mit Bedieneinheiten. Der Interrupt-Level hat dabei die höchste Priorität und kann grundsätzlich alle anderen Aktivitäten unterbrechen. Allerdings wird die Bearbeitungszeit seiner Bedienphase zu 0 angenommen. (Dies läßt sich problemlos ändern, wenn eine Interrupt-Service-Routine modelliert werden soll, deren Bearbeitungszeit nicht vernachlässigbar ist.) Die Bediendauer für den Fork-Level wird ebenfalls zu 0 angenommen, kann aber leicht geändert werden. Über diese Bedieneinheit werden die Kommunikationsaktivitäten des Betriebssystems abgewickelt (vgl. Kapitel 2). Der Aktivitäten-Level ist schließlich der Bereich, in dem die untersuchten Aktivitäten mit ihren zugeteilten Prioritäten um den Prozessor konkurrieren. Dabei können die Prioritäten in diesem Bereich unterbrechend (up-System) oder nichtunterbrechend sein (nup-System). Dies bezieht sich aber immer auf den Aktivitäten-Level.
- Die Eingangs- und Ausgangsmultiplexer der Steuergerätemodule sind gleichzeitig auch Teile des Busmoduls. Im Ausgangsmultiplexer der Steuergerätemodule findet die Auswahl der sendewilligen Aktivität mit der höchsten Priorität statt, die dann dem Eingangsmultiplexer des Busmoduls angeboten wird. Dieser wählt wiederum aus allen ihm angebotenen Aktivitäten diejenige mit der höchsten Priorität aus und bearbeitet sie mit der Busbedieneinheit.
- Über die Generatoren werden Aktivitäten initiiert, deren zeitlicher Abstand einer Verteilungsfunktion gehorcht, wobei der minimale Wert dieser Funktion der Periodendauer eines Dienstes entspricht. Sobald die Aktivität in die Warteschlange der Bedieneinheit eingetragen wurde, wird ein Zeitmeßgerät gestartet. Nach vollständiger Bearbeitung (up- oder nup-Strategie) durchläuft sie die Fork-Level-Bedieneinheit, in der die Kommunikationsverbindung umgesetzt wird. Danach kann sie in den Ausgangsmultiplexer des Steuergerätemoduls übertragen werden, in dem sie auf die Übertragung durch den Bus warten muß. Wurde die Aktivität durch die Busbedieneinheit bearbeitet, dann stellt der Ausgangsmultiplexer des Busmoduls die Aktivität den Eingangsmultiplexern der Steuergerätemodule zur Verfügung. Diese melden ihrerseits das Eintreffen einer neuen Aktivität durch das Auslösen eines Interrupts. Ist ein Steuergerätemodul ein gültiger Empfänger für eine Aktivität, dann wird diese, nach Durchlaufen des Kommunikationssubsystems (auf Fork-Level), in die Warteschlange der Bedieneinheit gestellt, welche die Senke und somit die letzte Aktivität des Dienstes repräsentiert. Ist diese Aktivität bearbeitet, dann wird das Zeitmeßgerät gestoppt und das ermittelte Ergebnis zur statistischen Auswertung weitergegeben.
- Die Nachrichtensenken werden entweder ebenfalls als Generatoren modelliert („aktive Senken“), da sie ankommende Botschaften zu bestimmten Zeitpunkten aus den Puffer-



M_A Ausgangsmultiplexer

M_E Eingangsmultiplexer

IR Interrupt-Level (immer unterbrechend)

FL Fork-Level (gegenüber A immer unterbrechend)

A Aktivitäten-Level (bezüglich A entweder unterbrechbar oder nichtunterbrechbar)

Für IR und FL kann die Bearbeitungszeit zu 0 gesetzt werden

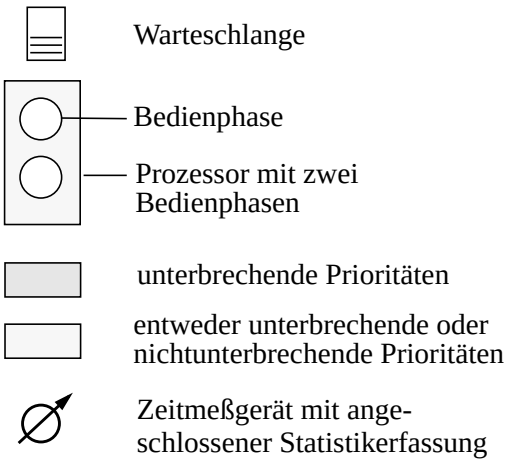


Bild A-10: Warteschlangenmodell des 3-Dienste Szenarios aus Kapitel 5

speichern holen (Polling). Die Alternative dazu sind passive Senken, die eine Callback-Routine in einem Pufferspeicher des Kommunikationssubsystems installieren, die dann aufgerufen wird, sobald ein neues Datum in den Speicher eingeschrieben wurde.

Die vollständige Funktionsweise der Simulation kann hier nur sehr kurz gefaßt wiedergegeben werden, da eine ausführliche Diskussion den Rahmen dieser Arbeit sprengen würde. Für ausführlichere Information sei deshalb nochmals auf die am Anfang von Kapitel 5 zitierte Literatur verwiesen.

A4 Notation für das objektorientierte Design

Um die Konzepte und die Realisierung des Werkzeugs in Kapitel 7 zu verdeutlichen, wurden Klassen- und Objektdiagramme verwendet. Die Darstellung orientiert sich dabei an der Notation nach Booch [15]. Die kurz vor der Veröffentlichung stehende Unified Modelling Language (UML), welche die Methoden und Notationen von G. Booch, J. Rumbaugh und I. Jacobson zusammenführen soll, wurde noch nicht verwendet, da sie zum Zeitpunkt dieser Arbeit noch im Diskussionsstadium war (Version 0.91). In der Darstellung wurden allerdings nur Elemente verwendet, die auch in der UML wiederzufinden sein werden. So wurde z.B. auf die „uses“-Beziehung der Booch-Notation verzichtet. Eine Diskussion der Methodik findet sich beispielsweise in [15] oder [41].

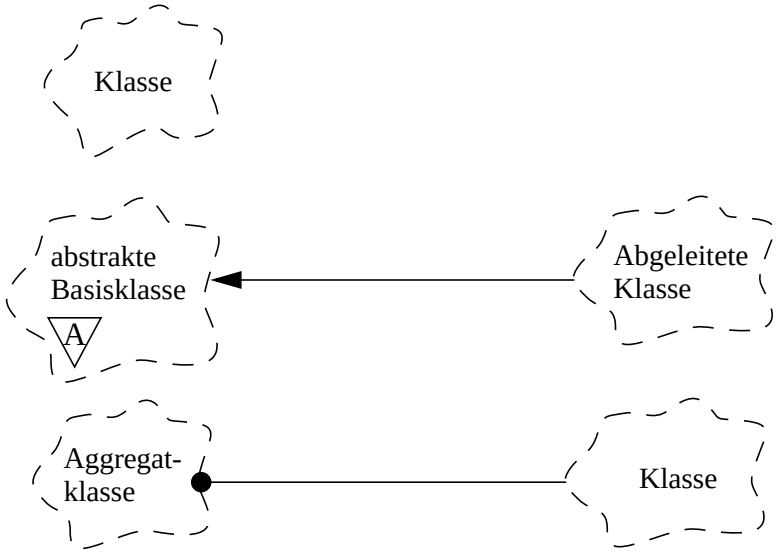


Bild A-11: Klassendiagramme

Klassendiagramme beschreiben die Beziehungen von Klassen untereinander. Klassen werden als gestrichelte Wolken, die den Klassennamen enthalten, dargestellt. Beziehungen werden durch Verbindungen zwischen diesen Klassen dargestellt, die durch sog. „Adornments“ genauer spezifiziert werden (siehe Bild A-11). Die wichtigsten Beziehungen zwischen den Klassen sind die Vererbung (oder Generalisierung, je nach Standpunkt) und das Enthalten von Instanzen einer anderen Klasse („Aggregation“). Die Vererbung wird durch einen Pfeil dargestellt, der von der abgeleiteten, spezialisierten Klasse zur allgemeineren Basisklasse zeigt. Er kennzeichnet eine in Pfeilrichtung bestehende „ist eine“-Beziehung. Die Enthalten-Beziehung („hat“-Beziehung) wird durch einen ausgefüllten Kreis an der Klasse dargestellt, welche die andere Klasse beinhaltet. Eine weitere Verfeinerung der Darstellung bilden Adornments innerhalb der Klassen. In dieser Arbeit wird, zur Kennzeichnung abstrakter Basisklassen (von diesen Klassen gibt es in der Regel keine eigenen Instanzen) ein  verwendet.

Durch Interaktionsdiagramme (auch „Message Sequence Charts“) kann das Zusammenarbeiten von Objekten verdeutlicht werden. Bild A-12 zeigt einen Ausschnitt eines Interaktionsdiagramms, bei dem der Austausch von Botschaften zwischen den Objekten über der Zeit aufgetragen wird. Die Namen der betrachteten Objekte stehen dazu über einer gestrichelten Zeitachse. Der Austausch von Botschaften oder Ereignissen zwischen den Objekten wird durch Pfeile dargestellt, die außerdem den Namen der Botschaft (den Namen der aufgerufenen Methode) tragen. Rechtecke längs der Zeitachse geben zusätzlich an, wie der Kontrollfluß

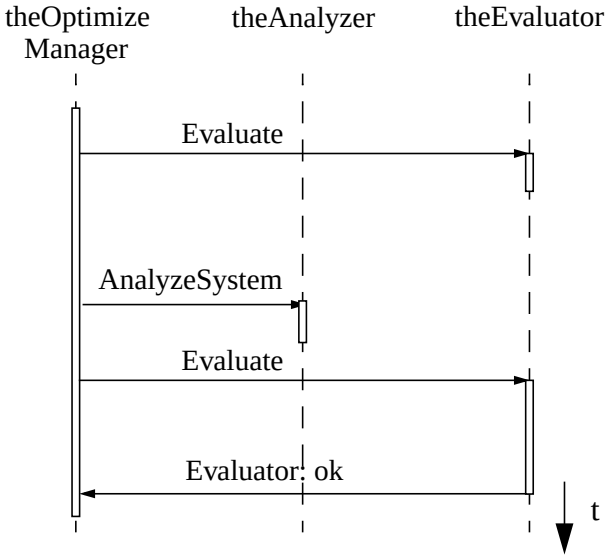


Bild A-12: Interaktionsdiagramm

innerhalb des Systems aussieht. Rückgabewerte von Methodenaufrufen werden deshalb nicht explizit gezeichnet, es sei denn, es müssen besondere Verhältnisse angegeben werden. Sonst entspricht das Ende eines Kontrollflußrechtecks dem Zeitpunkt des Verlassens der aufgerufenen Methode.