Lastbezogene Rekonfigurierung von komponentenbasierten Verteilten Systemen mit benutzergesteuerten Anwendungen

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

vorgelegt von
Volker Feil
geb. in Eberbach

Hauptberichter: Prof. Dr.-Ing. Dr. h. c. mult. P. J. Kühn
Mitberichter: Prof. Dr.-Ing. Dr. h. c. P. Göhner

Tag der mündlichen Prüfung: 14.04.2005

Institut für Kommunikationsnetze und Rechnersysteme
der Universität Stuttgart
2005

Abstract

This report investigates the load-aware reconfiguration of component-based distributed systems with user-controlled applications. At first, the structure of the report is described. Then, the two main topics of this work are presented: A new method for modeling component-based distributed systems with user-controlled applications and an evaluation of several design proposals for the load-aware dynamic reconfiguration of these distributed systems. The evaluation is based on results that are found by applying the presented modeling method.

1 Structure of the report

Chapter 1 – Introduction

This chapter emphasizes the importance of component software by considering exemplarily future in-car telematics systems. These distributed systems may contain user-controlled applications that are decomposed in software components. This report investigates mechanisms for dynamically redistributing components in order to minimize response times of user requests, and therefore to improve the system performance.

Chapter 2 – Services in distributed systems

Chapter 2 shows the realization of today's and future distributed systems by considering three typical domains: The enterprise domain, the home entertainment domain, and the in-car telematics domain. This includes the presentation of relevant middleware technologies and component software technologies. Furthermore, the significance of the client/server cooperation model for these technologies is discussed.

Chapter 3 – Configuration of component-based distributed systems

Important research contributions that deal with the configuration management of component-based distributed systems are presented. To manage the configuration of component-based distributed systems comprises to handle the life-cycle as well as the interactions of components.

Chapter 4 – Modeling configurations

This chapter presents a new method for modeling the configuration of component-based distributed systems. The method is appropriate if a distributed system mainly contains user-con-

trolled applications that are composed of distributed software components. The method is based upon closed queueing network theories, and therefore allows to evaluate the performance of distributed systems. The feasibility of the modeling method is shown by presenting measurement results regarding the process scheduling of actual applications. For instance, the process scheduling for a purchasable WWW browser is analyzed.

Chapter 5 – Load-aware reconfiguration

Chapter 5 presents mechanisms for the reconfiguration of component-based distributed systems. They are based on load distribution algorithms. Mechanisms that are based on standard algorithms (e.g. round robin) are compared with special mechanisms that dynamically apply the presented modeling method in order to achieve load awareness.

Chapter 6 – Reconfiguration as part of the resource management

This chapter considers distributed systems that comprise a resource management for the handling of A/V streams with a constant bit rate. Such a management allows the reservation of bus system resources as well as processor resources. Not reserved resources are free for use by software components that require only best effort quality of service. In such systems dynamic reconfiguration by redistributing these software components allows to improve the performance of the whole system.

2 An exemplary study: Component software for the in-car telematics domain

The term *component software* describes a software paradigm with increasing importance during the last years. A component software technology contains mechanisms for composing software by using purchasable single components. Because of these mechanisms the reusability of software will be improved.

Especially for the in-car telematics domain, a change to the component software paradigm is an interesting challenge. Nowadays, suppliers deliver their products as functions containing hardware devices to the vehicle manufacturers. They act as system integrators and connect the devices via optical ring busses. The implementations of the functions are generally not reusable, if the supplier of a hardware device is changed or if a function has to be provided in a new device by another supplier. Because of this lack of reusability vehicle manufacturers seek for a finer granularity of future products. The aim is that the functions themselves will be provided as the suppliers' products. In this scenario, a function will be realized as a reusable software component.

Furthermore, during a vehicle's life time integrating additional functions or upgrading existing functions will become simpler. This aspect addresses the problem, that a vehicle's product lifecycle is much longer than an average software innovation cycle. Till now, the novelty of telematics functions inside a vehicle depends essentially on the vehicle's age because a func-

tion can only be additionally integrated or upgraded expensively in a workshop. Looking at the in-car telematics domain the usage of component software is still in the beginning. For instance, some of today's luxury vehicles contain telematics systems with Java components.

3 Modeling configurations of component-based distributed systems with user-controlled applications

The report implies the following preconditions: Software components are located on the devices of a distributed system. The components interact without respecting device boundaries. An application consists of distributed components. By using a so called user interface a user triggers actions that influence the control flow of the application. Software technologies are available that enable the dynamic component migration to other devices without human administration. This means that a running software is responsible for configuring the distributed system dynamically.

The report analyses how the performance of a component-based distributed system can be improved by reconfiguring the system. The reconfiguring work is realized by the migration of components from their source devices to their target devices.

Control flow of decomposed applications

The presented model assumes that a user actively triggers the usage of a software service that is provided by an application (for example by clicking a button of a window). The model considers only decomposed applications and assumes that components can only play two roles during an inter-component cooperation: they appear either as client or as server. A client-component surrenders its control to a server-component by using its service. This means, that the client is blocked till it gets back the control from the server. During computing a result for its client a server-component itself may take over the client role for several times.

During the application's computation that is caused after a user's action, a blocking of the user is often necessary in order to avoid data inconsistencies. Therefore, in this report a so called critical actions queue for realizing the blocking of the user is proposed. This queue consists

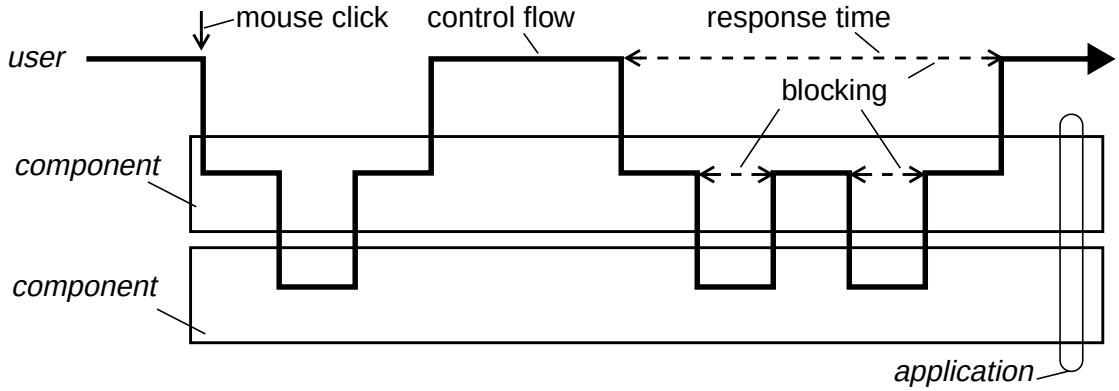


Figure 1: Control flow of a decomposed application

only of user actions that can cause inconsistencies if they trigger a simultaneous processing. Using this queue only one of these actions is handled at each time. The report validates the significance for the actual usage of critical actions queues by analyzing measurements that regard the process behavior of actual applications (e.g. a WWW browser). One important result is: The assumption that an application comprises only one relevant control flow is appropriate for many actual user-controlled applications. Figure 1 depicts a possible control flow of a decomposed application.

Modeling distributed systems as closed queuing networks

By modeling distributed systems as closed queuing networks (CQNs) their performance can be evaluated because analytical computing approaches are applicable. For instance, the computation of so called product form solutions in order to get system state probabilities is well-known.

In the model an application is represented by a chain inside a CQN. The components are represented as chain members. Because of the assignment of components to run-time environments, there is a corresponding assignment of the chain members to single servers of the CQN that represent the run-time environments. It is assumed, that the servers work with a processor sharing strategy. The user is modelled as an infinite server. His or her thinking time gap is modelled as service time. Only one request that represents the control flow circulates inside the chain. The circulation models the delegation chain given by server-components that take over the client role during their computing. In the client role they request their servers and wait for the response by blocking.

Activity graphs and their transformation in chains

On the left side of Figure 2 a directed and coherent graph is shown which represents an application. In this report this type of graph is named activity graph. One node of an activity graph represents the user. The other nodes represent the components of the application. The example in Figure 2 shows a very simple graph that represents an application with only three components A, B, and C. The nodes are connected by directed links. Each link corresponds to an interface relationship between two components. For instance, the link from node A to node B represents an interface that is imported from the client-component A and exported from the server-component B. Each link has a weight. The weight of the link from A to B represents the mean service time of server B that is responsible for client A, and the mean interarrival time of the calls from client A to server B.

The report presents an algorithm that transforms an activity graph to a chain of a CQN. The basic idea is to traverse an activity graph – with the help of well-known algorithms like depth first search (DFS) or breadth first search (BFS) – and therefore to consider each link only once. A link corresponds to a chain member of the application's chain in the QCN. The order of the

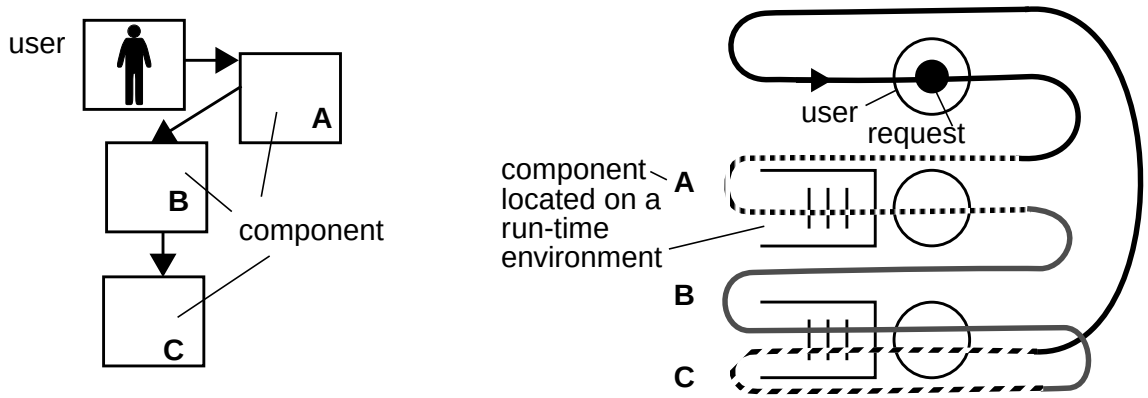


Figure 2: Control flow of a decomposed application

chain members does not matter. The location of any chain member in the CQN is unambiguously given by the single server that represents the run-time environment of the corresponding server-component. The chain is closed by a chain member that represents the user. On the right side of Figure 2 a chain is depicted that corresponds to the activity graph on the left side. For this example, the components B and C are located on the same run-time environment.

4 Load-aware reconfiguration

Based on the presented model, this report investigates reconfiguration mechanisms that utilize the output of load distribution algorithms. The aim is to distribute load in a distributed system in order to improve its over-all performance. To improve the performance could mean, e. g., to minimize the summation of all application response times. The report demonstrates that reconfiguration mechanisms that are based on well-known standard load distribution strategies are feasible. The first considered standard strategy is to assign complete applications to run-time environments by applying a round robin algorithm. The second considered standard strategy is to assign a complete application to the run-time environment with minimum utilization. Furthermore, the report presents a new mechanism. It dynamically reconfigures the system by using the presented modeling method in order to find a better configuration through the results of appropriate performance analyses. It is described briefly in the following paragraph.

Reconfiguration mechanism that applies the presented modeling method

At first, an application is completely assigned to a certain run-time environment by applying the round robin algorithm.

During the system’s run-time measurements are performed that allow to approximately calculate the mean service rate and the mean interarrival time of server requests for each client/server interface. A central configuration management component gathers these mean values in order to build or to update the activity graph.

If an application is inactive because the user is in a thinking phase, an application’s component is randomly chosen as a candidate for a rearrangement. In principle, by assigning the chosen

component to other run-time environments, new configurations are possible. With the help of the activity graphs and by applying a heuristic that is based on the analytical mean value analysis (MVA), the performance of any configuration that is possible by rearranging the chosen component is evaluated. This means, that the summation of all application response times is computed approximately by applying the heuristic. The configuration with the minimum summation value is determined as the best. Therefore, a component migration is initiated in order to realize this configuration.

Effectiveness of the considered reconfiguration mechanisms

The report presents results of a study that evaluates the effectiveness of the considered reconfiguration mechanisms. The results are found by analytical computations and by simulations. Two main results can be expressed qualitatively:

- For a distributed system with low computing power, the employment of more simple mechanisms is appropriate.
  The performance of a reconfiguration mechanism is limited by the computation overhead that is caused by measuring, refining, and collecting system state information. For instance, the mechanism that applies the presented modeling method requires activity graphs. For their creation the knowledge about the mean service rates and mean interarrival times of the requests for all client/server interfaces is necessary. To keep the information up-to-date, a continuous and computation-intensive distributed measuring, distributed refining, and central collecting of these mean values is required. For a distributed system with low computing power the utilization of a simple round robin-based mechanism can be more worthwhile than using such a complex mechanism.

- For a distributed system with small homogeneity, the employment of more complex mechanisms is appropriate.
  A mechanism that is based on round robin does not consider the current load of the system. If the load is inhomogeneously distributed the mere utilization of round robin leads to poor results. In this case, a mechanism that considers the utilization of the run-time environments – e. g. in order to find the new location of an application that must be installed – works better than such a simple mechanism. However, it is more expensive because of the computional overhead that is caused by measuring the utilization. A utilization based mechanism considers the current load distribution, but it does not consider the future load distribution that is caused by the reconfiguration itself. On the contrary, the mechanism that applies the presented modeling method considers this new load distribution that results out of the reconfiguration. Therefore, this mechanism is able to handle load inhomogeneities at best.

5 Future Scenario: Applying load-aware reconfiguration within an A/V streams handling configuration management

In the context of this work a new architecture for a configuration management is proposed. It is called DANA (*Distributed Systems Architecture for Networks in Automotive Environments*), and it is developed for future distributed systems of the in-car telematics domain. The configuration management of DANA allows the handling of A/V streams with a constant bit rate as well as the handling of Java components that interact with the help of well-known middleware technologies.

To enable the exchange of A/V streams with constant bit rate, resource reservations on the level of bus systems as well as of devices (in order to get certain processor capacities) are necessary. Within DANA's configuration management the embedding of a reconfiguration management can be worthwhile. Then, it is responsible for the redistribution of Java components which can be necessary because of load fluctuations caused by the dynamic resource reservations for A/V streams.

6 Conclusion

These are the essential contributions of the report:

- The report presents and validates a new method for the performance modeling of component-based distributed systems with user controlled applications.
  An application is represented by a chain within a closed product form queuing network. The software components of the application corresponds to the chain members.

- It presents a new load-aware dynamic reconfiguration mechanism that is based on the presented modeling method.
  Its responsibility is to evaluate the performance of possible configurations that can be arranged by simple changes of the current configuration. Furthermore, it has to choose the best solution as the future configuration. Finally, it has to initiate the reconfiguration by causing the migration of only one component.

- It evaluates this new reconfiguration mechanism by comparing it with mechanisms that apply well-known load distribution strategies.
  The evaluation respects the usage of the reconfiguration mechanisms in component-based systems with user-controlled applications.

- It presents an architecture for a configuration management.
  The configuration management allows the handling of A/V streams with constant bit rate as well as the handling of Java components. Redistribution of Java components with the help of the proposed reconfiguration mechanism can be worthwhile if resource reservations for A/V streams occur.

Inhaltsverzeichnis

Abkürzungen

AC	Admission Control
ADL	Architecture Description Language
ATM	Asynchronous Transfer Mode
A/V	Audio und Video
AWT	Abstract Windowing Toolkit
BCMP	Theorem von Basket, Chandy, Muntz und Palacios
BFS	Breadth First Search
CAN	Controller Area Network
CCM	CORBA Component Model
CD	Compact Disk
CDR	Common Data Representation
CEI	Channel Endpoint Identifier
COM	Component Object Model
CORBA	Common ORB Architecture
COSIMA	Component System Information and Management Architecture
DANA	Distributed Systems Architecture for Networks in Automotive Environments
DCM	Device Control Module
DCOM	Distributed COM
DFS	Depth First Search
DMA	Direct Memory Access
DVD	Digital Versatile Disk
D2B	Digital Domestic Bus
EDF	Earliest Deadline First
EJB	Enterprise Java Beans
EMA	Exponential Moving Average
FCFS	First Come First Served
FIFO	First In First Out
GC	Garbage Collection
GPWN	Geschlossenes Produktformwarteschlangennetz
GUI	Graphical User Interface
GWN	Geschlossenes Warteschlangennetz
HAVi	Home Audio Video Interoperability
HDTV	High Definition Television
HTTP	Hypertext Transfer Protocol
IDB	Intelligent Transportation System Data Bus

IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IIOP	Internet Inter ORB Protocol
IP	Internet Protocol
IS	Inquiry Scan
JIT	Just In Time
JMF	Java Media Framework
JNI	Java Native Interface
JVM	Java Virtual Machine
LBANC	Local Balance Algorithm for Normalizing Constants
LCFS-PR	Last Come First Served with Preemptive Resume
LCM	Local Channel Manager
LGS	Lineares Gleichungssystem
LS	Lookup Service
MCH	Modifizierte Core-Heuristik
MOST	Media Oriented Systems Transport
MPEG	Moving Picture Expert Group
MTS	Microsoft Transaction Server
MVA	Mean Value Analysis
MVC	Model View Controller
OMG	Object Management Group
ORB	Object Request Broker
OSGi	Open Services Gateway Initiative
OSI	Open Systems Interconnection
PC	Personal Computer
POA	Portable Object Adapter
PS	Processor Sharing
QoS	Quality of Service
RCM	Range Channel Manager
RMI	Remote Method Invokation
RPC	Remote Procedure Call
RR	Round Robin
RSVP	Resource Reservation Protocol
RTP	Real-Time Transport Protocol
SBM	Subnet Bandwidth Manager
SDH	Synchronous Digital Hierarchy
SDP	Service Discovery Protocol
SLM	Salutation Manager
SLP	Service Location Protocol
SOAP	Simple Object Access Protocol
SPN	Stochastisches Petri-Netz
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol

TLA	Top Level Architecture
TM	Transport Manager
TTP	Time Triggered Protocol
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
UML	Unified Modeling Language
WSDL	Web Service Description Language
WWW	World Wide Web
W3C	WWW Consortium
XML	Extensible Markup Language

Formelzeichen

a_i	Anzahl der Anwendungen, die der Ablaufumgebung *i* zugeordnet sind
c	Konstante
$\underline{C}$	zur Berechnung der SPN-Produktformlösung benötigter Vektor
$d_{irs}(\underline{k})$	Parameter beim Linearizer-Algorithmus
$e(\underline{k}, r)$	Funktion, die das *r*-te Element des Vektors $\underline{k}$ zurückliefert
$f_{ir}(\underline{k})$	Quotient der mittleren Anzahl von Anforderungen einer Klasse *r* in einem Knoten *i* zur Gesamtanzahl von Anforderungen der Klasse *r*
G	Normalisierungskonstante
$G(k)$	Normalisierungskonstante bei *k* Anforderungen im Netz
$G(\underline{k})$	Normalisierungskonstante bei konstanter Anzahl von Anforderungen je Klasse im Netz, ausgedrückt durch den Vektor $\underline{k}$.
i, j, q, r, s, x, z	Variablen
I	Anzahl der Iterationen beim Core-Algorithmus
$\underline{I}$	Inzidenzmatrix eines Petri-Netzes
k	Anzahl von Anforderungen bzw. Marken im System
k_i	Anzahl von Anforderungen bzw. Marken im Knoten *i* (Bediensystem *i* bzw. Platz p_i)
k_{ir}	Anzahl von Anforderungen der Klasse *r* im Knoten *i*
$\bar{k}$	mittlere Anzahl von Anforderungen bzw. Marken in einem System
$\bar{k}_i(\underline{k})$	mittlere Anzahl von Anforderungen im Knoten *i* bei gegebenem $\underline{k}$
$\bar{k}_{ir}(\underline{k})$	mittlere Anzahl von Anforderungen der (Teil-)Kette *r* im Knoten *i* bei gegebenem $\underline{k}$
$\underline{k}$	Vektor, der die Anzahl der Anforderungen im Netz für jede Klasse angibt
$\underline{M}_e(i)$	Eingabemenge beim Feuern der Transition *i* im Petri-Netz
$\underline{M}_a(i)$	Ausgabemenge beim Feuern der Transition *i* im Petri-Netz
n_i	*n*-fache Kapazität einer Ablaufumgebung *i* gegenüber einer Referenzablaufumgebung
N	Anzahl der Bediensysteme im Netz
N_p	Anzahl der Plätze (Stellen) im Petri-Netz
N_r	Anzahl der Bediensysteme, in denen sich mindestens eine Anforderung der Klasse *r* befinden kann
N_t	Anzahl der Transitionen im Petri-Netz
p_{ij}	Wahrscheinlichkeit, dass eine im Knoten *i* fertig bediente Anforderung zum Knoten j gelangt
$p_{ir,\,js}$	Wahrscheinlichkeit, dass eine Anforderung der Klasse *r* nach Fertigbedienung im Knoten *i* als neue Anforderung der Klasse *s* zum Knoten *j* gelangt

$p(k)$	Wahrscheinlichkeit, dass sich im untersuchten Bereich k Anforderungen befinden
$p_i(k_i)$	Wahrscheinlichkeit, dass sich im i-ten Knoten k_i Anforderungen befinden
$p(\underline{S})$	Wahrscheinlichkeit, dass sich das System im Zustand $\underline{S}$ befindet
$P(x \leq X)$	Wahrscheinlichkeitsverteilungsfunktion
q_r	Fairness-Faktor für Anwendung r
Q	Summe aller Fairness-Faktoren
R	Anzahl von Anforderungsklassen im System
$\underline{S}$	Vektor, der einen Netzzustand (Systemzustand) beschreibt
$\underline{S}_i$	Vektor, der einen Zustand des Knotens i im Warteschlangennetz beschreibt
$\underline{S}^n$	Vektor, der einen Netzzustand (Systemzustand) n beschreibt
t	Zeitparameter
t_{in}	n-ter Wert in einer Reihe, der dem Abstand zwischen zwei Ankünften der Anforderungen A_n und A_{n+1} entspricht
t_{sn}	n-ter Wert in einer Reihe, der der Bedienzeit einer Anforderung A_n entspricht
$\dot{t}$	mittlere Aufenthaltsdauer (Durchlaufdauer)
$\dot{t}_{ir}$	mittlere Aufenthaltsdauer einer Anforderung der Klasse r im Knoten i
$\dot{t}_{ir}(\underline{k})$	mittlere Aufenthaltsdauer einer Anforderung der Klasse r im Knoten i bei gegebenem $\underline{k}$
$\dot{t}_R$	Antwortzeit einer Anwendung bei R existierenden Anwendungen
v_i	Besuchskoeffizient von Knoten i
v_{ir}	Besuchskoeffizient von Knoten i für die Klasse r
W	Anzahl der Schnittstellenbeziehungen im Verteilten System
$X(t)$	Zufallsvariable, die vom Parameter t abhängt
y_i	Variable bei der Berechnung der SPN-Produktformlösung
z_i	Zahl z für Ablaufumgebung i, die für das d'Hondtsche Höchstzahlenverfahren verwendet wird
Z	Anzahl aller Teilketten im Warteschlangennetz
$\Delta(i, z)$	Funktion, die nur dann 1 ergibt, wenn Teilkette z durch Knoten i gelegt ist, und die ansonsten 0 ergibt
$\varepsilon(j)$	Funktion zur Ermittlung einer Transition i bei gegebener Transition j, so dass $\underline{M}_e(i) = \underline{M}_a(j)$ gilt
$\zeta(z)$	Funktion, die bei Übergabe der Teilkette z die Kette ermittelt, zu der z gehört
λ	mittlere Ankunftsrate (Durchsatz bei verlustfreien, stabilen Systemen)
λ_i	mittlere Ankunftsrate im Knoten i eines Warteschlangennetzes
λ_{ir}	mittlere Ankunftsrate für Anforderungen der Klasse r im Knoten i eines Warteschlangennetzes
$\lambda_i(\underline{S})$	zustandsabhängige mittlere Ankunftsrate von Anforderungen im Knoten i
$\lambda(\underline{S}^n, \underline{S}^m)$	Übergangsrate vom Zustand $\underline{S}^n$ in den Zustand $\underline{S}^m$
μ	mittlere Bedienrate
μ'	mittlere Bedienrate in einem beliebigen Knoten für eine beliebige Anwendung, wenn man alle ihre Komponenten zusammengefasst als eine betrachtet
μ_i	mittlere Bedienrate im Knoten i

μ'_r	mittlere Bedienrate in einem beliebigen Knoten für die Anwendung *r*, wenn man alle ihre Komponenten zusammengefasst als eine betrachtet
μ_{ir}	mittlere Bedienrate im Knoten *i* einer Anforderung der Kette bzw. Teilkette *r*
μ'_{ir}	mittlere Bedienraten für die Anwendung *r*, wenn man alle ihre Komponenten, die auf Knoten *i* lokalisiert sind, zusammengefasst als eine betrachtet
ρ	Auslastung
ρ_i	Auslastung im Knoten *i*
τ	mittlere Dauer einer Benutzerdenkpause
τ_r	mittlere Dauer der Benutzerdenkpause für die Anwendung *r*
$\Phi(\underline{S})$	zustandsabhängiger Parameter bei der Produktformlösung nach Kelly
$\Phi_i(k_i)$	Verhältnis zwischen der tatsächlichen Übergangsrate einer Anforderung vom Knoten *i* zu einem beliebig anderen Knoten bei k_i Anforderungen im Knoten *i* und der normierten Bedienrate (μ_i / ν_i)
$\Phi_{ir}(\underline{S}_i)$	Verhältnis zwischen der tatsächlichen Übergangsrate einer Anforderung der Klasse *r* vom Knoten *i* zu einem beliebig anderen Knoten beim Zustand $\underline{S}_i$ im Knoten *i* und der normierten Bedienrate (μ_{ir} / ν_{ir})
$\Psi(\underline{S})$	zustandsabhängiger Parameter zur Bestimmung der Schaltrate von Transitionen in Stochastischen Petri-Netzen
$\Re$	Anzahl der Rechenschritte bei einem Algorithmus
$\underline{0}$	Nullvektor
$\underline{1}_k$	Vektor, dessen *k*-tes Element gleich 1 ist und dessen übrige Elemente gleich 0 sind

Kapitel 1

Einleitung

1.1 Umfeld und Motivation

Der Begriff Komponenten-Software beschreibt ein Paradigma in der Software-Technik, das in den vergangenen Jahren stark an Bedeutung zunahm und nun in vielen Bereichen entweder schon fest etabliert ist oder vor einer möglichen Einführung steht. Eine Komponenten-Software-Technologie beinhaltet Mechanismen zum Zusammenbau von Software aus einzelnen Komponenten. Auf diese Weise wird eine Verbesserung der Wiederverwendbarkeit von Software angestrebt. In dieser Arbeit wird speziell der Einzug des Komponenten-Software-Paradigmas in den Unternehmensbereich, den Heimbereich und den Fahrzeugtelematikbereich betrachtet.

Insbesondere für die Fahrzeugtelematik ist ein Wechsel auf das Komponenten-Software-Paradigma eine interessante Herausforderung. Heutzutage liefern Zulieferer dem Fahrzeughersteller ihre Produkte in Form von einzelnen, mehrere Funktionen beinhaltenden Geräten. Die Realisierungen der Funktionen sind i. A. nicht mehr wiederverwendbar, wenn für ein Gerät der Zulieferer gewechselt wird oder wenn sie innerhalb eines neuen Gerätes, für das ein anderer Zulieferer zuständig ist, bereitgestellt werden sollen. In Zukunft wird deshalb eine feinere Produktgranularität angestrebt, so dass die Funktionen einzeln vom Zulieferer als Produkt bereitgestellt und vom Fahrzeughersteller ins System integriert werden können. Diese Funktionen sollen in Form von wiederverwendbaren Software-Komponenten realisiert werden.

Im Vergleich zu heute ermöglicht der Einsatz einer Komponenten-Software-Technologie zudem ein einfacheres „Einbauen“ von Software ins Fahrzeug während seiner Lebenszeit. Hierdurch ist ein Weg vorgezeichnet, wie neue Fahrzeugtelematikentwicklungen zukünftig ins Fahrzeug gebracht werden können. Bisher war die Aktualität der Telematikfunktionen im Fahrzeug im Wesentlichen abhängig vom Fahrzeugalter, da eine Aktualisierung nur aufwändig in der Werkstatt vollzogen werden konnte. Die Fahrzeuglebenszeit ist jedoch viel länger als die

Dauer der Software-Innovationszyklen.

Der Einsatz von Komponenten-Software befindet sich im Fahrzeugtelematikbereich in den Anfängen, wie man an heutigen Serienfahrzeugen der Luxus- und Komfort-Klasse, die vereinzelt Telematiksysteme mit Java-Komponenten beinhalten, sehen kann.

Neben dem Aufstreben der Komponenten-Software ist in den genannten Bereichen ein weiterer Trend erkennbar. Die Endgeräte werden zunehmend mehr miteinander vernetzt. Es entstehen Verteilte Systeme, die zwar lokal auf ein Unternehmen, auf das Heim oder auf ein Fahrzeug begrenzt sind, jedoch Zugang zu einer globalen Netzinfrastruktur besitzen.

In dieser Arbeit wird ein komponentenbasiertes Verteiltes System als ein lokal begrenzter, vernetzter Bereich angesehen, auf dessen Endgeräte sich Software-Komponenten befinden, die geräteübergreifend miteinander kooperieren. Eine Anwendung setzt sich i. A. aus mehreren Komponenten zusammen. Sie wird vom Benutzer gesteuert, wenn er über eine Benutzerschnittstelle Aktionen tätigt, die den Ablauf der Anwendung beeinflussen.

Es wird in der vorliegenden Arbeit davon ausgegangen, dass Software-Technologien in den Verteilten Systemen zur Verfügung stehen, die ein Verschieben von Software-Komponenten auf andere Endgeräte zur Systemlaufzeit transparent in dem Sinne ermöglichen, dass hierfür keine menschlichen, administrativen Eingriffe vorgenommen werden müssen. Dies bedeutet, dass durch derartige Technologien eine Software bereitgestellt werden kann, die die Konfiguration des Verteilten Systems dynamisch verwaltet.

Die vorliegende Arbeit befasst sich mit der Frage, wie der Einsatz von Verfahren, die eine lastbezogene Rekonfigurierung durch ein Umordnen von Komponenten auf andere Endgeräte während der Systemlaufzeit vornehmen, die Systemleistung verbessern kann.

1.2 Übersicht über die Arbeit

Folgende Punkte werden betrachtet:

- Es wird untersucht, wie heutige Verteilte Systeme gestaltet sind und zukünftige Systeme realisiert werden. Hierzu werden die heute relevanten Middleware- und Komponenten-Software-Technologien betrachtet. Zudem werden wichtige Ergebnisse von Forschungsbeiträgen, die sich mit Konfigurationsverwaltungen für Verteilte Systeme auseinandersetzen, zusammengefasst dargestellt.
- Es wird eine Methode präsentiert, mit der die Konfiguration eines Verteilten Systems mit sich aus verteilten Komponenten zusammensetzenden, benutzergesteuerten Anwendungen so modelliert werden kann, dass eine Bewertung der Systemleistung möglich ist.

- Es wird die Anwendbarkeit dieser Modellierungsmethode nachgewiesen, indem Ergebnisse von Messungen über existierende, benutzergesteuerte Anwendungen ausgewertet werden.
- Es werden mit Hilfe der Modellierungsmethode verschiedene Lastverteilungsverfahren, die lastbezogene Rekonfigurierungen vornehmen, untersucht und bewertet.
- Es wird gezeigt, bei welchen komponentenbasierten Verteilten Systemen der Einsatz von lastbezogenen Rekonfigurierungsverfahren besonders lohnenswert ist.

Die Arbeit ist folgendermaßen aufgebaut:

Im 2. Kapitel werden die für diese Arbeit notwendigen Grundlagen auf dem Gebiet der Komponenten-Software und auf dem Gebiet der Verteilten Systeme dargestellt. Es wird auf die Bedeutung des Internet-Protokolls für die heute im Unternehmens- und Heimbereich vorkommenden Software-Technologien eingegangen. Anschließend werden die heute relevanten Middleware- und Komponenten-Software-Technologien vorgestellt und klassifiziert. Schließlich wird die Realisierung heutiger Verteilter Systeme für die Fahrzeugtelematik betrachtet.

Das 3. Kapitel befasst sich mit dem Begriff der Konfiguration. In einem ersten Teil werden zunächst aktuelle Forschungsbeiträge vorgestellt, die sich u a. mit der Vermeidung von Inkonsistenzen bei der Rekonfigurierung eines komponentenbasierten Verteilten Systems auseinandersetzen. Es werden anschließend die Aufgaben einer zur Systemlaufzeit arbeitenden Konfigurationsverwaltung erläutert. In einem zweiten Teil werden Konfigurationen von Verteilten Systemen mit benutzergesteuerten Anwendungen, die in Komponenten zerlegt sind, betrachtet. Es wird gezeigt, dass in manchen Fällen die Systemleistung verbessert werden kann, wenn eine Anwendung in Komponenten zerlegt wird und diese im System verteilt werden. Schließlich wird betrachtet, wie verteilte Komponenten nach dem Client/Server-Modell miteinander kooperieren.

Die Betrachtungen über die Client/Server-Kooperationen führen zu einer Methode zur Modellierung von Konfigurationen Verteilter Systeme, die in Komponenten zerlegte Anwendungen beinhalten. Im 4. Kapitel wird diese Methode, mit der eine Bewertung der Systemleistung möglich ist, präsentiert. Eine Anwendung wird hierbei durch eine in ein Warteschlangennetz gelegte Kette, in der eine Anforderung zirkuliert, modelliert. Schließlich werden die bei einer Umordnung von Komponenten auftretenden Synchronisationsvorgänge mit Hilfe der Petri-Netzmodellierung betrachtet, so dass auch dieser Aspekt in die Bewertung der Systemleistung einfließen kann.

Im 5. Kapitel werden Lastverteilungsverfahren für den Einsatz bei lastbezogenen Rekonfigurierungen vorgestellt und aufgrund der im vorherigen Kapitel eingeführten Methode bewertet. Lastbezogene Rekonfigurierung ist hierbei eine Teilaufgabe einer zur Systemlaufzeit arbeitenden Konfigurationsverwaltung. Zum einen werden Standardverfahren wie z. B. Round Robin

betrachtet. Demgegenüber werden Verfahren präsentiert und bewertet, die ihre Lastverteilungsentscheidungen mit Hilfe der im 4. Kapitel vorgestellten Methode treffen.

In 6. Kapitel wird betrachtet, wie sich die Rekonfigurierungsaufgabe in eine umfassende Ressourcen-Verwaltung einordnen lässt. Es wird darauf eingegangen, wie durch die Reservierung von Ressourcen für Audio/Video-Ströme und deren Freigabe die Prozessorkapazität von Komponentenablaufumgebungen schwanken kann. Es wird dargestellt, dass der Einsatz eines Verfahrens zur lastbezogenen Rekonfigurierung für solche Szenarien lohnenswert ist.

Die Arbeit endet mit dem 7. Kapitel. Dort werden die wichtigsten Ergebnisse zusammengefasst und es wird in einem Ausblick dargestellt, dass lastbezogene Rekonfigurierung in zukünftigen Verteilten Systemen für die Fahrzeugtelematik Anwendung finden kann.

Kapitel 2

Dienste in lokal begrenzten Verteilten Systemen

In diesem Kapitel wird die Realisierung von Diensten und deren Verwaltung in heutigen Verteilten Systemen vorgestellt. Der Begriff Dienst bezeichnet die Ausführung der Funktion eines Diensterbringers für seine Dienstnutzer. Eine Anwendung erbringt einen Dienst für den Benutzer und nutzt gegebenenfalls weitere, im Verteilten System vorhandene Dienste. Dieses Prinzip setzt sich verschachtelt fort: Ein hochwertiger Dienst wird erbracht, indem niederwertigere Dienste genutzt werden.

Es werden drei verschiedene Einsatzbereiche für lokal begrenzte Verteilte Systeme vorgestellt: Verteilte Systeme im Unternehmensbereich realisieren die Vernetzung von Arbeitsplätzen und Servern in Unternehmen. Weiter wird die Vernetzung von Geräten der Heimunterhaltungselektronik und zudem die heutige Vernetzung von Geräten der Fahrzeugtelematik betrachtet. Schließlich wird ein Ausblick auf eine mögliche, zukünftige Vernetzung in der Fahrzeugtelematik gegeben.

Neben den lokal begrenzten Verteilten Systemen existieren globale Verteilte Systeme, auf die in dieser Arbeit nicht näher eingegangen wird. Als Beispiele sind vor allem das globale Internet und das klassische Telekommunikationsnetz zu nennen.

2.1 Dienste in Verteilten Systemen zur Unternehmensvernetzung

In diesem Abschnitt wird ein allgemeiner Überblick über Realisierungsformen von Diensten bei heutigen Verteilten Systemen für die Unternehmensvernetzung gegeben.

2.1.1 Anforderungen von Anwendungen und Systemrealisierung

Den Ausgangspunkt der Betrachtungen bilden die in den Unternehmensnetzen anzutreffenden Anwendungen. Im Folgenden wird eine Klassifikation der Anwendungen hinsichtlich der geforderten Dienstgüten (engl. *Quality of Service*, QoS) gegeben und gezeigt, wie diese Anforderungen bei der Realisierung von Verteilten Systemen berücksichtigt werden.

2.1.1.1 Klassifikation von Anwendungen

Die Klassifikation orientiert sich an den Betrachtungen von Wright [138]. Es findet eine Unterscheidung in Datendienste, interaktive A/V-Dienste (*Audio und Video*) und A/V-Abrufdienste statt:

- Datendienste

  Die Ressourcen-Anforderungen einer Anwendung im Verteilten System (z. B. Prozessorkapazität und Bitrate im Netz) hängen u. a. vom Umfang der zu verarbeitenden Daten ab und können enorm variieren. Im Allgemeinen dürfen bei der Verarbeitung bzw. beim Austausch von Daten keine Verluste auftreten. Verzögerungen werden toleriert, wobei bei der Anwesenheit eines menschlichen Benutzers eine Toleranzgrenze beachtet werden muss.

- Interaktive A/V-Dienste

  Die Anforderungen an interaktive Sprachübertragung sind bezüglich der auftretenden Verzögerungen und Verzögerungsschwankungen hoch. Informationsverluste sind zu einem gewissen Teil tolerierbar. Die Grenzwerte bezüglich Verzögerungen und Verzögerungsschwankungen orientieren sich an der subjektiven Toleranz des Menschen bei der Konversation.

- A/V-Abrufdienste

  Im Allgemeinen führen Informationsverluste, die auch durch Datenkomprimierung verursacht werden können, zu einem Qualitätsverlust, der vom menschlichen Nutzer nur innerhalb einer subjektiv festgelegten Grenze in Kauf genommen wird. Während Verzögerungen in einem gewissen Bereich toleriert werden, werden Verzögerungsschwankungen als störend empfunden. Es ist die Übertragung unkomprimierter Video-Ströme z. B. im HDTV-Format (*High Definition Television*) möglich. Zudem können Audio- und Video-Ströme z. B. durch ein MPEG-Verfahren (*Moving Picture Expert Group*) komprimiert werden.

2.1.1.2 Realisierung von lokal begrenzten Verteilten Systemen

Heutige Verteilte Systeme sind so aufgebaut, dass man ihre Kommunikationsfunktionen in Schichten einteilen kann. Es hat sich die Terminologie des OSI-Referenzmodells (*Open Systems Interconnection*) [58] mit sieben Schichten durchgesetzt. Jeder Schicht wird in der Regel ein Protokoll zugeordnet, das den Austausch von Protokolleinheiten zwischen den an der Kommunikation beteiligten Instanzen festlegt und somit den Datenaustausch realisiert.

In den Verteilten Systemen zur Unternehmensvernetzung spielt IP (*Internet Protocol*) als Protokoll der Vermittlungsschicht (Schicht 3 im OSI-Modell) eine herausragende Rolle (siehe Bild 2.1). Oberhalb der Vermittlungsschicht ist die Transportschicht (Schicht 4 im OSI-Modell) angeordnet. Hier finden das verbindungsorientierte Protokoll TCP (*Transmission Control Protocol*) und das verbindungslose Protokoll UDP (*User Datagram Protocol*) Verwendung. Die obersten Schichten (Schichten 5 bis 7 im OSI-Modell) werden bei IP-Netzen in einer Anwendungsschicht zusammengefasst. Hier werden anwendungsspezifische Protokolle eingesetzt. Ein typisches Protokoll ist HTTP (*Hypertext Transfer Protocol*), das bei der Datendienstanwendung WWW (*World Wide Web*) benutzt wird. Da keine Datenverluste erlaubt sind, nutzt HTTP die Dienste von TCP, das einen verlustfreien Datenaustausch garantiert. Für Videokonferenz-Anwendungen wird zumeist RTP (*Real-Time Transport Protocol*) verwendet, das auf UDP aufsetzt. Die Anforderungen, innerhalb gewissen Verzögerungs- und Verzögerungsschwankungsgrenzen die Daten auszugeben, werden durch den Einsatz von RTP unterstützt. Die empfangenen Daten werden hierbei gepuffert und zu für die Anwendung zulässigen Zeitpunkten ausgegeben. Datenverluste, die bei der Verwendung von UDP vorkommen können, werden in Kauf genommen.

Die unterhalb der Vermittlungsschicht vorzufindenden Protokolle (Schichten 1 und 2 im OSI-Modell) sind abhängig von der verwendeten Übertragungstechnologie. In lokalen Netzen dominiert Ethernet und in Weitverkehrsnetzen werden oftmals die Technologien SDH (*Synchronous Digital Hierarchy*) oder ATM (*Asynchronous Transfer Mode*) eingesetzt, so dass in Unternehmensnetzen an den Schnittstellen zu den Weitverkehrsnetzen eine Umsetzung erfolgen muss.

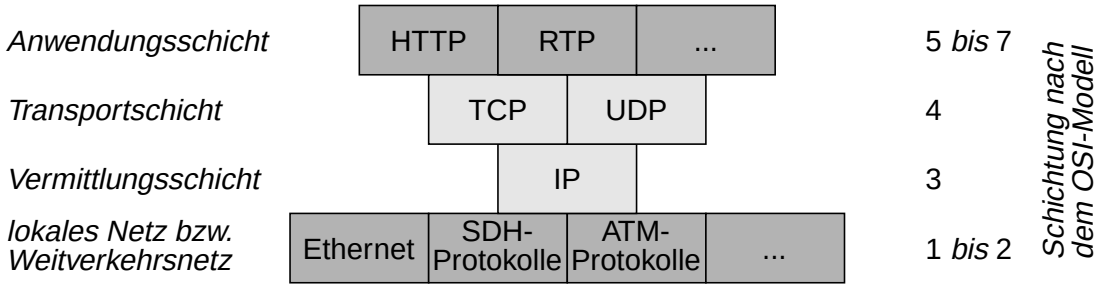


Bild 2.1: Sanduhr-Modell bei den Protokoll-Schichten in IP-Netzen

Bild 2.1 zeigt das in heutigen Verteilten Systemen vorherrschende „Sanduhr“-Modell. Während in den unteren und oberen Schichten eine Vielzahl von Protokollen verwendet wird, dominiert in der Vermittlungsschicht IP.

Es existieren Ansätze zur Berücksichtigung der Dienstgüteanforderungen bei A/V-Anwendungen in lokalen IP-Netzen, die in [32] klassifiziert sind. Der Standard IEEE 802.1D (*Institute of Electrical and Electronics Engineers*) spezifiziert Prioritätsbits innerhalb des durch den Standard IEEE 802.1Q erweiterten Ethernet-Rahmens. Sogenannte Switches können als Schicht-2-Vermittlungsknoten diese Prioritäten berücksichtigen. Rahmen für A/V-Ströme können somit durch eine höhere Priorität bevorzugt behandelt werden, um die benötigte Übertragungskapazität trotz weiteren Verkehrs, der nun niederprior behandelt wird, zu erhalten. Zudem wird innerhalb der RSVP-Spezifikation (*Resource Reservation Protocol*) ein sogenannter SBM (*Subnet Bandwidth Manager*) für die Verwaltung von Übertragungskapazität und die Durchführung einer Zulassungskontrolle (engl. *Admission Control*, AC) in lokalen Netzen vorgeschlagen. Anwendungen können nur dann A/V-Ströme austauschen, wenn die benötigte Übertragungskapazität beim SBM reserviert werden kann. Beim Versuch einer Überreservierung wird der Austausch nicht erlaubt.

2.1.2 Nutzung und Erbringung von Diensten

Dienstnutzer und -erbringer in der Anwendungsschicht werden durch Software-Instanzen realisiert, die in der Regel zum gegenseitigen Austausch von Daten Dienste der Transportschicht nutzen. Hierbei besitzt die Transportschicht eine Datenschnittstelle, die im Programmkontext als Programmierschnittstelle auftritt.

2.1.2.1 Kooperationsmodelle für Dienstnutzer und Diensterbringer

Für die Kooperation zwischen Dienstnutzer und Diensterbringer existieren zwei wesentliche Modelle.

- Client/Server-Modell

  Nach dem Client/Server-Modell entscheidet der Client aktiv, wann er einen Dienst nutzen möchte, während der Server passiv auf die Diensterbringung wartet. Beide Instanzen gehen während der Dienstnutzung bzw. -erbringung explizit eine Beziehung miteinander ein. Sehr oft sind Dienstnutzer und Diensterbringer synchron zueinander, d. h. der Client blockiert solange, bis der Server den Dienst erbracht hat, damit er dann das Ergebnis der Diensterbringung weiterverarbeiten kann. Das Einrichten einer Beziehung zwischen Dienstnutzer und Diensterbringer wird in dieser Arbeit auch als Binden bezeichnet.

- Erzeuger/Verbraucher-Modell

  Die vom Erzeuger produzierten Daten, die über eine dritte Instanz ausgetauscht werden können, werden vom Verbraucher bei Bedarf konsumiert. In der Regel blockiert der Erzeuger, falls der Datenspeicher der dritten Instanz mit noch nicht konsumierten Daten vollständig gefüllt ist. Zudem kann der Verbraucher blockieren, wenn der Datenspeicher keine neu erzeugten, d. h. noch nicht konsumierten Daten enthält.

Bei der Entwicklung von Software für Verteilte Systeme von Unternehmen lässt sich eine starke Orientierung am Client/Server-Modell feststellen. Ein Beispiel für die Bedeutung des Client/Server-Modells ist die häufige Verwendung der Socket-Programmierschnittstelle [22], die u. a. bei UNIX- und bei Windows-Betriebssystemen bereitgestellt wird. UNIX und Windows sind prozessorientierte Betriebssysteme. Ein Prozess ist hierbei eine sequentielle Folge von einzelnen Programmschritten, durch die eine abgeschlossene Aufgabe bearbeitet wird [50, 76]. Jedem Prozess steht ein eigener Speicherbereich zur Ablage von Daten zur Verfügung, und er wird bei den genannten Betriebssystemen nach einem unterbrechenden Prioritätenverfahren kombiniert mit einem Zeitscheibenverfahren von einer Prozessverwaltung einem Prozessor zugeteilt. Es wird zwischen Benutzerprozessen und Betriebssystemprozessen (Kernel-Prozessen) mit höherer Priorität unterschieden. Bezieht man sich auf das Schichtenmodell in Bild 2.1, so laufen in der Regel Software-Instanzen, die der Anwendungsschicht zuzuordnen sind, in einem Benutzerprozess, und Software-Instanzen, die unterhalb angeordnet sind, in einem Kernel-Prozess ab. Neben den eigentlichen Anwendungen werden auch Software-Instanzen, die Systemdienste (z. B. den Internet-spezifischen *Domain Name Service*, DNS) bereitstellen, der Anwendungsschicht zugeordnet.

Client und Server sind voneinander entfernt, wenn sie nicht einem gemeinsamen Prozess zugeordnet werden können. Mit Hilfe der Socket-Programmierschnittstelle kann die Interaktion von Clients mit entfernten Servern programmiert werden. Der prozessübergreifende Datenaustausch wird dann von einer Socket-Instanz, die die Schnittstelle realisiert und die Protokolle TCP oder UDP verwendet, ermöglicht.

2.1.2.2 Dienstnutzung und -erbringung mit Hilfe von Middleware

Schon im Jahre 1968 wurde von Naur und Randell [89] erkannt, dass die andauernd größer und komplexer werdende Software deren Entwicklung immer krisenanfälliger werden lässt, weil sich Anforderungen an ihre Wiederverwendbarkeit, Robustheit und Wartbarkeit immer schwieriger erfüllen lassen.

Ein allgemeiner Ansatz zur Vermeidung bzw. Entschärfung dieser Problematik ist, Software zu strukturieren und sie in verschiedene Abstraktionsniveaus einzuteilen. Als Paradebeispiel für dieses Vorgehen ist die Einführung des OSI-Referenzmodells zu nennen, bei dem Kommunikationsfunktionen in mehreren Schichten übereinanderliegen.

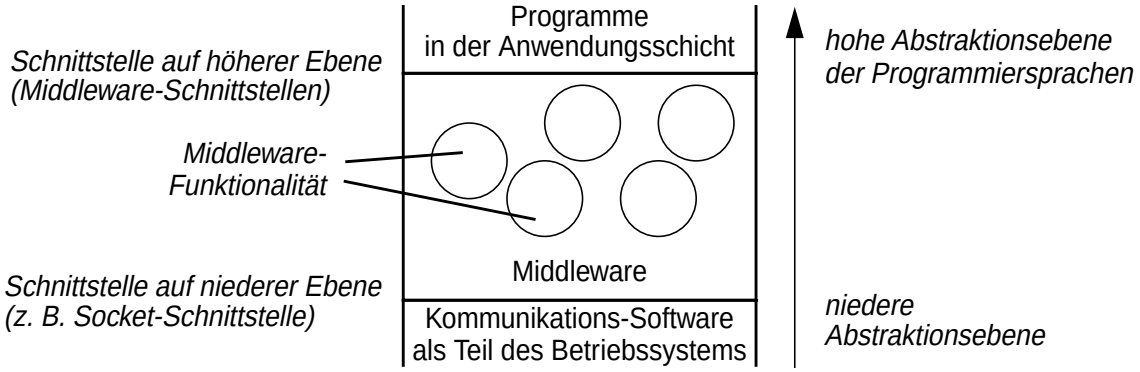


Bild 2.2: Anheben des Abstraktionsniveaus durch Middleware

Ein weiteres Beispiel sind die Ansätze, die dem Client/Server-Modell folgenden Programmierschnittstellen (z. B. Socket-Schnittstelle) auf das bei heutigen Programmiersprachen vorherrschende Abstraktionsniveau anzuheben. Die Software, die dieses Anheben ermöglicht, wird Middleware genannt. Sie ist die Software, die den Anwendungen eine neue Schnittstelle bereitstellt und selbst die alte Schnittstelle zum Kommunikationssystem nutzt (siehe auch Bild 2.2).

Anheben des Abstraktionsniveaus durch Middleware-Technologien

Wesentliche Vorteile, die sich durch das Anheben des Abstraktionsniveaus bei Verwendung einer Middleware-Technologie ergeben, sind:

- Einheitliches Programmiermodell

  Ohne Berücksichtigung von Middleware-Schnittstellen muss ein Software-Entwickler bei der verteilten Realisierung eines Client/Server-Systems u. a. mit serialisierten Byte-Folgen umgehen, die zwischen Client und Server ausgetauscht werden. Durch die Verwendung von Middleware kann ein Software-Entwickler den Austausch von Daten zwischen einem Client und einem Server so gestalten, dass wie innerhalb eines Prozesses Prozeduren bzw. Methoden aufgerufen werden. Beim Aufruf werden typisierte Variablenwerte als Parameter übergeben bzw. zurückgegeben. Ein Software-Entwickler muss nicht mehr in seinem Programmiermodell zwischen einer lokalen und einer entfernten Instanz (Client oder Server) unterscheiden. Die bei der niederen Programmierschnittstelle sequentiell auszuführenden Schritte zur Dienstbindung und Dienstnutzung bzw. -erbringung werden zusammengefasst und verborgen. Beispielsweise muss bei einer reinen Verwendung der TCP-Socket-Schnittstelle zuerst der Dienst gebunden werden, indem eine TCP-Verbindung aufgebaut wird. Erst dann kann der Dienst genutzt werden, indem die Daten über die Verbindung zwischen Client und Server ausgetauscht werden.

- Verbergen der Heterogenität Verteilter Systeme

  Durch Middleware wird die Heterogenität Verteilter Systeme verborgen. Dies bedeutet, dass ein Software-Entwickler sich nicht damit auseinandersetzen muss, dass die verwendeten Endgeräte, Betriebssysteme, Programmiersprachen und Kommunikationsprotokolle im Verteilten System unterschiedlich sein können.

- Bereitstellen zusätzlicher Funktionalität

  Zusätzliche, komplexe Funktionen, die die Verteilung von Software erfordert, wie z. B. Funktionen zur Lastverteilung, zur Dienstbekanntmachung oder zur Einhaltung von Sicherheitsrichtlinien, können durch Middleware erbracht werden.

Middleware-Technologie-Standards

Im Folgenden werden einige wichtige Middleware-Technologien, die das Client/Server-Modell umsetzen, vorgestellt.

- RPC (*Remote Procedure Call*) – Prozedurfernaufruf

  Das Funktionsprinzip des Prozedurfernaufrufs stammt aus den 70er Jahren und dient als Grundlage vieler nachfolgender Middleware-Standards. Es besteht darin, dass Stellvertreterinstanzen, auch Stubs genannt, die jeweilige entfernte Instanz innerhalb eines Prozesses lokal vertreten. Wie in Bild 2.3 gezeigt ist, ruft ein Client nicht direkt die Server-Prozedur auf, sondern die Stub-Prozedur auf Client-Seite, die den Server vertritt. Eine Server-Prozedur wird zudem immer vom Client-Stellvertreter auf Server-Seite, dem Server-Stub, aufgerufen. Beim Prozeduraufruf werden dem Client-Stub Parameter übergeben, die dann in eine serielle Byte-Folge gewandelt werden, um sie über das Netz transportieren zu können. Es

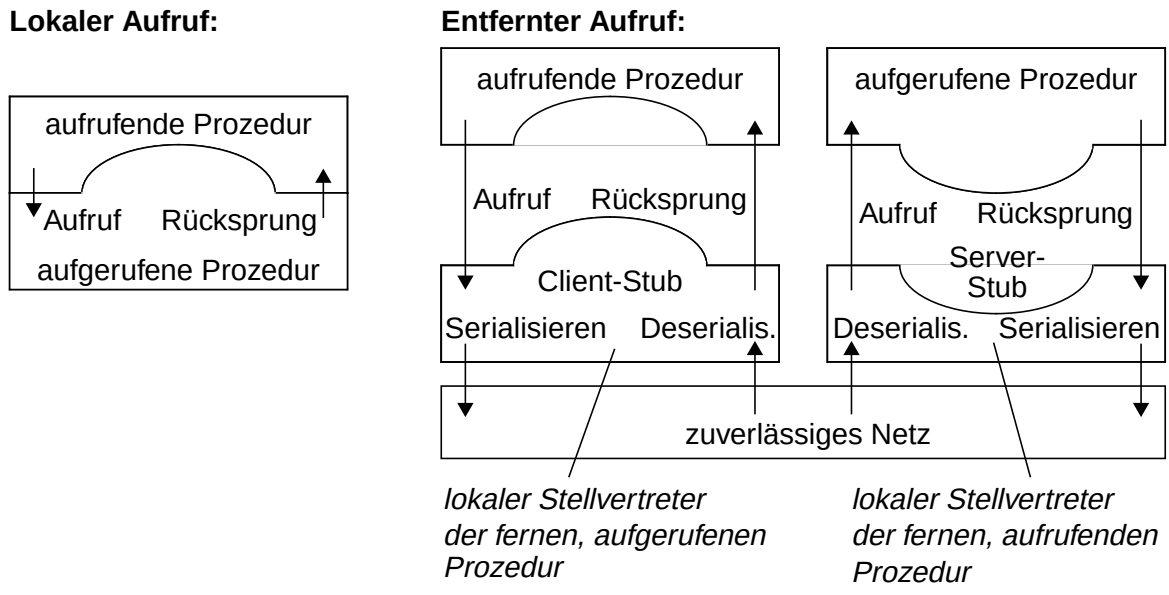


Bild 2.3: Lokaler Aufruf und Fernaufruf

existieren Standards für Umwandlungsregeln, die der Schicht 6 des OSI-Modells (Darstellungsschicht) zugeordnet sind. Der Server-Stub empfängt die Daten und wandelt sie in geeignete Datenrepräsentationen um, die dann als Parameter bei Aufruf der Server-Prozedur übergeben werden. Nach Berechnung des Ergebnisses muss der Rückgabewert ebenfalls als serialisierte Byte-Folge zur Client-Seite übertragen werden.

Dem Software-Entwickler wird eine Schnittstellenbeschreibungssprache (engl. *Interface Definition Language*, IDL) zur Verfügung gestellt, deren Syntax an die Programmiersprachen C bzw. C++ angelehnt ist, und mit der er die Schnittstelle zwischen Client und Server, d. h. die Signatur der Server-Prozedur im Kontext der jeweils verwendeten Programmiersprache, beschreibt. Aus der Schnittstellenbeschreibung wird unter Verwendung eines Übersetzerprogramms der programmiersprachenspezifische Code der Stubs erzeugt.

- RMI (*Remote Method Invokation*) – Methodenfernaufruf bei Java

  Der Programmiersprache Java [43] werden mit dem Methodenfernaufruf [19] Mechanismen in Form einer Bibliothek zur Seite gestellt, die denen des Prozedurfernaufrufs sehr ähnlich sind. Da Java dem objektorientierten Programmierparadigma folgt, sind der Client, der Server und die beiden Stubs Objekte, die durch Methodenaufrufe beauftragt werden können. Der Server-Stub wird auch als Skeleton bezeichnet. Die Existenz einer eigenen Schnittstellenbeschreibungssprache ist nicht notwendig, da die Schnittstelle zwischen Client und Server durch das spracheigene Interface-Konstrukt spezifiziert wird.

- CORBA (*Common Object Request Broker Architecture*)

  Ein weiterer Middleware-Standard für das objektorientierte Programmierparadigma ist CORBA [92]. Das zentrale Element bildet ein ORB (*Object Request Broker*, siehe Bild 2.4*)*, der die Infrastruktur zwischen dem Client und dem Server zur Verfügung stellt. Bei CORBA sind Server immer Objekte, wobei verschiedene Objektrealisierungstechnologien unterstützt werden. Deswegen ist auf Server-Seite der Einsatz von sogenannten Objektadaptern notwendig, die eine Anpassung auf die jeweilige Technologie vornehmen. Beispiels-

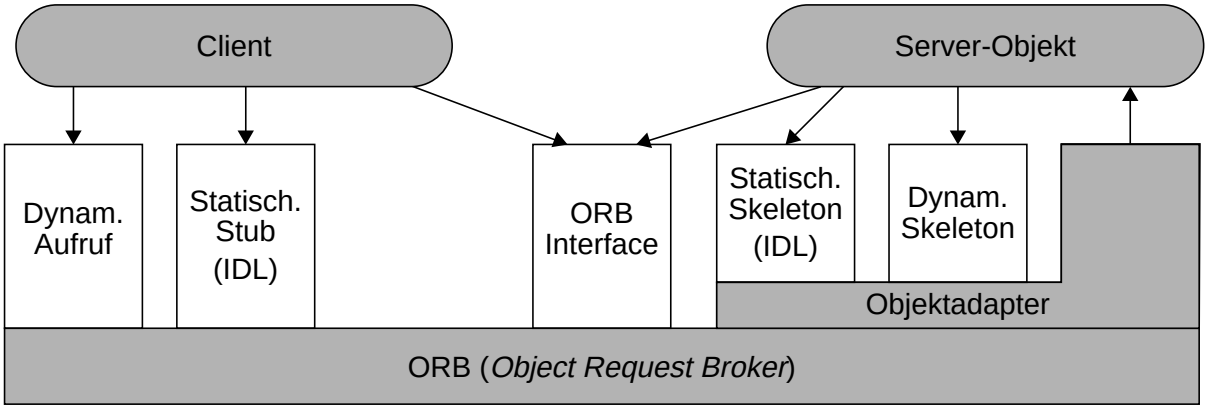


Bild 2.4: ORB – Object Request Broker

weise können spezielle Adapter den Zugriff auf in Datenbanken gespeicherte Objekte ermöglichen. Der Objektadapter für in gewöhnlichen Programmiersprachen geschriebene Server-Programme wird als POA (*Portable Object Adapter*) bezeichnet.

Bei CORBA besteht wiederum die Möglichkeit, eine Client/Server-Kommunikation über Stellvertreterinstanzen (Stub und Skeleton) zu ermöglichen. Für CORBA existiert eine Beschreibungssprache (IDL), mit der die Gestalt der Server-Schnittstelle vom Software-Entwickler festgelegt wird und die in den Kontext der jeweils verwendeten Programmiersprache übersetzt werden kann. Neben der Realisierung von Schnittstellen durch Stellvertreter-Instanzen, die i. A. vor der Laufzeit erzeugt und vorhanden sein müssen, besteht noch die Möglichkeit, Schnittstellen während der Laufzeit dynamisch zu generieren. Zudem können über eine spezielle ORB-Schnittstelle Eigenschaften des ORBs modifiziert werden.

Das IIOP (*Internet Inter-ORB Protocol)* ermöglicht den Austausch von typisierten Daten zwischen Client- und Server-Instanzen unter Verwendung von TCP/IP. Hierbei legt die CDR (*Common Data Representation*) die Transfersyntax, d. h. die Regeln, wie die typisierten Daten in einer sequentiellen Byte-Folge dargestellt werden, fest.

- DCOM (*Distributed Component Object Model*)

  DCOM [20] ist eine Erweiterung des auf dem Client/Server-Modell basierten COM (*Component Object Model*) der Firma Microsoft. Ein COM-Objekt in der Rolle eines Servers bietet seine Dienste einem anderen COM-Objekt, das dem Client entspricht, durch mindestens eine Schnittstelle an. DCOM erweitert COM dahingehend, dass die Clients entfernte Server-Objekte beauftragen können. Dies wird möglich, da DCOM intern RPC-Mechanismen einsetzt.

- SOAP (*Simple Object Access Protocol*)

  Mit Hilfe des SOAP-Standards [139] können Anwendungen strukturierte und typisierte Daten auf Basis des Protokolls HTTP austauschen. SOAP legt hierbei die Transfersyntax in einem XML-basierten (*Extensible Markup Language*) Format fest. SOAP beschreibt nicht, wie die Dienstschnittstelle aussehen soll. So ist beispielsweise bei CORBA-Systemen möglich, dass anstatt dem Protokoll IIOP das Protokoll SOAP zum Datenaustausch verwendet wird.

  Mit WSDL (*Web Service Description Language*) [140] wird eine auf XML basierte Grammatik zur Beschreibung von Dienstschnittstellen spezifiziert. In Verbindung mit SOAP können nun Clients und Server, die dann Web Services genannt werden, zusammenwirken.

2.1.3 Verwaltung von Diensten durch Middleware

Middleware-Technologien stellen neben einer Schnittstelle zum Kommunikationssystem zusätzliche Dienste und Funktionen zur Verwaltung der verteilten Software-Instanzen bereit. Sie sind Teil der in Bild 2.2 beschriebenen Middleware-Funktionalität, die den Mehrwert einer Technologie für die Software-Entwicklung steigern.

2.1.3.1 Verwaltungsdienste und -funktionen am Beispiel von CORBA

Im Folgenden werden exemplarisch einige bedeutende Dienste und Funktionen von CORBA vorgestellt [115]:

- Namensdienst [96]

  Bei CORBA ist die Möglichkeit gegeben, existierende Server einem Client bekanntzumachen. Der Client fragt bei einem Namensdienst nach einem zu nutzenden Dienst und bekommt im Erfolgsfall eine Referenz auf einen Server zurück, mit der er ihn beauftragen kann. Den zentrale Zugang zum Namensdienst geschieht i. A. über den ORB.

- Benachrichtigungsdienste [94, 97]

  Die vorgestellten Middleware-Technologien basieren auf dem Client/Server-Kooperationsmodell. Einige Middleware-Technologien (wie auch CORBA) stellen zusätzlich eine standardisierte Infrastruktur für Realisierungen gemäß dem Erzeuger/Verbraucher-Kooperationsmodell zur Verfügung. So können Nachrichten von Erzeugerinstanzen generiert und in einen sogenannten Kanal gesendet werden. Der Kanal speichert eine Nachricht solange, bis sie von mindestens einer Verbraucherinstanz konsumiert wurde oder bis ein Gültigkeitszeitraum überschritten wurde. Einer Nachricht kann eine Priorität, ein Gültigkeitszeitraum oder auch einen frühesten Lieferzeitpunkt zugeordnet werden. Erzeuger und Verbraucher sind entkoppelt.

- Sicherheitsdienste [98]

  - Es gibt bei CORBA einen Standard für einen Sicherheitsdienst, der eine vertrauliche Kommunikation ermöglicht. Die ausgetauschten Daten werden nur den dafür autorisierten Nutzern zugänglich gemacht.
  - Ein weiterer Standard spezifiziert, wie die Integrität einer Kommunikation gewährleistet werden kann. Die ausgetauschten Daten können bei einer solchen Kommunikation nur von dafür autorisierten Nutzern verändert werden.
  - Zudem wird ein Dienst zur Zuweisung von Verantwortlichkeit spezifiziert. Es werden Nicht-Abstreitbarkeitsmechanismen vorgeschlagen, damit die Verantwortung für ein Handeln nicht geleugnet werden kann.

In CORBA werden die jeweiligen sicherheitsspezifischen Dienste als Erweiterung des zentralen ORB-Elementes vorgesehen.

- Lastverteilungsfunktion [95]

  Die Realisierung eines Lastverteilungverfahrens erlaubt es, die Last die durch die Bearbeitung von Client-Anfragen zustande kommt, auf mehrere Server zu verteilen.

Es ist eine Vielzahl von weiteren Diensten und Funktionen, die in CORBA spezifiziert sind, zu nennen. Es gibt Dienste zur Bestimmung der Zeit im Verteilten System, zur Verwaltung von Operationen auf Gruppen von Objekten, zur Koordination von Client-Zugriffen auf gemeinsame Ressourcen und zur Verwaltung der Lizensierung von Software. Weitere Dienste sind Trading [105], bei dem den Dienstnutzern nach Optimierungsgesichtspunkten ein Diensterbringer zugeordnet wird, ein Transaktionsdienst, und Dienste, die das Erzeugen, Löschen, Kopieren, Migrieren und die Persistenz von Objekten verwalten.

2.1.3.2 Bekanntmachung von Diensten in Verteilten Systemen

Vor allem beim Client/Server-Kooperationsmodell, bei dem eine Bindung zwischen Dienstnutzer und Diensterbringer vorgesehen ist, muss in einem Verteilten System einem potenziellen Dienstnutzer der Zugangsort zu einem Erbringer bekannt gemacht werden. Dieser Vorgang wird auch als Dienstentdeckung (engl. *Service Discovery*) bezeichnet und ist ein grundlegender Dienst, der von einer Middleware-Technologie bereitgestellt wird. Bei CORBA wird eine Dienstbekanntmachung z. B. durch den Namensdienst ermöglicht.

Gewöhnlich wird ein Bekanntmachungsmechanismus durch einen Registraturdienst realisiert. Das Problem, dass einem Dienstnutzer sein Diensterbringer bekannt gemacht werden muss, lässt sich somit auf das Problem abbilden, dass dem Dienstnutzer der Erbringer des Registraturdienstes bekannt sein muss. Er nutzt dann zuerst den Registraturdienst, indem er dort nach einem passenden Diensterbringer sucht. Jeder Diensterbringer muss folglich ebenso dem Erbringer des Registraturdienstes bekannt gemacht werden, d. h. er muss sich registrieren. Es existieren im Wesentlichen zwei Ansätze:

- Beim Portalkonzept kennt jede Instanz, die mit der Registratur zusammenwirken möchte, inhärent den zentralen Ort des Dienstzugangs. Der CORBA-Namensdienst ist beispielsweise nach dem Portalkonzept konzipiert.

- Beim Rundsendekonzept (engl. *Broadcast*) findet ein Zusammenwirken mit der Registratur statt, ohne den Ort des Zugangs zum Registraturdienst zu kennen. Es werden die Informationen von Instanzen, die entweder den Registraturdienst nutzen möchten oder die sich registrieren möchten, durch ein Rundsendeverfahren verteilt. Dieses Konzept kann besonders einfach innerhalb von lokalen Netzen realisiert werden, deren Übertragungsmechanismen inhärent auf dem Rundsendeprinzip beruhen. Dies ist z. B. bei Ethernet der Fall.

2.1.4 Realisierung von Diensten mit Hilfe von Software-Komponenten

Heutzutage existieren einige Komponenten-Software-Technologien, die den Zusammenbau von Anwendungen aus Software-Komponenten ermöglichen. Sie unterstützen die Auswahl, Konfiguration und die Verknüpfung von Komponenten [39]. Durch das Bereitstellen von Software-Komponenten als Standardbausteine soll die Entwicklung komplexer Software-Systeme beherrschbarer gemacht werden. Software-Komponenten haben hierbei Produkt-Charakter [39], d. h. es wird ein Markt von Komponentenherstellern angestrebt, die wiederverwendbare Software-Komponenten anbieten, aus denen dann die entsprechenden Anwendungen zusammengesetzt werden können.

Die in der vorliegenden Arbeit betrachteten Komponenten-Software-Technologien ermöglichen den Zusammenbau verteilter Anwendungs-Software. Während die bisher betrachteten Middleware-Technologien sich im Wesentlichen auf die Kommunikation zwischen den hierbei verteilten Komponenten beziehen, stehen bei den im Folgenden betrachteten Komponenten-Software-Technologien die Komponenten selbst und deren Verwaltung im Verteilten System im Vordergrund. Beide Technologiearten stehen nicht disjunkt zueinander. Insbesondere die in Kapitel 2.1.3 aufgezählten Verwaltungsdienste und -funktionen können durch beide Technologiearten realisiert werden.

2.1.4.1 Definitionen zum Begriff der Software-Komponente

In dieser Arbeit ist der Begriff Software-Komponente wie folgt definiert:

Eine Komponente beinhaltet ein Programm, das durch binären Code beschrieben ist. Sie kann zur Systemlaufzeit aufgrund einer Komponenten-Software-Technologie in eine Ablaufumgebung geladen werden. Eine Komponente ist aktiv, wenn auf dieser Ablaufumgebung mindestens ein Bearbeitungsvorgang, der nach ihrem Programm abläuft, existiert. Ein existierender Vorgang kann entweder ablaufen oder er kann blockiert sein. Es ist möglich, dass mehrere Bearbeitungsvorgänge einer Komponente zueinander nebenläufig sind. Eine Komponente besitzt einen Daten- und einen Aktivitätszustand. Durch den Zugang über eine Daten- oder über eine Verwaltungsschnittstelle wird der Aktivitätszustand von außen geändert. Nur durch den Zugriff über eine Datenschnittstelle kann der Datenzustand von außen verändert werden. Die Schnittstellen sind nach dem Client/Server-Kooperationsmodell mit Hilfe einer Middleware-Technologie gestaltet. Dies bedeutet, dass Komponenten bzw. ihre verantwortlichen Client- und Server-Instanzen zueinander gebunden sein müssen, um miteinander zu kooperieren. Bei der Realisierung der Schnittstellen nach dem Prinzip des Methodenfernaufrufs, bei der eine Komponentenbeauftragung durch einen Methodenaufruf realisiert wird, findet ein Binden direkt vor der Bearbeitung des Aufrufs statt und die Bindung wird sofort wieder nach dieser

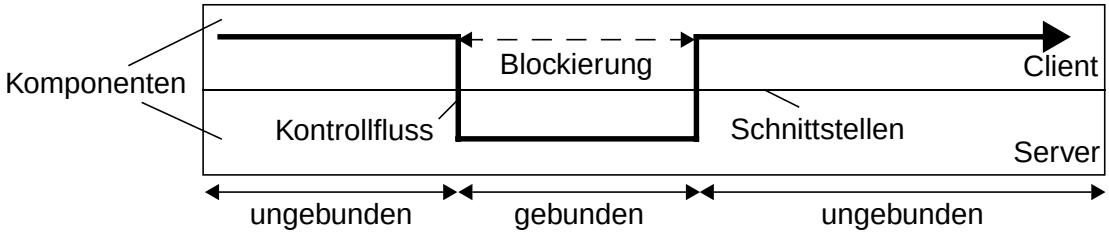


Bild 2.5: Kooperation zueinander gebundener Komponenten

Bearbeitung gelöst. Bild 2.5 zeigt den Kontrollfluss durch miteinander kooperierende Komponenten. Unter Kontrollfluss wird das Durchlaufen von Programmen aufgrund ihrer Kontrollstruktur verstanden. Nach dem Client/Server-Modell blockiert der Client solange, bis die Blokkierung vom Server aufgehoben wird. Er gibt während der Blockierung die Kontrolle an den Server ab.

Im Folgenden sind weitere, in der Literatur zu findende Definitionen zum Begriff Software-Komponente aufgeführt. Die zuerst genannten Definitionen besagen, dass eine Komponente Schnittstellen besitzen muss:

- „Eine Software-Komponente muss eine abgeschlossene funktionale Einheit mit vollständig definierten Schnittstellen bilden.“ [131]
- „Eine Software-Komponente ist ein Software-Paket mit genau definierten und bekannt gemachten Schnittstellen.“ [54]

Die nächsten beiden Definitionen beziehen sich auf den Aspekt der Software-Entwicklung und lenken den Schwerpunkt auf die Wiederverwendbarkeit einer Software-Komponente und deren Herstellung durch voneinander unabhängige Entwickler:

- „Software-Komponenten sind zum Zusammenarbeiten konzipiert und von verschiedenen Entwicklern unter Einsatz verschiedener Programmiersprachen, Entwicklungswerkzeugen und Endgerätumgebungen unabhängig voneinander erschaffen worden.“ [131]
- „Eine Software-Komponente ist ein Programmelement mit den folgenden Eigenschaften: Das Element kann von anderen Programmelementen (Clients) genutzt werden. Das Element muss von Client-Entwicklern eingesetzt werden können, die neue Systeme entwickeln, die der Entwickler des Programmelements nicht vorgesehen hat.“ [86]

Der Einsatz und das Verhalten in einer Laufzeitumgebung ist Mittelpunkt von folgenden Definitionen:

- „Eine Laufzeit-Software-Komponente ist ein dynamisch (zum System) bindbares Paket, das aus einer Einheit von ein oder mehreren Programmen besteht, und auf das über dokumentierte Schnittstellen, die zur Laufzeit entdeckt werden können, zugegriffen werden kann.“ [84]

- Komponenten sind „kleine Code-Blöcke, die zur Laufzeit zusammenarbeiten können.“ [131]

Das Komponenten-Software-Paradigma und das objektorientierte Paradigma sind eng miteinander verwandt. Die gleichen Ziele, Software wiederverwendbar und erweiterbar zu machen, werden mit den gleichen Mitteln jedoch mit unterschiedlicher Granularität verfolgt. In [86] wird das Verbergen von Information, Datenabstraktion durch Trennung der Implementierung von den Schnittstellen, Polymorphismus, dynamische Bindung und die Vererbung genannt. Folgende Definitionen beschreiben den Bezug der Begriffe Komponente und Objekt zueinander:

- „Eine Komponente ist idealerweise ein sprach- und plattformunabhängiges binäres Modul, das – ähnlich wie ein Objekt – spezielle Funktionen und Daten kapselt.“ [24]
- „Eine Komponente ist definiert als eine Sammlung von Objekten mit einer klaren Umgrenzung zu anderen Objekten bzw. Komponenten. Objekte innerhalb einer Komponente kooperieren stark, während das komponentenübergreifende Zusammenwirken relativ schwach ist.“ [103]
- Allgemein gilt, dass „(eine) Komponenten (-Software-Technologie) eine objektorientierte Technologie voraussetzt, anwendet und fördert.“ [87]

2.1.4.2 Mechanismen von Komponenten-Software-Technologien

Durch den Einsatz einer Komponenten-Software-Technologie werden bei der Software-Entwicklung Mechanismen standardisiert:

- Es wird ein standardisierter Weg zum Zusammensetzen von Anwendungen aus einzelnen Komponenten bereitgestellt. Insbesondere kann durch eine sogenannte ADL (*Architecture Description Language*) ein komponentenbasiertes System beschrieben werden [34, 65, 72].
- Es wird die Verwendung sowohl von Modellierungssprachen wie UML (*Unified Modeling Language* [99]) als auch von sogenannten Entwurfsmustern (engl. *Design Patterns*) [37] gefördert.

Durch den Einsatz einer Komponenten-Software-Technologie können zudem folgende Mechanismen zur Systemlaufzeit bereitgestellt werden:

- Es wird ein automatisierter Weg zum Installieren und Entfernen einer Komponente und somit eines Dienstes bereitgestellt.
- Die installierte Komponente wird auf standardisierte Art und Weise verwaltet. Hierzu werden die in Abschnitt 2.1.3 aufgezählten Dienste und Funktionen bereitgestellt.

Komponenten-Software-Technologien lassen i. A. die Integration von Middleware-Technologien zu, durch die eine standardisierte Schnittstelle zur Kommunikation von Komponenten bereitgestellt wird.

2.1.4.3 Komponenten-Software-Technologie-Standards

Im Folgenden werden wesentliche Komponenten-Software-Technologie-Standards vorgestellt.

Standards für die Internet- und Intranet-Vernetzung

Unter Internet-Vernetzung wird eine globale IP-basierte Vernetzung und unter Intranet-Vernetzung eine IP-basierte Vernetzung innerhalb eines Unternehmens verstanden. Zunächst werden zwei Standards von Komponenten genannt, die auf Client-Seite für beide Bereiche zum Einsatz kommen:

- Applets

  Ein Applet ist eine in Java geschriebene Komponente, welche in einer speziellen Laufzeitumgebung abläuft, die in der Regel innerhalb eines WWW-Browsers bereitgestellt wird. Der Zugang zu Systemdiensten und Ressourcen ist für ein Applet nur unter Berücksichtigung gegebener Sicherheitsrichtlinien möglich (Sandbox-Prinzip). Ein Applet kann mit Hilfe von HTTP in die Umgebung des Clients zur Systemlaufzeit geladen und dann ausgeführt werden.

- ActiveX-Komponenten

  Eine ActiveX-Komponente entspricht einer COM-Komponente (siehe Abschnitt 2.1.2.2 unter DCOM) und kann ebenfalls mit Hilfe von HTTP in einen WWW-Browser geladen werden und dort ablaufen.

Mit der Servlet-Technologie ist ein Standard gegeben, der auf Server-Seite eingesetzt werden kann.

- Servlets

  Ein Servlet [129] ist eine in Java geschriebene Komponente, welche die Funktionalität eines Servers, insbesondere eines HTTP-Servers, erweitert. Hierzu wird das Servlet vom Server entweder bei seinem Start oder bei einer entsprechenden Client-Anfrage geladen. Zur Initialisierung, zur Beantwortung einer Client-Anfrage und zur Deaktivierung stellt ein Servlet entsprechende Schnittstellen gemäß dem Java-Interface-Konzept bereit.

Standards für mehrstufige Architekturen in Unternehmensnetzen

Traditionelle Unternehmensnetze, wie z. B. klassische Intranet-Netze, sind gemäß zweistufigen Architekturen (engl. *Two Tier Architectures*) aufgebaut: Die Anwendungen treten in Funk-

tion von Dienstnutzern auf, die Systemdienste entsprechend dem Client/Server-Kooperationsmodell nutzen.

In Bild 2.6 ist eine mehr- bzw. dreistufige Architektur (engl. *Multi Tier Architecture*) dargestellt, die den zweistufigen Architekturansatz immer öfter ersetzt. Zwischen der ersten Stufe, der die Benutzerschnittstelle zugeordnet werden kann, und der dritten Stufe für die Systemdienste, existiert nun eine mittlere Stufe, in der die Steuerung des Anwendungsablaufs untergebracht ist.

Vorteile, die dieser Architekturansatz mit sich bringt, sind:

- Lastverteilung

  Die Software für die Anwendungsablaufsteuerung kann nach Lastverteilungsgesichtspunkten im System verteilt werden. Zudem können Mechanismen zur Lastverteilung und Überlastabwehr vorgesehen werden, um die Last auf Ressourcen der dritten Stufe (u. a. Datenbanken) zu verteilen oder Ressourcen vor Überlast zu schützen.

- Wiederverwendbarkeit

  Die Steuerung des Anwendungsablaufs (z. B. Konsistenzverwaltung bei Datenbankzugriffen) findet auf logischer Ebene nur an einer Stelle statt, kann aber Teil von vielen Anwendungen sein. Ein Ändern dieser Steuerung muss nur an einer Stelle und nicht mehr in vielen monolithischen Anwendungen vorgenommen werden.

- Verbergen der Infrastruktur im Verteilten System

  Die im vorherigen Absatz behandelte Konzentration der Steuerung des Anwendungsablaufs bringt zudem mit sich, dass die Nutzung von Systemdiensten von einer Stelle aus erfolgen kann und der eigentlichen anwendungsspezifischen Software verborgen bleibt. Ein Ändern

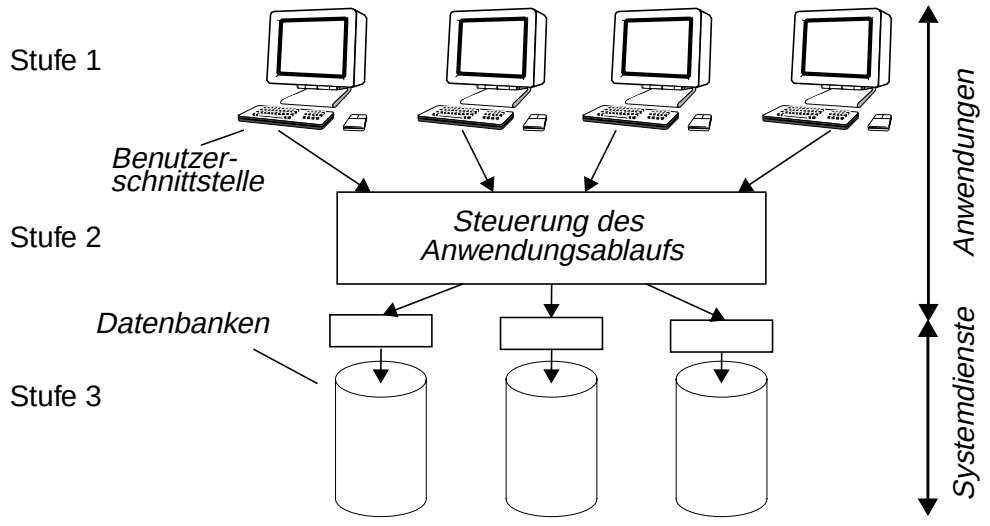


Bild 2.6: Dreistufige Architektur für ein Unternehmensnetz

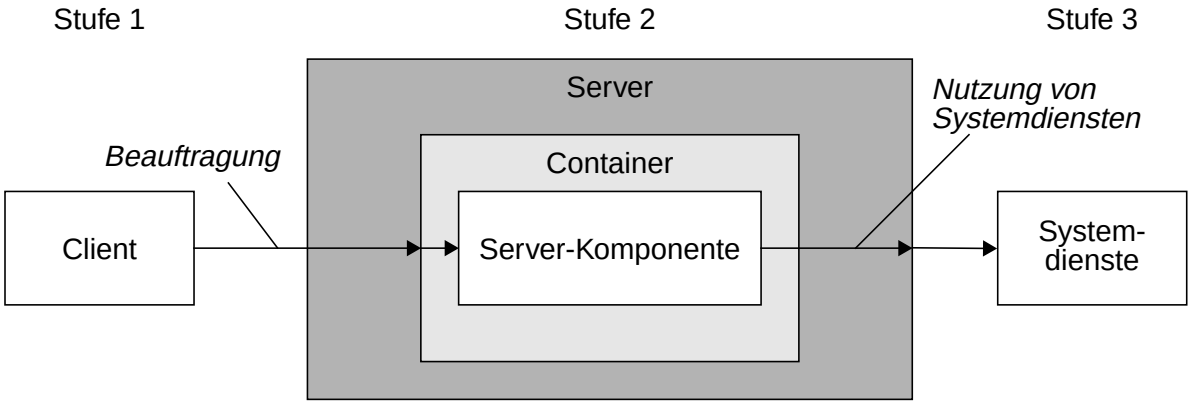


Bild 2.7: Komponenten-Ausführungsmodell bei EJB und CCM

der in der dritten Stufe vorgesehenen Infrastruktur ist deshalb transparent für die Entwicklung dieser Software.

Es existieren einige Software-Komponentenstandards für die mittlere Stufe eines Unternehmensnetzes:

- CORBA-basiertes Komponentenmodell (*CORBA Component Model*, CCM)

  CCM [93] ist als Erweiterung der CORBA-Middleware-Spezifikation anzusehen. Die Client/Server-Kommunikation geschieht nach dem Client/Server-Kommunikationsmodell über einen ORB gemäß Abschnitt 2.1.2.

- Java-basiertes Komponentenmodell (*Enterprise Java Beans*, EJB)

  Die EJB-Spezifikation [130] legt eine Client/Server-Kommunikation durch Java RMI nahe, die ebenfalls in Abschnitt 2.1.2 eingeführt wurde.

  Beide Spezifikationen besitzen das gleiche, in Bild 2.7 eingeführte Komponenten-Ausführungsmodell. Man unterscheidet auf Server-Seite (Stufe 2) zwischen folgenden Elementen:

  - Ein Server bildet die eigentliche Laufzeitumgebung. Er stellt den Zugang zu Betriebssystemdiensten und zu den Systemdiensten zur Verfügung. Er ist physikalisch auf mehreren Endgeräten realisiert und sorgt auf diesen für eine Lastverteilung.
  - Ein Container verwaltet eine Server-Komponente (z. B. nimmt er gegebenenfalls eine Persistenzverwaltung vor). Desweiteren ist er für die Aktivierung und Deaktivierung einer Komponente zuständig. Der Client ruft niemals eine Server-Komponente direkt auf, sondern immer über die Schnittstellen des Containers.
  - Bei den Server-Komponenten treten folgende Typen auf:
    - Es gibt Komponenten, die nur für die Bearbeitung einer Beauftragung zuständig sind. Solche Komponenten sind nicht persistent, d. h. nach der Bearbeitung der Client-Beauftragung werden sie deaktiviert.

  - Es gibt Komponenten, die nur für einen Client zuständig sind. Sie sind ebenfalls nicht persistent, d. h. mit der Deaktivierung des Clients werden sie ebenfalls deaktiviert.
  - Es gibt Komponenten, auf die von mehreren Clients aus über den entsprechenden Container zugegriffen werden kann und die persistent sind.

- COM+

  Ein produktnaher Standard ist COM+ [114], den die Firma Microsoft für seine Windows-Plattformen im Rahmen ihrer .net-Initiative bereitstellt. Er entspricht der Zusammenführung der Standards COM/DCOM und MTS (*Microsoft Transaction Server*), die noch um zusätzliche Verwaltungsdienste ergänzt werden. Auch MTS ähnelt dem in Bild 2.7 beschriebenen Komponentenausführungsmodell.

2.2 Dienste in Verteilten Systemen für die Vernetzung von Heimunterhaltungselektronik

Seit Anfang der 90er Jahre gibt es Ansätze, Elektronikgeräte des Heimunterhaltungsbereichs miteinander zu vernetzen, um durch das Zusammenwirken der Geräte einen Mehrwert zu erzielen. Im Laufe der Zeit kristallisierten sich einige Standards und Produkte heraus, die einen Durchbruch bei der Vernetzung in naher Zukunft ermöglichen können.

2.2.1 Anforderungen von Anwendungen und Systemrealisierung

Auch für die Anwendungen der Heimunterhaltungselektronik kann die in Abschnitt 2.1.1.1 eingeführte Klassifikation in Datendienste, interaktive A/V-Dienste und A/V-Abrufdienste vorgenommen werden.

In Abschnitt 2.1.1.2 wurde vorgestellt, dass in lokalen Unternehmensnetzen IP einheitlich verwendet wird. Jedes Endgerät ist somit mit einer IP-Adresse adressierbar. Auch in der Heimvernetzung deutet sich an, dass IP eine dominierende Stellung einnimmt und dass die Heimgeräte mit IP in einheitlicher Form global adressiert werden können.

Weiter haben derzeit im Heimbereich spezielle Bussysteme, die nicht für die Unternehmensvernetzung eingesetzt werden, eine große Bedeutung. So werden beispielsweise die zeitlichen Anforderungen beim Austausch von A/V-Strömen wegen fehlender Pufferungsmechanismen in den Endgeräten durch Reservierung fester Übertragungskapazitäten auf dem Bussystem erfüllt. Hierfür eignet sich das Bussystem IEEE 1394, auch als Firewire und iLink bezeichnet, das zu einer Standard-Technologie im Heimbereich geworden ist. Fast jede digitale Videokamera, aber auch viele PCs (*Personal Computers*), sind mit einem IEEE 1394-Anschluss ausgestattet, wobei heute diese Schnittstelle zumeist nur dazu verwendet wird, eine Videokamera an

einen PC anzuschliessen. Diese Punkt-zu-Punkt-Verknüpfung kann als Anfangsszenario einer umfassenden, zukünftigen Heimvernetzung angesehen werden.

IEEE 1394

IEEE 1394 ist ein Standard [56] für ein Bussystem, mit dem über eine serielle Schnittstelle Daten in einer Rate bis zu 400 Mbit/s ausgetauscht werden können. Das Bussystem soll verschiedenartige Geräte wie Videokameras, PCs, CD- und DVD-Spieler (*Compact Disk, Digital Versatile Disk*), Festplatten und Hifi-Geräte miteinander vernetzen. Es ist möglich, dass an einem sich im Betrieb befindlichen Bussystem weitere Geräte hinzugefügt bzw. entfernt werden können, wenn die baumförmige Topologie erhalten bleibt. Eine wesentliche Eigenschaft von IEEE 1394 ist, dass Garantien für Übertragungskapazitäten gegeben werden können. Dies wird möglich, da neben einem sogenannten asynchronen Übertragungsdienst auch ein sogenannter synchroner Übertragungsdienst bereitgestellt wird.

Die Übertragung von Daten mit Hilfe des asynchronen Übertragungsdienstes geschieht durch das Versenden von Rahmen variabler Länge von einem Sendegerät zu einem Empfangsgerät, wobei im Kopf eines Rahmens die Adresse des Sende- und Empfangsknotens enthalten ist. Der Empfangsknoten quittiert durch das Zurücksenden eines Bestätigungspaketes den Empfang. Durch diesen asynchronen Übertragungsdienst ist eine gesicherte Übertragung möglich, d. h. eine fehlerhafte Übertragung kann vom Sender wahrgenommen werden.

Die Übertragung von Daten mit Hilfe des synchronen Übertragungsdienstes geschieht durch das Versenden von Rahmen gleicher Länge in nahezu regelmäßigen Zeitabständen per Rundsenden. Dies bedeutet, dass jedes angeschlossene Gerät die Rahmen empfängt und ein einen logischen Kanal beschreibendes Feld im Kopf des Rahmens auswertet, um zu entscheiden, ob das Paket in den Empfangspuffer des Gerätes geschrieben wird. Bei IEEE 1394 wird oftmals anstatt von synchroner Übertragung auch von isochroner Übertragung gesprochen.

Es ist garantiert, dass innerhalb eines Taktzyklus die Arbitrierung des Übertragungsmediums für das Versenden eines Rahmens nach dem synchronen Übertragungsdienst früher als eine Arbitrierung eines Rahmens nach dem asynchronen Übertragungsdienstes behandelt wird. Somit ist bei keiner Überreservierung garantiert, dass ein Rahmen mittels des synchronen Übertragungsdienstes innerhalb eines Zyklus übertragen werden kann. Für die Übertragung von Daten mit Hilfe des synchronen Übertragungsdienstes kann deshalb eine konstante Bitrate garantiert werden.

2.2.2 Nutzung, Erbringung und Verwaltung von Diensten

Im Folgenden werden Middleware- bzw. Komponenten-Software-Technologien vorgestellt, die zukünftig bei der Heimvernetzung eine wichtige Rolle spielen können.

2.2.2.1 Spontane Vernetzung

Eine grundlegende, gemeinsame Intention der im Folgenden vorgestellten Technologien ist, den Anschluss (engl. *Plugin*) eines Gerätes an ein Netz zur Systemlaufzeit zu ermöglichen, so dass ohne zusätzlichen manuellen Administrationsaufwand die im Gerät lokalisierten Dienste die Dienstmenge im Verteilten System erweitern bzw. die im Verteilten System angebotenen Dienste von Instanzen im neu angeschlossenen Gerät genutzt werden können. Diese Eigenschaft wird auch mit dem Begriff Spontane Vernetzung [27] beschrieben. Die vorgestellten Technologien setzen voraus, dass nach Anschluss eines Gerätes die unterhalb der Anwendungsschicht liegenden Schichten automatisch initialisiert sind, d. h. dass das neu angeschlossene Gerät beispielsweise eine korrekte Schicht-2-Adresse und gegebenenfalls eine korrekte Schicht-3-Adresse besitzt.

- HAVi (*Home Audio Video Interoperability*)

  HAVi ist ein Java-basierter Standard [46], der die Integration von Heimunterhaltungselektronikgeräten verschiedener Hersteller in einem Verteilten System ermöglicht. Für das Bussystem IEEE 1394 existieren Realisierungen. Eine Intention von HAVi ist, die Ressourcen der am Netz angeschlossenen Geräte zu kombinieren und dem Benutzer zur Verfügung zu stellen.

  Bei HAVi werden Geräte in verschiedene Klassen eingeteilt. In Geräte der stärksten Leistungsklasse kann Java-Code zur Laufzeit geladen werden, der dann die Fähigkeiten des Gerätes erweitert oder an die Umgebung anpasst.

  Es wird zwischen Geräten, welche diensterbringende Instanzen und Geräten, welche nur dienstnutzende Instanzen beherbergen, unterschieden. Ein Gerät mit dienstnutzenden Instanzen besitzt Software-Module (*Device Control Module*, DCM), die Geräte mit dazugehörigen diensterbringenden Instanzen repräsentieren. Es erhält die DCMs automatisch beim Anschluss eines neuen Gerätes an das Netz. Bei IEEE 1394 wird z. B. durch den Anschluss eine neue Initialisierung des Bussystems und deshalb auch des HAVi-Systems ausgelöst, die eine automatische Verteilung der DCMs einschließt.

  Im Folgenden sind weitere wichtige Eigenschaften eines HAVi-Systems aufgezählt:

  - Zum Austausch von Nachrichten wird eine auf dem Erzeuger/Verbraucher-Modell basierte Infrastruktur angeboten.
  - Diensterbringende Instanzen können Informationen über ihren Dienst in einer geräteeigenen Registratur speichern. Dienstnutzende Instanzen können auf diese Registratur zugreifen, wenn sie in einem Gerät lokalisiert sind, das eine DCM des Diensterbringergerätes besitzt.

  - HAVi ermöglicht die Verwaltung des Austauschs von A/V-Strömen. Hierbei wird beispielsweise der synchrone Übertragungsdienst von IEEE 1394 genutzt.
  - HAVi unterstützt den exklusiven Zugriff auf Geräte. Beispielsweise kann ein Videorecorder für eine gewisse Zeit reserviert werden, um einen Spielfilm aufzuzeichnen.

- UPnP (*Universal Plug and Play*)

  UPnP steht für eine Sammlung von Spezifikationen, die einen Protokollstapel für das Zusammenwirken über ein IP-Netz miteinander verbundener Geräte definieren.

  Nach dem Anschluss eines Gerätes an das Netz sendet es eine Bekanntmachungsnachricht mittels UDP-Vielfachsendens (engl. *Multicast*) mit der Angabe seiner angebotenen Dienste in das Netz aus. Jeder potenzielle Dienstnutzer, der diese Nachricht empfängt, kann sich somit eine Dienstregistratur aufbauen. Er kann auch explizit durch das Versenden von UDP-Multicast-Nachrichten nach Diensten im Netz fragen und im Erfolgsfall eine Antwortnachricht erhalten. Die Information über den angebotenen Dienst wird mit Hilfe von XML codiert, so dass auch ein menschlicher Benutzer die Information interpretieren kann. Das Protokoll, welches das Format und die Regeln des Nachrichtenaustausches festlegt, wird SSDP (*Simple Service Discovery Protocol*, [40]) genannt.

  Bei UPnP dient SOAP als Transportprotokoll zum Übermitteln von sogenannten Steuerungsnachrichten vom Dienstnutzer zum Diensterbringer. Die Gestalt einer Steuerungsnachricht wird beim Austausch der SDDP-Nachrichten vom Diensterbringer bekanntgegeben.

  Einem Diensterbringer wird zu jedem Zeitpunkt ein Zustand zugeordnet. Zustandsänderungen kann der Diensterbringer über sogenannte Ereignisnachrichten, die direkt über TCP ausgetauscht werden, seinen potenziellen Dienstnutzern bekanntgeben.

- Jini

  Jini [136] ist eine weitere Technologie, der eine große Bedeutung im Heimunterhaltungsbereich zugesprochen wird. Sie wurde Anfang 1999 als ein Java-basiertes Produkt auf den Markt gebracht. Ein Gerät kann zur Systemlaufzeit an ein IP-Netz angeschlossen werden, da durch Dienstbekanntmachungsprotokolle der Dienst im System bekannt gemacht wird.

  Zentrales Element der Jini-Architektur ist ein sogenannter LS (*Lookup Service*), bei dem die Dienste registriert werden. Ein Dienstnutzer kann dann beim LS nach einem Dienst nachfragen. Das Ergebnis der Anfrage ist ein zum Dienstnutzer transferierter Code, der z. B. auch ein RMI-Stub (siehe Abschnitt 2.1.2.2 unter RMI) sein kann.

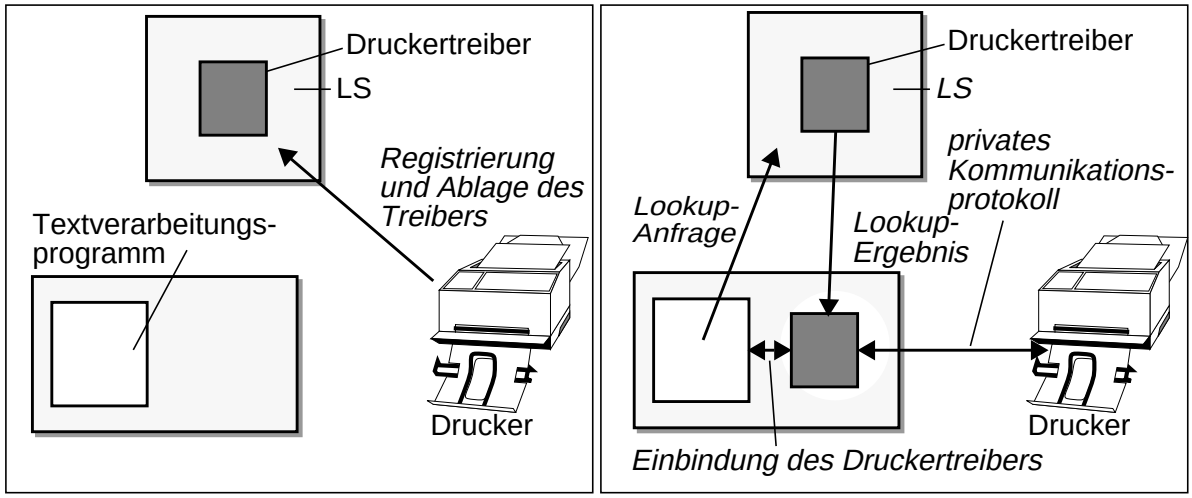


Bild 2.8: Funktion des LS von Jini

Bild 2.8 soll dieses Prinzip am Beispiel eines Jini-fähigen Druckers verdeutlichen. Der Drucker bietet einen Druckertreiber im Netz an, in dem er ihn beim LS ablegt. Ein Textverarbeitungsprogramm, das einen Druckdienst im Netz nutzen möchte, generiert eine Anfrage, und bekommt als Ergebnis den Druckertreiber geliefert. Das Zusammenwirken von Druckertreiber und Drucker kann über ein privates, optimiertes Kommunikationsprotokoll geschehen.

- Standards zur Bluetooth-Technologie

  Mit Hilfe der Bluetooth-Technologie [8] kann man drahtlose Netze aufbauen. Die relevanten Protokolle bei Bluetooth für die automatische Administration bei Hinzunahme eines Gerätes zum Netz und für die Bereitstellung eines Dienstbekanntmachungsmechanismus werden IS (*Inquiry Scan*) und SDP (*Service Discovery Protocol*) genannt. Mit Bluetooth ist eine Möglichkeit gegeben, breitbandige, drahtgebundene Heimnetze um eine schmalbandigere, drahtlose Vernetzung zu erweitern.

  Der Mechanismus von IS ist, dass suchende Geräte Pakete mit ihrem Zugangs-Code rundsenden und dass andere Geräte diese Pakete empfangen, wenn sie sich in einem Abhörzustand befinden. Diese Geräte können dann dem suchenden Gerät antworten.

  In jedem Bluetooth-Gerät, das Dienste anbietet, werden Informationen über diese Dienste in einer Registratur gehalten. Ein Client kann durch Verwendung von SDP diese Informationen erfragen.

- SLP (*Service Location Protocol*)

  SLP ist ein Standard [44] der IETF (*Internet Engineering Task Force*) zur Bereitstellung eines Dienstbekanntmachungsmechanismus, der keine bzw. nur eine minimale Administration erfordert. SLP ist vor allem für einfache Endgeräte geeignet.

- Salutation

  Salutation [110] ist eine Architektur, die mit dem sogenannten SLM (*Salutation Manager*) ein zentrales Element definiert. Ein SLM ist für alle Dienstnutzer und -erbringer seines Gerätes zuständig und bietet den sich auf anderen Geräten befindenden Instanzen eine RPC-Schnittstelle an. Über spezielle durch den SLM angebotene Dienste können ein Dienstnutzer und -erbringer eine sogenannte Sitzung einrichten und auch wieder terminieren.

  Eine automatischer Dienstbekanntmachungsmechanismus kann eingesetzt werden, wenn die verwendete RPC-Technologie Rundsenden unterstützt. Desweiteren kann Salutation mit den bisher genannten Technologien, die dann für die Dienstbekanntmachung zuständig sind, kombiniert werden.

  Durch Salutation wird eine bezüglich der verwendeteten Netztechnologie abstrakte Schnittstelle zur Verfügung gestellt. Eine Netztechnologie wird unterstützt, wenn ein entsprechender TM (*Transport Manager*) bereitgestellt wird. Der SLM greift auf Dienste dieses TMs zu.

2.2.2.2 Verwaltung von Software-Komponenten

Die Java-basierte OSGi-Architektur (*Open Services Gateway Initiative*) ist in [101] spezifiziert und soll den Zugang von Dienstnutzern, die sich in lokalen Netzen befinden, zu Diensten, die in einem globalen Internet angeboten werden, ermöglichen. Ziel ist es hierbei Komponenten, die Dienste repräsentieren, vom Internet in eine Laufzeitumgebung zu laden, und deren Aktivierung, Speicherung, Erneuerung und Entfernung zu verwalten. Eine OSGi-Komponente beinhaltet eine als Software-Bündel bezeichnete Menge von Code (u. a. auch Java-Code).

Die OSGi-Architektur ist explizit für den Heimunterhaltungsbereich aber auch für den Fahrzeugbereich vorgesehen. Da der Standard sich schwerpunktmäßig nicht mit Spontaner Vernetzung von Geräten und dem Zusammenwirken von Komponenten beschäftigt, sondern mit dem Zugang zu Diensten durch das Laden von Software-Komponenten in ein lokales Netz, ist eine Kombination mit den Middleware-Technologien zur Spontanen Vernetzung möglich und vorgesehen.

2.3 Dienste in Verteilten Systemen für die Vernetzung von Geräten der Fahrzeugtelematik

Auch in Fahrzeugen gewinnen die genannten Entwicklungen in den Bereichen der Informations- und Kommunikationstechnik eine immer größer werdende Bedeutung.

Heutige Fahrzeuge lassen sich bezüglich ihrer Elektronik und Software grob in drei Bereiche einteilen, in denen unterschiedliche Funktionalitäten bereitgestellt werden. Im Motorbereich

werden für das Fahrzeug unerlässliche Dienste erbracht, die für das Bereitstellen der Fahrfunktion notwendig sind. Im Komfort- und Luxus-Bereich werden ergänzende Mehrwertdienste (z. B. automatische Fensterverstellung oder Abspeicherung der Sitzposition) erbracht. Im Fahrzeugtelematikbereich werden Dienste erbracht, die vor allem zur Information und Unterhaltung der Insassen dienen.

Der Begriff Telematik ist hierbei ein Kunstwort, das sich aus den Begriffen Telekommunikation und Informatik zusammensetzt. Er verdeutlicht die Konvergenz dieser beiden Disziplinen und bildet den Oberbegriff für die Integration von Sprach-, Daten-, Stand- und Bewegtbildkommunikationstechnik [63]. Fahrzeugtelematik schließt auch die fahrzeugseitige Verkehrstelematik ein, durch die Mehrwertdienste zusammengefasst werden, mit deren Hilfe dem Fahrer Informationen gegeben werden, um den Verkehr zu beeinflussen. Beispielsweise sollen sie den Fahrer um einen Stau lenken.

Zunächst werden in diesem Abschnitt Anwendungen aufgezählt, die eine wesentliche Rolle auf dem Gebiet der Fahrzeugtelematik spielen. Es wird gezeigt, welche Anforderungen an die Dienstgüte gestellt werden. Schließlich wird darauf eingegangen, wie Dienste in Verteilten Systemen des Fahrzeugtelematikbereichs realisiert werden.

2.3.1 Anforderungen von Anwendungen und Systemrealisierung

Die klassische Anwendung der Fahrzeugtelematik ist der Empfang von Rundfunk, der schon seit vielen Jahren standardmäßig im Fahrzeug angeboten wird.

In Bild 2.9 ist ein typischer Aufbau bei heutigen bzw. zukunftsnahen Verteilten Systemen dargestellt. Er veranschaulicht die Zunahme des Funktionsumfangs bei der Fahrzeugtelematik. Es ist ein dezentraler Ansatz skizziert, bei dem jede Anwendung auf einem eigenen Gerät lokalisiert ist und bei dem die Sitzplatzendgeräte hauptsächlich die Benutzerschnittstellen realisieren. Durch die Pfeile ist angedeutet, dass zentralere Lösungen möglich sind, bei denen mehrere Anwendungen auf einem Endgerät (typischerweise auf einem Sitzplatzendgerät) lokalisiert sind.

2.3.1.1 Klassifikation von Anwendungen

Verschiedene Hersteller im Umfeld der Automobilzulieferer und Unterhaltungselektronikproduzenten stellen dem Fahrzeughersteller, der als Systemintegrator auftritt, Geräte für Anwendungen zum Einbau während der Fahrzeugproduktion zur Verfügung. Ein Nachrüsten ist i. A. nur begrenzt und mit viel Aufwand in der Werkstatt möglich. Die Anwendungen für die Fahrzeugtelematik können durch die in Abschnitt 2.1.1 eingeführte Klassifikation wiederum in Datendienste, interaktive A/V-Dienste und A/V-Abrufdienste strukturiert werden.

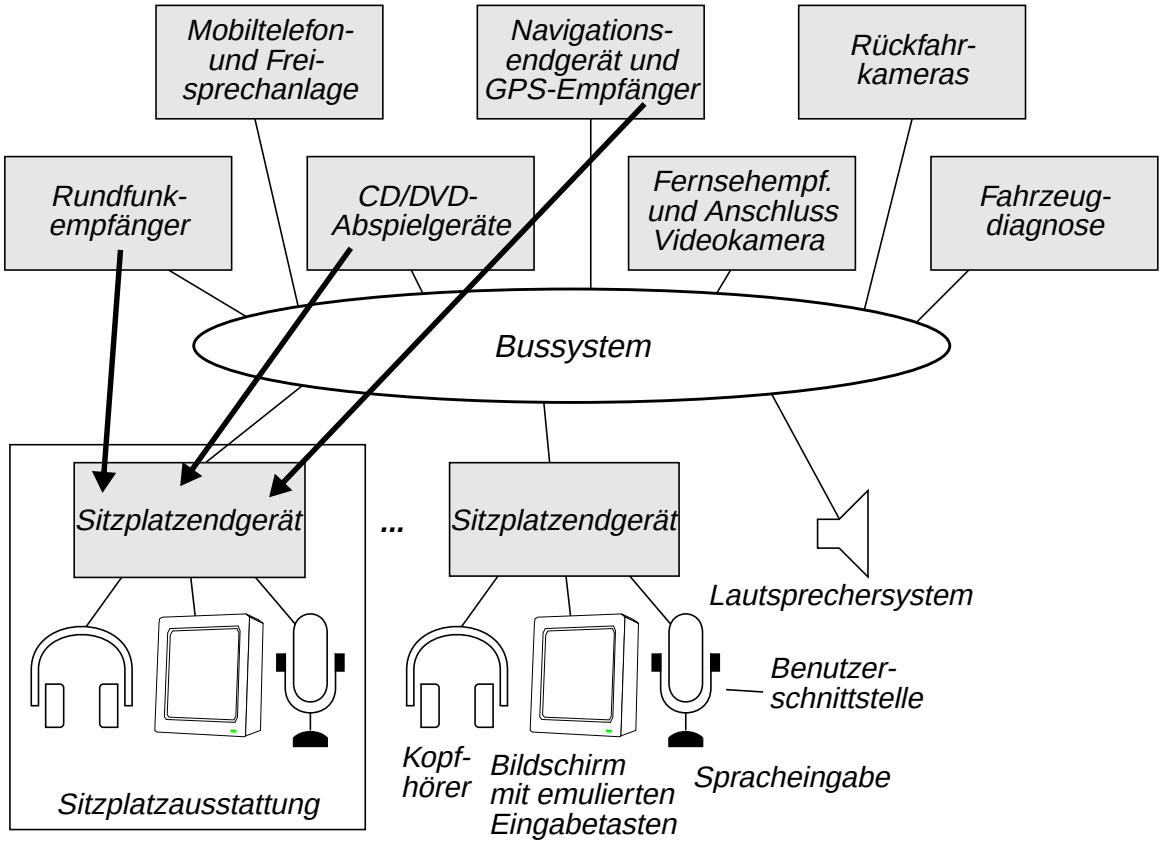


Bild 2.9: Aufbau heutiger und zukunftsnaher Verteilter Systeme für die Fahrzeugtelematik

- Datendienste

  Zu dieser Klasse gehören fahrzeugspezifische Dienste wie Navigation [55] und Fahrzeugdiagnose [26]. Zudem zählen Internet-Dienste (WWW [60] und Email [135]) dazu. In dem im Bild 2.9 eingeführten Szenario kann ein Internet-Dienst direkt auf einem Sitzplatzendgerät lokalisiert sein. Desweiteren sind Präsentationsdienste für Verkehrsnachrichten, Parkplatzinformationen [51], Informationen über nahe Sehenswürdigkeiten [135] und weitere Verkehrstelematikdienste [45, 134] zu nennen.

- Interaktive A/V-Dienste

  Hier ist vor allem die Verwendung des Mobiltelefons [55] zur Sprachkommunikation zu nennen.

- A/V-Abrufdienste

  Hierzu zählen digitaler Rundfunk, digitales TV [121] und das Abspielen von A/V-Aufzeichnungen [55] von einem CD- oder DVD-Abspielgerät oder einer Video-Kamera, sowie der Einsatz von Rückfahrkameras beispielsweise zur Unterstützung des Einparkens.

2.3.1.2 Realisierung von Verteilten Systemen für die Fahrzeugtelematik

Wegen der räumlichen Verteilung von Sensoren und Aktoren und wegen Bauraumkonflikten sind heute i. A. der Motor-, Komfort- und Fahrzeugtelematikbereich jeweils als separates Verteiltes System realisiert. Eine Kopplung der Systeme findet durch explizite Kopplungsgeräte (z. B. Gateways) statt. Zunächst wird eine Übersicht über die Motor- und Komfortbereiche gegeben. Anschließend werden die heutigen Realisierungsformen im Fahrzeugtelematikbereich vorgestellt, die Unterschiede zu denen in den beiden anderen Bereiche aufweisen.

Motor- und Komfortbereiche

In den Motor- und Komfortbereichen wird eine stetig wachsende Anzahl von Steuergeräten, die typischerweise mit einem 8 MHz Prozessor und einem Speicher von etwa 60 kByte ROM und nur 2 kByte RAM ausgestattet sind [111], miteinander vernetzt [4, 90, 126, 127]. Bussysteme, wie das weit verbreitete CAN-System (*Controller Area Network*) [108] und die möglichen Nachfolger, wie das TTP-System (*Time Triggered Protocol*) [69, 70] und das FlexRay-System [9], müssen vor allem Anforderungen bezüglich der Echtzeitbetriebsfähigkeit und Fehlertoleranz erfüllen. Gerade im Motor- und Komfortbereich kann ein Versagen des Echtzeitbetriebs zu katastrophalen Auswirkungen führen. Unter dem Begriff Fehlertoleranz wird die Gewährleistung der Erbringung einer geforderten Funktion verstanden, auch wenn sich eine Systemeinheit fehlerhaft verhält bzw. ausfällt. Um dies zu ermöglichen, müssen in der Regel Redundanzen in das System eingefügt werden.

Fahrzeugtelematikbereich

Die Anforderungen im Fahrzeugtelematikbereich unterscheiden sich wesentlich von den Anforderungen in den Motor- und Komfortbereichen. Die Anwendungen dieses Bereichs verursachen eine höhere Bitrate, womit leistungsstärkere Endgeräte und breitbandigere Bussysteme vorgesehen werden müssen. Insbesondere der Austausch von A/V-Strömen erfordert, dass eine Übertragungskapazität von mehreren Mbit/s bereitgestellt werden muss.

Früher wurden Geräte der Fahrzeugtelematik direkt miteinander verdrahtet. Mit der Zunahme des Funktionsumfangs geht jedoch ein Anwachsen der Komplexität bei der Verkabelung einher. Die Komplexität bezieht sich nicht nur auf die stetig wachsende Kabelbaumlänge, sondern auch auf Unterschiede bei Kabelkonfektionen und -steckern. Aus diesen Gründen wurde auch in diesem Bereich eine Vernetzung der Geräte und Komponenten durch Bussysteme angestrebt.

Da aus wirtschaftlichen Überlegungen die Endgeräte möglichst wenig Pufferspeicher besitzen sollen, wird eine Übertragung von A/V-Strömen mit konstanter Rate [122] über einen synchronen Übertragungsdienst angestrebt. Der Takt in den Endgeräten ist hierbei der konstanten Rate so angepasst, dass innerhalb eines Taktzyklus die Daten verarbeitet und auf einem Bildschirm

oder Lautsprecher ausgegeben werden können, ohne eine Zwischenspeicherung zum Ausgleich von Verzögerungsschwankungen vornehmen zu müssen.

Alle hier vorgestellten Bussysteme besitzen als Gemeinsamkeit, dass sie zwei verschiedenartige Übertragungsdienste anbieten:

- Asynchroner Übertragungsdienst: Für die gesicherte Übertragung der Daten werden keine zeitlichen Garantien gegeben. Eine fehlerhafte Übertragung kann beim Sender festgestellt werden. Gegebenenfalls wird wiederholt versucht, fehlerfrei zu übertragen.
- Synchroner Übertragungsdienst: Da für die Übertragung von Daten eine fest reservierte Übertragungskapazität zur Verfügung steht, können zeitliche Garantien gegeben werden. Fehlerhafte Übertragungen werden nicht festgestellt.

Im Folgenden werden einige Kandidaten für die Realisierung eines Bussystems für die Fahrzeugtelematik vorgestellt, die die geforderten Dienstgüten berücksichtigen.

- D2B (*Digital Domestic Bus*)

  Der Automobilhersteller DaimlerChrysler baut seit 1998 das D2B-System [18, 102] serienmäßig in Fahrzeuge der gehobenen Klasse ein. Bei einem D2B-System sind die Geräte durch Lichtwellenleiter (Plastikfasern) ringförmig vernetzt. Neben einem asynchronen Übertragungsdienst für die Übertragung von Steuerungsdaten steht ein synchroner Übertragungsdienst für die Übertragung von Audio-Strömen bereit. Mit Hilfe des D2B-Systems können vier CD-Audio-Ströme (je 1,41 Mbit/s) gleichzeitig übertragen werden. Die Übertragung von Video-Strömen ist nicht vorgesehen.

- MOST (*Media Oriented Systems Transport*)

  MOST [68, 102] wurde als D2B-Nachfolgesystem entwickelt. Wie bei D2B werden Geräte durch Lichtwellenleiter in der Regel zu einer Ringtopologie zusammengefügt. Auch hier existiert eine Unterscheidung in einen asynchronen und einen synchronen Übertragungsdienst. Die Gesamtbitrate von MOST beträgt 22,1 Mbit/s. Neben der Übertragung von Audio-Strömen ist nun auch die Übertragung von Video-Strömen vorgesehen.

- IDB-1394

  Die in Abschnitt 2.3.1.2 eingeführte IEEE 1394-Technologie ist ebenfalls ein Kandidat für den Fahrzeugeinsatz. Sie wird in der Fahrzeugtelematikwelt IDB-1394 (*Intelligent Transportation System Data Bus*) genannt.

2.3.2 Nutzung, Erbringung und Verwaltung von Diensten

Früher wurden Anwendungen der Fahrzeugtelematik monolithisch in einem Gerät realisiert. In heutigen Verteilten Systemen für die Fahrzeugtelematik ist eine Strukturierung in dem Sinne

erkennbar, dass verschiedene Zulieferer einen definierten Funktionsumfang in ihren Geräten bereitstellen, die dann vom Fahrzeughersteller durch Anbinden an ein Bussystem zu einem Gesamtsystem integriert werden [128]. Die Geräte können mehrere Systemdienste und Anwendungen beinhalten.

2.3.2.1 Statische Zuordnung von Funktionen an Endgeräte

Die im Verteilten System vorzufindenden Anwendungen und Systemdienste sind heute i. A. statisch einem Gerät zugeordnet, d. h. sie stehen im Fahrzeug bereit, wenn während der Fahrzeugproduktion ein entsprechendes Gerät eingebaut wurde. Allgemein gilt, dass aus Systemintergratorsicht die zu betrachtenden Schnittstellen sich auf diejenigen zwischen Endgerät und Bussystem reduzieren. Die in Abschnitt 2.1.3 vorgestellten Verwaltungsdienste werden i. A. nicht benötigt, da das System vom Systemintegrator statisch konfiguriert wird. Somit ist z. B. das Entdecken eines zur Laufzeit ins System kommenden Dienstes nicht notwendig.

Da an einem Endgerät, das die Benutzerschnittstelle realisiert, alle zur Präsentation notwendigen Daten zusammengeführt werden, nimmt es eine herausragende Stellung unter den eingebauten Endgeräten ein. So wird auf diesem Gerät angestrebt, verschiedene Bildquellen zu einer einheitlichen Bilddarstellung zu integrieren. Bildinhalte (Bewegtbilder, Anzeigefunktionen, etc.) sollen hierbei auf einem Bildschirm nach zur Laufzeit generierten Benutzerwünschen zusammensetzbar sein. Neben der Bereitstellung eines Bildschirms (gegebenenfalls mit emulierten Tasten zur Benutzereingabe), eines speziellen Grafikprozessors und dementsprechender Hardware zur Dekodierung von beispielsweise MPEG-Strömen ist ein nicht spezialisierter Prozessor vorgesehen, auf dem die Software zur Präsentationsverwaltung ablaufen kann [6, 132]. Wegen der hohen Leistungsfähigkeit dieses Prozessors und seiner Konzentratorfunktion beim Datenaustausch, ist es oftmals lohnenswert das Verteilte System in dem Sinne zu zentralisieren, dass verschiedene Anwendungen (z. B. Navigation, Radioempfänger und WWW-Browser) direkt auf einem Sitzplatzendgerät ablaufen (siehe auch die Pfeile in Bild 2.9).

Bisher war im Fahrzeug nur eine einzige Benutzerschnittstelle vorgesehen, die für den Fahrer einsehbar im Vorderraum lokalisiert wurde. Bei den neuesten Entwicklungen wird angestrebt, an jedem Sitzplatz eine Benutzerschnittstelle bereitzustellen. Zu berücksichtigen ist, dass eventuell auftretende Ressourcen-Zugriffskonflikte im Verteilten System nicht mehr durch eine lokale Präsentationsverwaltung gelöst werden können, sondern dass zusätzlich die Einführung einer globalen Präsentationsverwaltung notwendig wird.

2.3.2.2 Komponenten-Software – Ein Evolutionsszenario für die Fahrzeugtelematik

In diesem Abschnitt wird diskutiert, wie ausgehend von den jetzigen Lösungen zukünftige Verteilte Systeme für die Fahrzeugtelematik gestaltet sein können. Hierbei wird angenommen,

dass zukünftig in diesem Bereich Komponenten-Software eine herausragende Bedeutung besitzen wird. Dass ein solches Szenario in zukünftigen Verteilten Systemen für die Fahrzeugtelematik möglich ist, zeigt sich in dem heutigen, vereinzelten Vorhandensein von Java-Komponenten. So sind Fahrzeugtelematiksysteme, die der in [120] vorgestellten, Java-beinhaltenden Software-Architektur TLA (*Top Level Architecture*) folgen, Teil von heutigen Fahrzeugen der Komfort- und Luxusklasse. Innerhalb dieser Architektur ist ein dynamisches Verwalten von Java-Komponenten durch OSGi spezifiziert.

Motivation für die Einführung von Komponenten-Software

Über die Frage, wie zentral bzw. dezentral ein Verteiltes System für die Fahrzeugtelematik aufgebaut wird, entscheiden heutzutage nicht nur technische Aspekte. Zudem müssen auch die bestehenden Strukturen zwischen dem Fahrzeughersteller und den Zulieferern berücksichtigt werden. Bei einer möglichen Zentralisierung, bei der das Sitzplatzendgerät eine führende Rolle übernimmt, können einige Probleme auftreten. Da das Endgerät vom Systemintegrator als „Black Box“-Produkt anzusehen ist, muss zunächst die Frage beantwortet werden, ob ein einzelner Zulieferer überhaupt die geforderte Funktionspalette alleine in seinem Endgerät integrieren kann. Zudem sind bei einem Zuliefererwechsel die realisierten Funktionen in der Regel nicht mehr wiederverwendbar und müssen vom neuen Zulieferer nochmals implementiert werden. Es existieren Bestrebungen, diesen Problemen entgegenzuwirken, indem die Produktgranularität in dem Sinne geändert wird, dass nicht nur die Endgeräte sondern auch die einzelnen Funktionen in Form von Software-Komponenten die Produkte bilden [31, 112].

Ein weiterer Grund für das Einführen von Komponenten-Software ist in der Dynamik neuer aufkommender Fahrzeugtelematikfunktionen zu sehen. So enthält die in Abschnitt 2.3.1.1 betrachtete Menge heutige und zukunftsnahe Anwendungen für die Fahrzeugtelematik. Niemand kann jedoch eine genaue Aussage über die in Zukunft im Fahrzeug gewünschten Anwendungen geben. Die Innovationszyklen bei der Informations- und Kommunikationstechnik sind so kurz, dass Anwendungen in der Regel während der Lebensdauer eines Fahrzeuges veralten. Aus diesem Grund soll in Zukunft die einfache Austauschbarkeit von Diensten während der Fahrzeuglebensdauer gewährleistet werden können.

Neben der Bereitstellung unterschiedlicher Dienstgüten für die einzelnen Dienste und der Wiederverwendbarkeit schon realisierter Funktionen ist die Austauschbarkeit von Diensten eine Hauptanforderung an ein zukünftiges Fahrzeugtelematiksystem [28]. Deshalb wird erwartet, dass die heute statisch aufgebauten Verteilten Systeme um dynamische Austauschmechanismen erweitert werden. Eine Möglichkeit ist, die Dienste durch Software-Komponenten zu realisieren. Durch eine Konfigurationsverwaltung können dann bei neuen Dienstwünschen der Benutzer die entsprechenden Komponenten ins Fahrzeug geladen und das Verteilte System neu konfiguriert werden.

Damit stellt die Fahrzeugtelematik ein zukünftiges Einsatzgebiet für komponentenbasierte Verteilte Systeme mit einer Konfigurationsverwaltung dar. Dieses Szenario soll als Beispiel dienen, an dem die folgenden, theoretischen Überlegungen dieser Arbeit validiert werden.

Etablierung von Middleware- und Komponenten-Software-Technologien

Die in Abschnitt 2.1 vorgestellten Technologien für die Unternehmensvernetzung verwenden einheitlich IP. In der heutigen Fahrzeugtelematik werden Bussysteme wie MOST und D2B eingesetzt, die eine Reservierung fester Übertragungskapazitäten für A/V-Anwendungen durch das Bereitstellen synchroner Übertragungsdienste ermöglichen. Eine Etablierung von Middleware-Technologien aus dem Bereich der Unternehmensvernetzung ist bei einer Konvergenz von Mechanismen zur Verwaltung von Übertragungskapazitäten für A/V-Anwendungen und Internet-basierten Middleware-Technologien, die den Software-Instanzen in der Anwendungsschicht prinzipiell nur asynchrone Transportsdienste bereitstellen, möglich.

- Durch Verwendung von Prioritäten je Ethernet-Rahmen und Verwaltungsintanzen, wie z. B. SBMs (siehe Abschnitt 2.1.1.2), können auch in Ethernet-basierten lokalen Netzen Übertragungskapazitäten garantiert werden. Damit könnten die heute etablierten Bussysteme wie D2B oder MOST durch Ethernet ersetzt werden, wenn man bei diesen Überlegungen die Problematik der elektromagnetischen Verträglichkeit außer acht läßt.
- Desweiteren ist auch eine weitere Verwendung der heute etablierten Bussysteme wie D2B oder MOST denkbar, wobei durch die Integration eines Internet-Protokollstapels Internet-basierte Middleware-Technologien Anwendung finden können.

Für Komponentenablaufumgebungen aus dem Bereich der Unternehmensvernetzung gilt zum einen, dass die Standards für die Internet- und Intranet-Vernetzung die Anforderungen zur Verwaltung von Fahrzeugtelematik-Komponenten nicht erfüllen. So fehlt ihnen beispielsweise eine Persistenzverwaltung. Zum anderen sind Umgebungen, die vollständig nach den Standards für mehrstufige Architekturen in Unternehmensnetzen aufgebaut sind, hingegen so komplex, dass sie sehr leistungsstarke Endgeräte, die für Verteilte Systeme der Fahrzeugtelematik aus Kostengründen untypisch sind, benötigen.

Im Gegensatz dazu sind die in Kapitel 2.2 vorgestellten Technologien zur Heimvernetzung interessante Kandidaten für den Einsatz in Verteilten Systemen für die Fahrzeugtelematik.

- HAVi bietet die Möglichkeit, die Dienstgüteanforderungen von A/V-Anwendungen durch direkte Nutzung des synchronen Übertragungsdienstes zu berücksichtigen.
- Mehrere Technologien bieten die Möglichkeit, für den Benutzer transparent Dienste zu verwalten und somit die Austauschbarkeit von Anwendungen zu ermöglichen.

- Technologien wie z. B. HAVi und Bluetooth ermöglichen den Anschluss zusätzlicher Geräte mit Dienstnutzern und -erbringern während der Fahrzeuglebensdauer durch Bereitstellung von Mechanismen zur Spontanen Vernetzung.
- Wenn Dienste in Form von Software-Komponenten in das Verteilte System geladen werden, bietet die OSGi-Technologie Mechanismen zur Komponentenverwaltung an. Zur Bekanntmachung neu hinzugekommener Dienste können wiederum Technologien wie UPnP, Jini, Salutation oder SLP verwendet werden.

Vielversprechend ist eine Kombination z. B. aus OSGi, HAVi und Jini. Alle drei genannten Technologien basieren auf Java. Im weiteren Verlauf werden daher Java-Laufzeitumgebungen als Ablaufumgebungen für Software-Komponenten betrachtet. Zur Realisierung von Komponentenschnittstellen werden Internet-basierte Middleware-Technologien wie CORBA oder Java RMI, die eine Komponentenkooperation über Methodenfernaufrufe erlauben, in Betracht gezogen.

Komponentenablaufumgebungen – Trennen der Software von der Hardware

Bei dem in dieser Arbeit näher betrachteten Evolutionsszenario werden die Bussysteme, die asynchrone und synchrone Übertragungsdienste unterstützen, auch in Zukunft erhalten bleiben. Anwendungen und Systemdienste werden durch Software-Komponenten realisiert, die auf Ablaufumgebungen verwaltet werden. Sie können auf diese von einer externen Infrastruktur zur Systemlaufzeit geladen werden. Hierdurch wird zusammen mit dem Einsatz von Mechanismen zum Anschluss von Geräten zur Systemlaufzeit die Erfüllung der Anforderung nach Austauschbarkeit und Erneuerbarkeit von Anwendungen erreicht. Mögliche Orte für solche Ablaufumgebungen sind die in Abschnitt 2.3.2.1 eingeführten Sitzplatzendgeräte. Die Realisierung eines Szenarios mit einem zentralen, leistungsstarken Endgerät, das eine im Fahrzeug einzig vorhandene Komponentenablaufumgebung trägt, ist ebenfalls möglich. Die Grundlage für die folgenden Betrachtungen bildet jedoch die Realisierung eines Komponenten-Software-Systems durch Ablaufumgebungen, die auf mehrere Sitzplatzendgeräte verteilt sind.

2.3.2.3 Realisierung eines Sitzplatzendgerätes

In Bild 2.10 wird eine mögliche Realisierung eines Sitzplatzendgerätes vorgestellt, wobei die schematische Darstellung sich an Mikro-Controllern, wie sie in [33, 85] beschrieben sind, orientiert. Sowohl der interne Bus als auch der Peripheriebus besteht im Gegensatz zum seriellen Bussystem aus parallelen Steuer-, Adress- und Datenleitungen. Am internen Bus sind ein Mikroprozessor und eine Speichereinheit angeschlossen. Eine Peripherieschnittstelle ermöglicht den Zugang zum Peripheriebus z. B. durch Einsatz des Direktspeicherzugriffverfahrens (DMA, *Direct Memory Access*). Am Peripheriebus sind eine Eingabeeinheit, u. a. zur Verarbeitung von Spracheingaben oder von Eingaben über am Bildschirm emulierte Druckknöpfe,

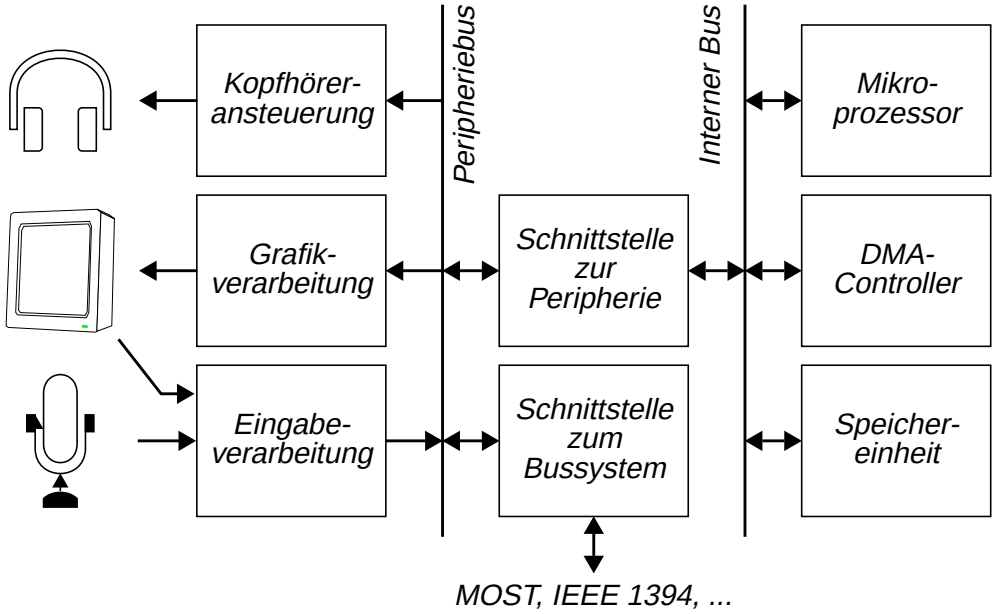


Bild 2.10: Schematische Darstellung der Realisierung des Sitzplatzendgerätes

sowie Ausgabeeinheiten zur Verarbeitung von Grafikdaten und zur Kopfhöreransteuerung angeschlossen. Die Verarbeitung von Grafikdaten in einer Ausgabeeinheit kann, wie in [85] erläutert wird, den Mikroprozessor entlasten. So ist es möglich, dass die eigentliche Verarbeitung der Grafikdaten in einem dort zugeordneten Grafikverarbeitungsprozessor stattfindet. Die Schnittstelleneinheit für das Bussystem ist wie bei [85] dem Peripheriebus zugeordnet.

Es gibt beim Austausch von Daten zwischen einem Sitzplatzendgerät und anderen Endgeräten zwei wesentliche Dienstgütekriterien zu beachten:

- A/V-Ströme, die auf dem Bussystem mit Hilfe eines synchronen Busdienstes übertragen werden, gelangen über den Peripheriebus zu den beiden Peripheriegeräten Bildschirm und Kopfhörer. Bei der in Bild 2.10 dargestellten Mikro-Controller-System müssen nur die Ressourcen der Bussystemschnittstelleneinheit, der Ausgabeeinheiten und des Peripheriebusses, jedoch nicht die Ressourcen des Mikroprozessors berücksichtigt werden.

  Im Umfeld dieser Arbeit [71] wurde eine prototypische Implementierung eines Audio-Systems mit Hilfe eines Mikro-Controller-Systems vorgenommen, bei dem die Einheit der Bussystemschnittstelle nicht an den Peripheriebus angeschlossen, sondern direkt an die Speichereinheit gekoppelt ist. Bei der dort gewählten, Ressourcen-aufwändigeren Lösung findet eine durch DMA realisierte, logische Abbildung der Schnittstelleneinheiten auf Bereiche in der Speichereinheit statt. Hierbei muss der Mikroprozessor eingesetzt werden, um Daten zwischen den beiden Bereichen auszutauschen.

- Daten, die auf dem Bussystem mit Hilfe des asynchronen Übertragungsdienstes übertragen werden, müssen in einer logischen Sichtweise zwischen Komponenten und in einer physikalischen Sichtweise zwischen entsprechenden Speichereinheiten ausgetauscht werden, da

sie den dort gespeicherten Datenzustand einer Komponente verändern. Bei dem Datenaustausch muss bei Verwendung des exemplarischen Mikro-Controller-Systems in Bild 2.10 auf die Ressourcen der Bussystem- und Peripherieschnittstelleneinheit, der Speichereinheit und auf die Ressourcen des Peripherie- und des internen Busses zugegriffen werden. Für eine Aufbereitung der ausgetauschten Daten muss zudem der Mikroprozessor belastet werden. Dies liegt daran, dass die benötigte Funktionalität zur Wandlung der mittels des Bussystems ausgetauschten seriellen Byte-Folgen in typisierte Daten und deren Rückwandlung gemäß dem OSI-Modell in der Transport- und der Darstellungsschicht (Schichten 4 und 6) bereitgestellt und in Software realisiert wird.

Kapitel 3

Konfiguration von komponenten-basierten Verteilten Systemen

In diesem Kapitel werden die Begriffe Konfiguration und Rekonfigurierung eingeführt. Nach einer Definition der Begriffe werden exemplarisch Konfigurationsverwaltungskonzepte für komponentenbasierte Verteilte Systeme in der Fahrzeugtelematik vorgestellt. Insbesondere wird ein Java-basiertes Komponenten-Software-System mit einer solchen Konfigurationsverwaltung näher betrachtet.

Anschließend wird in diesem Kapitel die Zerlegung einer monolithischen, benutzergesteuerten Anwendung in Komponenten untersucht. Die Betrachtungen führen zu einer Methode zur Modellierung von Konfigurationen Verteilter Systeme, die solche Anwendungskompositionen beinhalten.

3.1 Konfiguration und Rekonfigurierung

Die in dieser Arbeit betrachteten Verteilten Systeme beinhalten auf Endgeräten lokalisierte Ablaufumgebungen für Software-Komponenten, wobei die Endgeräte durch mindestens ein Bussystem miteinander gekoppelt sind. Eine Software-Komponente kann, wie in Abschnitt 2.1.4.1 definiert, zur Systemlaufzeit in eine Ablaufumgebung geladen und gegebenenfalls über eine Verwaltungsschnittstelle aktiviert werden. Über eine Datenschnittstelle kann sie mit anderen, zu ihr gebundenen Komponenten kooperieren. Unter dem Begriff Konfiguration wird in einer ersten Sicht der Zustand eines Verteilten Systems aufgrund der geladenen Software-Komponenten und ihren Bindungen verstanden. Der Begriff Rekonfigurierung beschreibt das Ändern eines solchen Zustandes.

3.1.1 Konfigurationsebenen

Die Begriffe Konfiguration und Rekonfigurierung können in einer näheren Betrachtung unterschiedliche Bedeutung besitzen.

3.1.1.1 Klassifikation der Begriffe Konfiguration und Rekonfigurierung

In [53] werden drei Konfigurationsebenen eingeführt:

- Eine strukturelle Konfiguration entspricht einem Graphen (siehe Bild 3.1), so dass in einer logischen Sichtweise Knoten die Komponenten und Kanten deren Bindungen repräsentieren. Eine strukturelle Rekonfigurierung entspricht dem Ändern dieses Graphen.
- Eine topologische Konfiguration entspricht einer Abbildung der logischen, strukturellen Konfiguration auf die physikalische Topologie des Verteilten Systems. Diese physikalische Topologie kann wiederum als Graph dargestellt werden, mit Ablaufumgebungen als Knoten. Eine Kante beschreibt, ob zwei Ablaufumgebungen durch ein Bussystem direkt miteinander verbunden sind (siehe Bild 3.1). Die Abbildung ist folglich eine Zuordnung der Komponenten und deren Bindungen auf die im Verteilten System bereitgestellten Ressourcen. Eine Komponente kann den physikalischen Ressourcen von nur einer Ablaufumgebung und eine Bindung kann den physikalischen Ressourcen von mehreren Bussystemen und Ablaufumgebungen zugeordnet werden. Eine topologische Rekonfigurierung entspricht

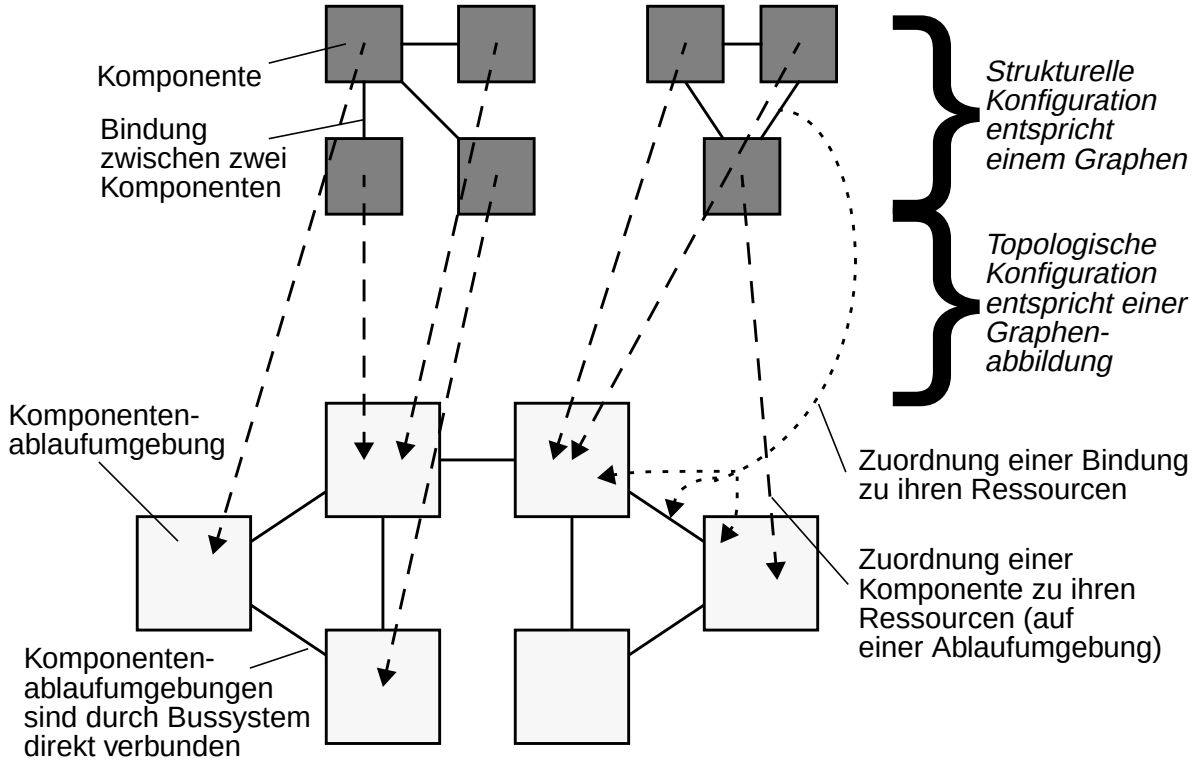


Bild 3.1: Logische und topologische Konfiguration

dem Ändern der Abbildung des Konfigurationsstrukturgraphen auf den Graphen, der die physikalische Topologie beschreibt.

- Eine komponenteninterne Konfiguration legt die Gestalt einer im Verteilten System aktiven Komponente in Form ihrer Implementierung fest. Unter einer komponenteninternen Rekonfigurierung wird dann das Ändern ihrer Implementierung während ihrer Aktivität verstanden.

3.1.1.2 Gründe für eine Rekonfigurierung

In diesem Abschnitt werden die Gründe für eine Rekonfigurierung für die verschiedenen Ebenen vorgestellt.

- Eine strukturelle Rekonfigurierung wird aus folgenden Gründen durchgeführt:
  - Während der Initialiserungsphase werden neue Komponenten hinzugenommen und Bindungen erstellt, bis der Initialzustand erreicht ist.
  - Bei Deaktivierung des Systems werden alle Komponenten entfernt und alle Bindungen aufgelöst.
  - Komponenten, die einen benötigten Dienst erbringen sollen, werden zur Systemlaufzeit geladen und gegebenenfalls aktiviert.
  - Komponenten, deren Dienste nicht mehr benötigt werden, werden zur Systemlaufzeit entfernt.
  - Um einen Dienst einer anderen Komponente zu nutzen, muss eine Komponente zur Systemlaufzeit diese an sich binden. Nach der Nutzung kann die Bindung wieder gelöst werden.
- Eine topologische Rekonfigurierung wird aus folgenden Gründe durchgeführt:
  - Jede strukturelle Rekonfigurierung löst eine topologischen Rekonfigurierung aus.
  - Eine topologische Rekonfigurierung wird vorgenommen, um Last entsprechend vorgegebener Kriterien zu verteilen.
  - Eine topologische Rekonfigurierung kann wegen einer temporären Änderung des Angebots an physikalischen Ressourcen durchgeführt werden. So kann sie z. B. eine Reaktion auf den Ausfall von Ressourcen sein.
- Eine komponenteninterne Rekonfigurierung wird vorgenommen, um das Programm von Komponenten, die über einen langen Zeitraum aktiv sind, während ihrer Aktivität zu ersetzen. Eine solche Rekonfigurierung wird oftmals zur Fehlerbeseitigung in Programmen durchgeführt.

3.1.1.3 Statische und dynamische Abhängigkeiten

Eine erfolgreiche Rekonfigurierung ist nur bei Berücksichtigung von Abhängigkeiten, die eine betroffene Komponente zu ihrer Umgebung besitzt, möglich. Man unterscheidet zwischen statischen und dynamischen Abhängigkeiten:

- Statische Abhängigkeiten

  Es existieren grundsätzliche Anforderungen einer Komponente, deren Erfüllung notwendig ist, damit während einer Aktivität einer Komponente keine Fehler auftreten. Hardware-Ressourcen und durch Software-Komponenten bereitgestellte Dienste müssen zugreifbar sein und entsprechende QoS-Anforderungen müssen befriedigt werden. Alle grundsätzlich möglichen Anforderungen, die während einer Aktivität auftreten können, können für jede Komponente statisch benannt werden. Insbesondere können alle statischen Datenschnittstellen (z. B. Schnittstellen mit Stubs und Skeletons), die eine Komponente anbietet bzw. die von anderen Komponenten für sie angeboten werden, statisch benannt werden. Eine Komponentenverwaltung kann mögliche Fehlerzustände während der Aktivität einer Komponente vermeiden, indem nur ein Laden auf eine Zielablaufumgebung erlaubt wird, auf der diese grundsätzlichen Anforderungen berücksichtigt werden können.

- Dynamische Abhängigkeiten

  Wenn eine Komponente aktiv ist, kann sie mit anderen Komponenten kooperieren. Hierzu müssen die Komponenten zueinander gebunden sein und für diese Bindungen müssen Ressourcen reserviert werden. Eine Rekonfigurierung, die eine gebundenen Komponente miteinschließt, kann zu Konsistenzverletzungen führen, wenn die Ressourcenzuordnungen nach der Migration Fehler verursachen. Unter Konsistenz versteht man hier, dass bei jeder Komponente das lokal gespeicherte Abbild des für sie sichtbaren Teils des Systemzustands nicht widersprüchlich zum tatsächlichen Systemzustand ist.

Die Unterscheidung in statische und dynamische Abhängigkeiten hat Auswirkungen auf den Konfigurationsbegriff: Im Folgenden wird die Gestalt eines Verteilten Systems aufgrund der vorhandenen Komponenten und deren Bindungen, wie sie in Abschnitt 3.1.1.1 betrachtet werden, als bindungsstrukturelle Konfiguration bezeichnet. Durch die gegebenen statischen Abhängigkeiten können alle Bindungen, die aufgrund der statischen Schnittstellen der vorhandenen Komponenten möglich sind, benannt werden. Die Gestalt eines Verteilten Systems aufgrund der vorhandenen Komponenten und aller möglichen Bindungen wird in dieser Arbeit als schnittstellenstrukturelle Konfiguration bezeichnet.

In [66, 67] werden für diese beiden Aspekte die Begriffe statische und dynamische Abhängigkeiten eingeführt. Es wird gefordert, dass sowohl die statischen als auch die dynamischen Abhängigkeiten einer Konfigurationsverwaltung bekannt sein müssen, damit sie eine fehlervermeidende und konsistente Rekonfigurierung vornehmen kann.

3.1.2 Topologische Rekonfigurierung durch Komponentenmigration

Der Schwerpunkt dieser Arbeit liegt auf der Betrachtung der topologischen Rekonfigurierung aufgrund von Lastverteilungsgesichtspunkten, bei der der Programm-Code und der Datenzustand einer Komponente zu einer Zielablaufumgebung verschoben wird. Dieses Verschieben einer Komponente wird auch als Komponentenmigration bezeichnet. In dieser Arbeit wird unter dem Begriff der Komponentenmigration ein Vorgang verstanden, bei dem eine Komponente von einer Ausgangsablaufumgebung entfernt und danach auf eine Zielablaufumgebung geladen und gegebenenfalls aktiviert wird.

3.1.2.1 Definition der Migrationsstärke

Die Begriffe Starke und Schwache Migration sind vor allem durch die Verbreitung sogenannter Agententechnologien [36, 109] geprägt:

- Unter Schwacher Migration versteht man das alleinige Verschieben des Codes und des Datenzustandes einer Komponente von einer Ausgangs- zu einer Zielablaufumgebung. Die in Abschnitt 2.1.2.2 vorgestellte Middleware-Technologie Java RMI ermöglicht beispielsweise eine schwache Migration von Java-basierten Komponenten.
- Unter Starker Migration versteht man das Verschieben des Codes, des Datenzustandes und des Aktivitätszustands einer Komponente. Ein Aktivitätszustand ist durch die Ablaufzustände der Bearbeitungsvorgänge (Prozesse) bestimmt. Zur Bestimmung eines Ablaufzustands ist die Betrachtung des Aufrufstapels notwendig, mit dem die Reihenfolge der Routinen-, Prozedur- oder Methodenaufrufe festgehalten wird. Desweiteren werden in einem Aufrufstapel die lokalen Daten dieser Routinen, Prozeduren oder Methoden gespeichert. Der Ablaufzustand ergibt sich zudem aus dem Befehlsfolgezähler, durch den die Stelle der Abarbeitung in der aktuellen Routine, Prozedur oder Methode festgehalten wird.

Oftmals werden in Arbeiten, die sich auf das objektorientierte Paradigma beziehen (z. B. in [59]) alternative Begriffe verwendet. Es wird die Schwache Migration auch als Objektmigration und die Starke Migration als Prozessmigration bezeichnet.

In [35] wird eine detaillierte Klassifikation von Migrationsmechanismen vorgestellt. Dabei wird unter Migration das Verschieben des Codes und eventuell des Aktivitätszustands zu einer anderen Ablaufumgebung verstanden. Zudem wird darauf eingegangen, dass ein Austausch des Datenzustandes nicht nur durch ein Kopieren oder Verschieben der Daten von der Ausgangs- zur Zielablaufumgebung erreicht werden kann. So ist es u. a. auch möglich, dass die Daten bei der Ausgangsablaufumgebung bleiben und dass über eine im Verteilten System eindeutige Referenz von der Ziel- auf Daten auf der Ausgangsablaufumgebung zugegriffen werden kann.

3.1.2.2 Konsistente Migration durch Beachtung der dynamischen Abhängigkeiten

Im Folgenden werden Mechanismen für eine konsistente Migration vorgestellt. Hierbei werden aus einer Vielzahl von Arbeiten [5, 16, 17, 53, 67, 73] zwei Arbeiten genauer vorgestellt: Die Arbeit von Kramer [73] ist als grundlegender Beitrag auf diesem Gebiet anzusehen. Die in der Arbeit von Chen [16, 17] vorgestellten Mechanismen bilden die Basis der in dieser Arbeit vorgenommenen Betrachtungen zur Komponentenmigration. Beide Arbeiten stellen Mechanismen vor, bei denen während der Migration die Kooperation der betroffenen Komponente mit anderen ausgeschlossen wird. Auf diese Weise wird gewährleistet, dass keine Inkonsistenzen auftreten.

Neben schon eingeführten Begriffen wie Aufruf, Beauftragung und Bindung werden in diesem Abschnitt noch die Begriffe Transaktion und Referenz verwendet. Eine Transaktion besteht aus einem mehrfachen Datenaustausch zwischen zwei zueinander gebundenen Komponenten. Es wird angenommen, dass eine Transaktion nach endlicher Zeit beendet ist. Ein Prozedur- oder Methoden(-fern-)aufruf, der im Folgenden nur als Aufruf bezeichnet wird, ist eine spezielle Realisierung einer Transaktion mit einem zweimaligen Austausch von Daten. Übergabeparameter werden von der dienstnutzenden zu der diensterbringenden Komponente hin- und Ergebnisparameter zurücktransferiert. Die erforderliche Bindung wird direkt vor der Bearbeitung des Aufrufs eingerichtet und sofort nach der Bearbeitung wieder gelöst. Eine Referenz entspricht einer Benennung der Lokalität einer Komponente. Eine Komponente muss beim Transferieren von Daten die Referenz der Zielkomponente kennen.

- Konfigurationsverwaltung bei Kramer

  Diese Arbeit spielt bei den Konsistenzbetrachtungen bezüglich der Migration eine wesentliche Rolle, da Kramer grundsätzlich die Konsistenz von atomischen Vorgängen betrachtet, aus denen sich eine Migration zusammensetzt. Diese Vorgänge entsprechen dem Entfernen einer Komponente auf der Ausgangsumgebung und dem Erzeugen einer Komponente auf der Zielumgebung.

  Von Kramer werden hierbei Operationen (`erzeuge()`, `entferne()`, `binde()` und `löse()`) vorgestellt, die das Erzeugen und Entfernen einer Komponente und das Binden und das Auflösen einer Bindung zwischen zwei Komponenten zur Systemlaufzeit ermöglichen. Die Operationen können von einer Instanz, die für die Konfigurationsverwaltung verantwortlich ist, initiiert werden und in einer Komponente ausgeführt werden (siehe Bild 3.2).

  Die Konfigurationsverwaltung initiiert das Binden immer von einer Komponente aus, und diese Initiative wird von einer anderen Komponente entgegengenommen. In Bild 3.2 ist eine Bindung als Kante eines Graphen dargestellt, die von einem Knoten, der die Initiator-

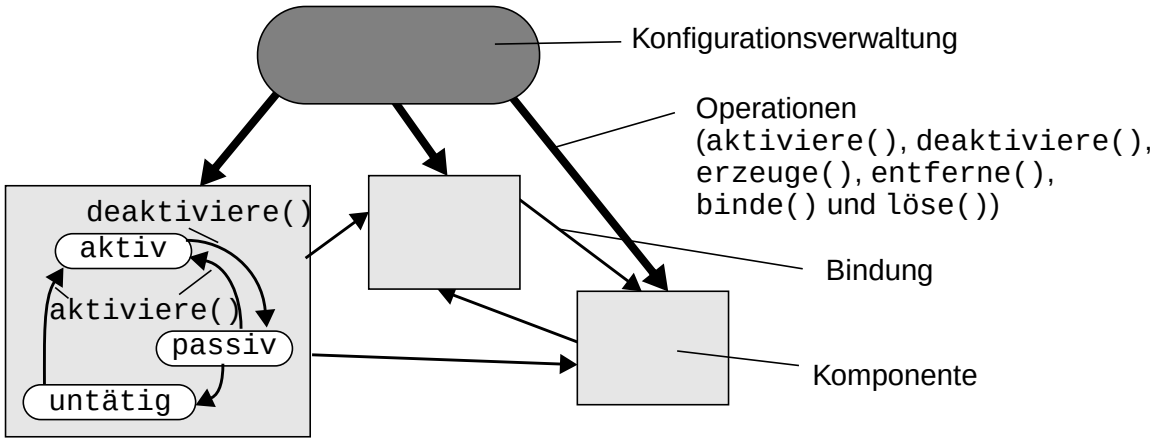


Bild 3.2: Zusammenwirken zwischen Konfigurationsverwaltung und Komponenten

komponente repräsentiert, weggerichtet ist. Eine Transaktion kann nur durch eine Komponente initiiert werden, von welcher auch das Einrichten der Bindung initiiert wurde.

Es werden die beiden Komponentenzustände `passiv` und `aktiv` eingeführt. Im aktiven Zustand kann eine Komponente Transaktionen initiieren, derartige Initiativen entgegennehmen und Transaktionen durchführen. Im passiven Zustand kann eine Komponente keine Transaktionen initiieren. Es existiert zudem keine von ihr initiierte Transaktion. Eine Komponente kann in einen Zustand `untätig` übergehen, wenn sie sich im passiven Zustand befindet und sie keine Transaktionswünsche annimmt und auch an keiner laufenden Transaktion beteiligt ist. Die Konfigurationsverwaltung führt Operationen in Komponenten aus, um Zustandsübergänge zu erzwingen. Mit `aktiviere()` wird der Übergang in den aktiven Zustand und mit `deaktiviere()` der Übergang vom aktiven in den passiven Zustand veranlasst.

Eine Komponente kann gemäß der Definition in dieser Arbeit (siehe Abschnitt 2.1.4.1) aktiv sein, wenn sie sich bei Kramer entweder im Zustand `aktiv`, `passiv` oder gar `untätig` befindet. Der Grund hierfür ist, das gemäß der Definition in dieser Arbeit eine Komponente allein durch eine Verwaltungsschnittstelle aktiviert werden kann. Dann befindet sie sich nach Kramer, der seine Begriffe auf die Inter-Komponentenkooperation bezieht, im untätigen Zustand. Sie kann erst danach in den kramerschen, aktiven Zustand übergehen, damit von ihr Transaktionen initiiert werden können. Eine Komponente ist gemäß dieser Arbeit aktiv und befindet sich im kramerschen Zustand `passiv`, wenn sie zwar keine Transaktionen initiieren, jedoch an nicht von ihr initiierten Transaktionen beteiligt ist.

In diesem Abschnitt werden nur Transaktionen, deren Abschluss nicht von der Beendigung weiterer Transaktionen abhängt, betrachtet. In [73] wird eine erweiterte Betrachtung vorgestellt.

Es kann für jede Komponente eine Menge definiert werden, die die Komponente selbst und alle Komponenten, die mindestens eine Transaktion zu ihr initiiert haben, enthält. Man kann

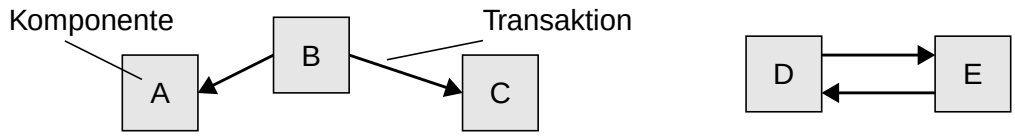


Bild 3.3: Exemplarische Konfigurationen

erkennen, dass sich diese Komponente im kramerschen, untätigen Zustand befindet, wenn alle Komponenten dieser Menge den kramerschen, aktiven Zustand verlassen haben. Die Menge beschreibt folglich die Komponenten, auf die eine Konfigurationsverwaltung einwirken muss, damit die betrachtete Komponente den untätigen Zustand annehmen kann.

Kramer stellt eine grundlegende Forderung, deren Erfüllung notwendig ist, um Konsistenz zu gewährleisten: Während die Konfigurationsverwaltung auf eine Komponente durch eine `erzeuge()`-, `entferne()`-, `binde()`- oder `löse()`-Operation einwirkt, darf diese an keiner Transaktion beteiligt sein, d. h. sie muss sich im Zustand `untätig` befinden.

Die Notwendigkeit dieser Forderung soll durch ein Beispiel für die in Bild 3.3 gegebene Konfiguration der Komponenten A, B und C veranschaulicht werden. Die Komponenten B und C führen die in Bild 3.4 dargestellte Transaktion durch. Während der Transaktion veranlasst die Konfigurationsverwaltung eine Systemänderung. Dies führt dann zu der dargestellten Inkonsistenz.

Die Operation `erzeuge()` kann immer ausgeführt werden, ohne dass Konsistenzverletzungen auftreten. Die Operation `entferne()` kann hingegen nur dann in einer Komponente ausgeführt werden, wenn keine andere Komponente zu ihr gebunden ist. Hierzu muss vorher die Operation `löse()` so oft angewendet werden, bis dieser Zustand erreicht wird. Die Operationen `binde()` und `löse()` dürfen nur dann ausgeführt werden, wenn die Komponente, von der die Bindung initiiert wird bzw. wurde, sich im Zustand `untätig` befindet.

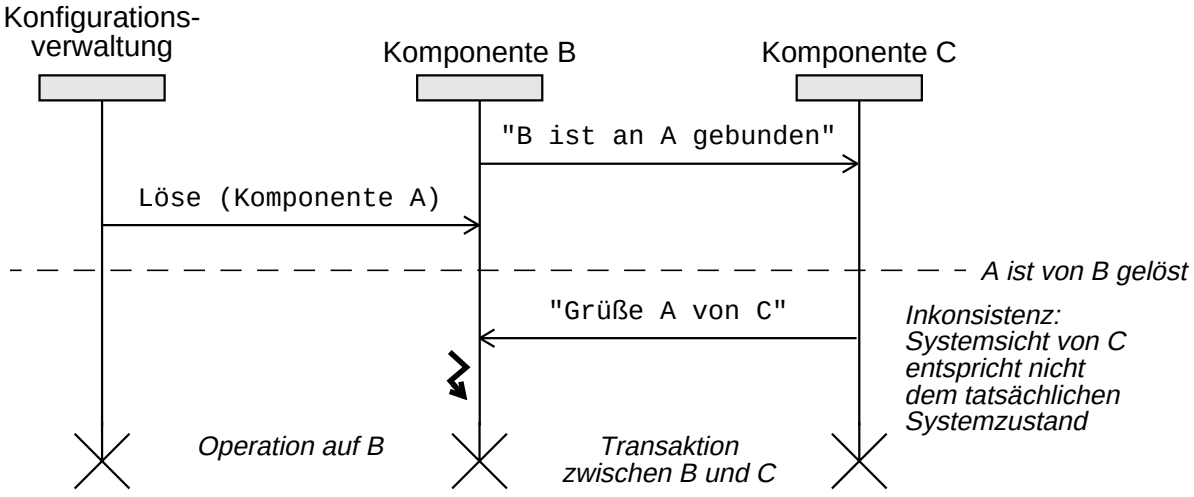


Bild 3.4: Beispiel für eine Konsistenzverletzung

- Konsistente Migration nach Chen

  In der Arbeit von Chen wird davon ausgegangen, dass die Komponenten durch Beauftragungen im Sinne von Methodenaufrufen kooperieren. Im Unterschied zu Kramer wird das Binden und Lösen von der Komponente selbst und nicht von einer Konfigurationsverwaltung initiiert, da die beiden Vorgänge inhärent zu einer von der Komponente initiierten Beauftragung gehören. Bei Chen kann eine Komponente simultan an mehreren Aufrufen beteiligt sein. Dies bedeutet, dass im Gegensatz zu Kramer in einer Komponente Binde- und Löse-Vorgänge stattfinden können, während die Komponente an Transaktionen beteiligt ist.

  Eine Komponente kann nach Chen nur migrieren, wenn keine Aufrufe, an der sie beteiligt ist, ausgeführt werden. Sie kann dann auf der Ausgangsablaufumgebung entfernt und auf der Zielablaufumgebung neu erzeugt werden. Hierbei wird die Einhaltung der kramerschen Konsistenzbedingung gefordert, dass während einer von der Konfigurationsverwaltung initiierten Operation (hier Migration), die betroffene Komponente an keiner Transaktion beteiligt sein darf. Dass eine Komponente an keinem ausgeführten Aufruf beteiligt ist, kann von der Konfigurationsverwaltung festgestellt werden, indem jeder Aufruf von ihr mitprotokolliert wird.

  Fällt die Entscheidung für eine Komponentenmigration, muss folglich gewartet werden, bis die Ausführungen aller Aufrufe, an der die Komponente beteiligt ist, beendet worden sind. Aufrufe, die zwischenzeitlich an die Komponente gerichtet bzw. von der Komponente initiiert werden, werden vor der Ausführung blockiert und erst nach der Migration wird die Blockierung aufgelöst. Hierbei müssen Verklemmungen vermieden werden. So darf nicht, um die in Bild 3.3 dargestellte Komponente D migrationsfähig zu machen, der Aufruf von E nach D blockiert werden, wenn er innerhalb des Aufrufs von D nach E initiiert wurde. Dies hätte dann auch eine Blockierung dieses Aufrufs zur Folge, so dass D nie in einen migrationsfähigen Zustand gelangen könnte und die Blockierungen nie aufgehoben werden würden. Die Konfigurationsverwaltung protokolliert deshalb zu jedem Aufruf einen Abhängigkeitspfad mit, um eine Verklemmung bei solchen Aufrufzyklen zu vermeiden.

  Eine konsistente Migration ist nur möglich, wenn die Änderung der Lokalität einer migrierten Komponente bei allen existierenden Referenzen auf diese Komponente berücksichtigt wird. Nach der Migration sind die Aufrufe direkt an die migrierte, d. h. neu erzeugte Komponente auf der Zielablaufumgebung gerichtet. Dies ist durch Einführung spezieller Stubs möglich, die wie gewöhnlich lokal einen Aufruf entgegennehmen, ihn im Sinne ihrer Middleware-Aufgabe weiterverarbeiten und zusätzlich noch für die Transparenzverwaltung verantwortlich sind.

3.1.3 Konfigurationsverwaltung am Beispiel der Fahrzeugtelematik

In diesem Abschnitt werden zunächst Dienste und Funktionen einer möglichen Konfigurationsverwaltung vorgestellt, wie sie zukünftig in Verteilten Systemen für die Fahrzeugtelematik eingesetzt werden könnte. Danach wird eine Java-basierte Realisierung einer Komponentenablaufumgebung, für die eine derartige Konfigurationsverwaltung verantwortlich ist, präsentiert.

3.1.3.1 Dienste und Funktionen einer Konfigurationsverwaltung

In Bild 3.5 wird eine Schichtung der Funktionen einer Konfigurationsverwaltung dargestellt. Die oberste Schicht beinhaltet die Funktionen zur Verwaltung der strukturellen Konfiguration. Eine strukturelle Rekonfigurierung geht nach Abschnitt 3.1.1.2 mit einer topologischen Rekonfigurierung einher. Deswegen werden Dienste der darunterliegenden Schicht, die für die topologische Konfiguration verantwortlich ist, genutzt. Diese Schicht beinhaltet auch Funktionen zur Lastverteilungsverwaltung. Dienste und Funktionen dieser Schicht nutzen wiederum Dienste einer darunterliegenden Schicht. Diese ermöglichen z. B. das Laden von authentifizierten Komponenten oder die Nutzung von Diensten unter Berücksichtigung von Sicherheitsrichtlinien. Verwaltungsdienste der untersten Schicht können dann durch den Einsatz von Heimvernetzungstechnologien bereitgestellt werden. Sie ermöglichen z. B. das Laden von Komponenten oder die Nutzung von Diensten mit Hilfe von Methodenfernaufrufen.

Im Folgenden werden die in diesem Abschnitt neu eingeführten Dienste und Funktionen näher erläutert.

<table>
<tr><td colspan="12">Verwaltung der strukturellen Konfiguration
(u. a. Dienstinstallation, Benutzerprofilverwaltung)</td></tr>
<tr><td colspan="12">Verwaltung der topologischen Konfiguration
(u. a. Lastverteilungsverwaltung, Reaktion auf Ressourcen-Ausfall)</td></tr>
<tr><td colspan="6">Dienste zur erweiterten
Komponentenverwaltung:
z. B. Komponentenauthentifikation,
Migrationsverwaltung</td><td colspan="6">Dienste zur erweiterten
Dienstverwaltung:
z. B. Zugriffskontrolle auf Dienste,
Migrationsverwaltung</td></tr>
<tr><td colspan="6">Basisdienste zur
Komponentenverwaltung
z. B. von OSGi</td><td colspan="6">Basisdienste zur
Dienstverwaltung
z. B. von HAVi, Jini, UPnP, DANA, ...</td></tr>
<tr><td>Laden</td><td>Aktivieren</td><td>Speichern</td><td>Ersetzen</td><td>Migrieren</td><td>Entfernen</td><td>Bekannt-
machen</td><td>Binden</td><td>Nutzen/
Erbringen</td><td>Lösen</td><td>Ressourcen
reservieren</td><td>...</td></tr>
</table>

Bild 3.5: Schichtung der Konfigurationsverwaltung

- Verwaltung der strukturellen Konfiguration
  - Dienstinstallation

    Es werden Funktionen bereitgestellt, die feststellen, dass ein Dienst erbracht werden muss und die daraufhin eine für die Diensterbringung verantwortliche Komponente von einer externen Infrastruktur ins Fahrzeug laden. Die Komponenten können auch von fahrzeuginternen Zwischenspeichern (engl. *Caches*) geladen werden.

    Der Konfigurationsverwaltung ist durch das Wissen über die statischen Abhängigkeiten (siehe Abschnitt 3.1.2.2) der geladenen Komponenten zu jedem Zeitpunkt bekannt, welche Dienste grundsätzlich im Verteilten System benötigt werden.
  - Benutzerprofilverwaltung

    Sie beinhaltet Funktionen, die feststellen, aus welchen Komponenten sich die Anwendungen eines Benutzers zusammensetzen. Es wird festgestellt, welche strukturelle Teilkonfiguration des Verteilten Systems dem Profil eines Benutzers entspricht. Nach diesem Profil können dann Komponenten geladen und angeordnet werden.
- Dienste zur Komponenten- und Diensteverwaltung mit erweiterter Funktionalität
  - Komponentenauthentifikation

    Beim Laden neuer Komponenten von einer externen Infrastruktur in das Telematiksystem des Fahrzeugs müssen Sicherheitsrichtlinien beachtet werden. Im Umfeld dieser Arbeit ist in [79] ein Vorschlag für eine Sicherheitsarchitektur entstanden, die eine Authentisierung von extern geladenen Komponenten gegenüber einer Verwaltungsinstanz ermöglicht. Bei erfolgreicher Authentisierung wird sichergestellt, dass nur eine der Verwaltungsinstanz bekannte Instanz für die Codierung einer geladenen Komponente verantwortlich ist.
  - Zugriffskontrolle auf Dienste

    Im Umfeld dieser Arbeit ist in [123] eine Erweiterung von Java RMI entstanden, durch die bei jedem Methodenfernaufruf geprüft werden kann, ob die aufrufende Instanz berechtigt ist, das Server-Objekt zu beauftragen. Die RMI-Stubs werden hierzu um eine Rechteverwaltung ergänzt.
  - Überführen einer Komponente in einen migrationsfähigen Zustand

    Es werden Dienste bereitgestellt, die es ermöglichen, eine Komponente in einen migrationsfähigen Zustand überzuführen. Hierzu müssen die in Abschnitt 3.1.2.2 vorgestellten Verfahren (z. B. von Kramer oder Chen) angewendet werden.

- Basisdienste zur Dienstverwaltung

  Reservierung von Ressourcen

  HAVi ermöglicht die Reservierung von Übertragungskapazitäten für die Nutzung synchroner Übertragungsdienste. Im Umfeld dieser Arbeit ist die Architektur DANA (*Distributed Systems Architecture for Networks in Automotive Environments*, [28, 29, 30]) entwickelt worden, die neben der Reservierung von Ressourcen des Bussystems auch die Reservierung von Endgerät-Ressourcen (CPU-Zeit, Übertragungskapazitäten des Peripherie- oder internen Busses) ermöglicht.

3.1.3.2 Exemplarische Betrachtung von Konfigurationsverwaltungsarchitekturen

Eine mögliche Ausprägung einer Konfigurationsverwaltung soll im Folgenden exemplarisch vorgestellt werden. Diese Konfigurationsverwaltung orientiert sich an den Architekturen COSIMA (*Component System Information and Management Architecture*, [128]) und DANA. Diese Architekturen beinhalten miteinander kooperierende Verwaltungsinstanzen, wobei die in Abschnitt 3.1.3.1 vorgestellte Schichtenhierarchie realisiert wird. Bei beiden besteht eine Komponente aus ihrem eigentlichen Nutzanteil und einem Verwaltungsanteil. Damit wird die eigentliche Kooperation der Komponente mit der Konfigurationsverwaltung vor dem Nutzanteil verborgen.

Es wird nicht festgelegt, mit welchen Middleware- und Komponenten-Software-Technologien die Konfigurationsverwaltung zu realisieren ist. Standardmäßige Middleware- und Komponenten-Software-Technologien können nahezu beliebig in die Architektur integriert werden (siehe Bild 3.6**)**.

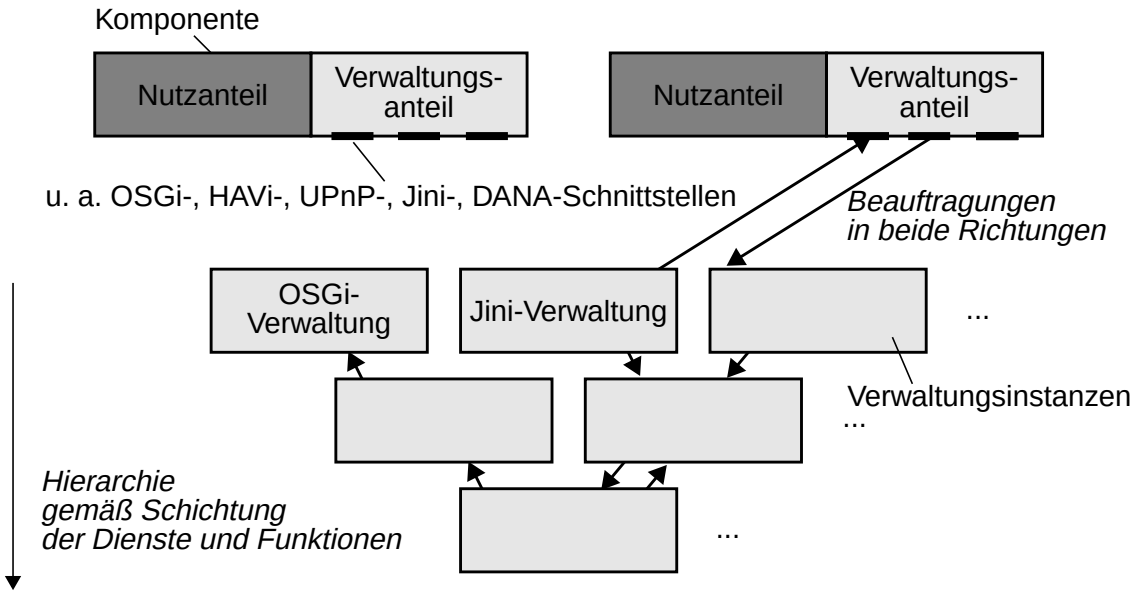


Bild 3.6: Kooperation zwischen Komponenten und Verwaltungsinstanzen

3.1.3.3 Java-Komponentenablaufumgebungen

Der von einem Java-Compiler erzeugte Code kann nur auf einer Java-spezifischen Ablaufumgebung ausgeführt werden. Der Maschinenbefehlssatz dieser Ablaufumgebung wird von vielen Endgeräten durch deren Maschinenbefehlssatz emuliert. Die java-spezifische Ablaufumgebung wird deshalb als Virtuelle Java-Maschine (JVM, *Java Virtual Machine*) bezeichnet. Es sind drei Ausführungsformen bekannt:

- Der JVM-Code kann mittels einer sogenannten Interpreter-Anwendung während der Ausführung in den realen Maschinen-Code übersetzt werden. Sogenannte JIT-Compiler (*Just In Time*) ergänzen dieses Prinzip, wobei durch sie der JVM-Code auch bei Mehrfachausführung nur einmal zur Programmlaufzeit übersetzt werden muss.
- Der JVM-Code kann mittels einer sogenannte Compiler-Anwendung vor der Ausführung in den realen Maschinen-Code übersetzt werden. In einem Prozess, der von einem Betriebssystem verwaltet wird, kann der übersetzte Code durchlaufen und ausgeführt werden. Eine Alternative ist, den Code direkt ohne Betriebssystem auszuführen. Dies ist prinzipiell möglich, da den Java-Programmen durch Java-eigene Bibliotheken Betriebssystemdienste und Betriebssystemfunktionen bereitgestellt werden. Diese müssen dann ohne Zuhilfenahme eines schon existierenden Betriebssystems realisiert werden.
- Schließlich besteht die Möglichkeit, den JVM-Maschinenbefehlssatz direkt auf einem Prozessor zu realisieren.

Im Folgenden werden einige Dienste und Funktionen eines Betriebssystems, die durch eine JVM bereitgestellt werden, vorgestellt.

Verwaltung von Prozessen

Viele Betriebssysteme können die Bearbeitung von voneinander unabhängigen (d. h. nebenläufigen) Vorgängen so abwechseln, dass sie nach außen als simultan bearbeitet erscheinen, auch wenn nur eine Bearbeitungseinheit zur Verfügung steht. Beim Bearbeitungswechsel wird immer der Zustand des zu unterbrechenden Vorgangs gesichert. Prozessorientierte Betriebssysteme verwalten Prozesse. Neben gewöhnlichen Prozessen, denen ein eigener Speicherbereich zugeteilt ist, gibt es noch sogenannte leichtgewichtige Prozesse (engl. *Threads*), die sich einen Speicherbereich teilen. Ein gewöhnlicher Prozess kann mehrere leichtgewichtige Prozesse beinhalten.

Eine JVM kann durch den Ablauf eines Prozesses realisiert werden, der z. B. von einem prozessorientierten Betriebssystem als Teil einer Prozessmenge verwaltet werden kann. Teil dieses JVM-Prozesses sind gegebenenfalls leichtgewichtige Prozesse. Die Verwaltung leichtgewichtiger Prozesse kann innerhalb des JVM-Prozesses vorgenommen werden (engl. *Green Threads*), oder gegebenenfalls dem prozessorientierten Betriebssystem, das den JVM-Prozess

verwaltet, überlassen werden (engl. *Native Threads*). Die Prozessverwaltung wird oftmals auch als lokales Prozess-Scheduling bezeichnet.

Sowohl ein gewöhnlicher als auch ein leichtgewichtiger Prozess besitzt einen Verwaltungszustand. In Bild 3.7 sind alle möglichen Zustände und Übergänge bei einer sehr einfachen Prozessverwaltung dargestellt. Das Zustandsübergangsdiagramm basiert auf einem in [50] vorgeschlagenen Ansatz. Ein Prozess wird existent (1), wenn er bei einer Prozessverwaltung angemeldet wird. Durch einen Startvorgang geht ein existenter Prozess in den bereiten Zustand über (2) und wartet auf die Abarbeitung seines Programms. Wenn er in den laufenden Zustand übergeht (3), wird sein Programm solange abgearbeitet und sein Befehlsfolgezähler solange modifiziert, bis das Ende der Abarbeitung erreicht worden ist und er beendet wird (4). Danach wird er entweder von neuem gestartet (5) oder abgemeldet (6). Dann ist er wieder der Prozessverwaltung unbekannt. Ein weiterer Grund, einen laufenden Zustand zu verlassen ist, dass die weitere Abarbeitung den Zugriff auf eine gegebenenfalls dem Prozess noch nicht zugeteilte Ressource erfordert. Der Prozess wird deshalb solange blockiert (7), bis das Ereignis, auf das er wartet, eingetreten ist. Anschließend geht der Prozess wieder in den Zustand `bereit` über (8). Schließlich kann der Prozess auch von der Prozessverwaltung aus dem laufenden in den bereiten Zustand versetzt werden (9). Die Prozessverwaltung initiiert diesen Zustandsübergang, um das Ablaufen anderer Prozesse zu ermöglichen.

Gemäß der Java-Spezifikation [43] kann jedem leichtgewichtigen Prozess eine Priorität zugeordnet werden. Befinden sich bei einem Ein-Prozessorsystem zum selben Zeitpunkt mehrere leichtgewichtige Prozesse im Zustand `bereit`, wird i. A. der Prozess mit der höchsten Priorität in den Zustand `laufend` übergehen. Die Spezifikation macht jedoch keine genaueren Vorschriften, da sonst die Vielfalt der Möglichkeiten, eine JVM zu realisieren, eingeschränkt werden würde. Zumeist wird die Spezifikation wie folgt umgesetzt:

- Wenn kein Prozess abläuft und wenn mehrere Prozesse bereit sind, dann wird ein Prozess mit der höchsten Priorität in den Zustand `laufend` übergeleitet.

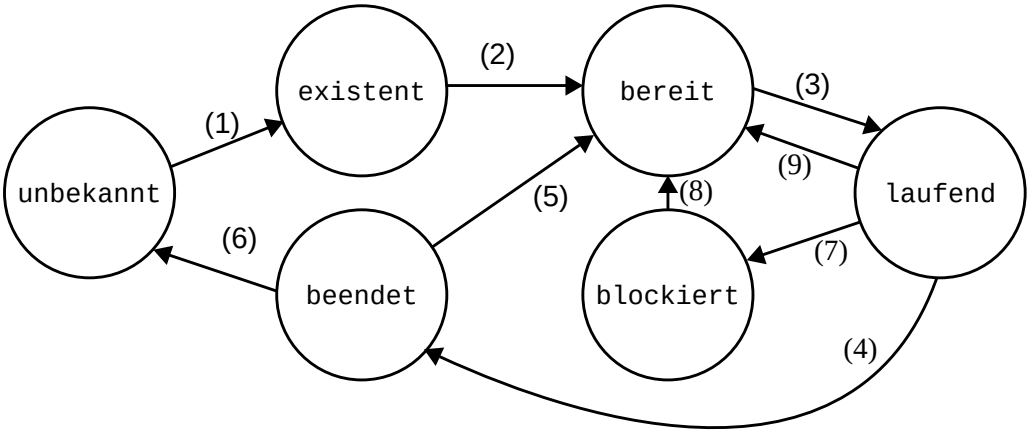


Bild 3.7: Zustände von Prozessen

- Wenn ein Prozess abläuft und wenn ein Prozess mit höherer Priorität bereit wird, so wird der laufende Prozess unterbrochen, damit der höherpriore Prozess ablaufen kann.
- Für Prozesse mit gleichen Prioritäten wird oft ein Zeitscheibenverfahren angewendet. Dieses Verfahren ist innerhalb der JVMs von Sun für Java 1.0 auf den Plattformen Windows 95 und Windows NT realisiert. Die JVM von Blackdawn für Java 1.0 auf Linux in der Version 1.1.8 arbeitet jedoch ohne Zeitscheiben, d. h. die leichtgewichtigen Prozesse mit gleichen Prioritäten unterbrechen sich nicht gegenseitig sondern werden gemäß dem FIFO-Prinzip (*First In First Out*) abgearbeitet [91].

Daneben existieren Bestrebungen, Java-Prozessverwaltungen bereitzustellen, die Prozesse nach Fertigstellungsterminen einer Bearbeitung zuzuteilen, so dass immer derjenige Prozess mit dem zeitnahsten Fertigstellungstermin gerade abläuft (engl. *Earliest Deadline First*, EDF) [104]. Hierzu muss dem Software-Entwickler eine zusätzliche Java-Bibliothek zur Verfügung gestellt werden [133].

Verwaltung der Synchronisation von Prozessen

Nebenläufige Prozesse müssen synchronisiert werden, wenn sie miteinander kooperieren oder gegeneinander um eine Ressource konkurrieren. Synchronisation schränkt die Unabhängigkeit der Abfolge der Abarbeitung verschiedener Prozesse ein [50]. Sie kann durch gegenseitige Beobachtung oder durch einen hierfür beauftragten Überwacher erfolgen [25]. Bei Java wird mit dem sogenannten Monitor-Konzept [52] standardmäßig eine Synchronisation durch einen Überwacher realisiert.

Bei Java können Daten in Objekten gekapselt werden, so dass man nur über Methoden auf sie zugreifen kann. Ein dem Objekt zugeordneter Monitor (Überwacher) ist für die Synchronisation beim Zugriff auf die Daten verantwortlich. Wenn ein Prozess bei der Abarbeitung einer Methode in einen sogenannten kritischen Bereich eintritt, dann sperrt der Monitor den Eintritt in alle anderen kritischen Bereiche des Objekts. Erst bei Verlassen dieses kritischen Bereichs wird diese Monitor-Sperre zurückgesetzt. Dies bedeutet, dass Prozesse, die einen kritischen Bereich durchlaufen wollen, blockiert sind, solange ein anderer Prozess einen kritischen Bereich durchläuft. Wenn alle öffentlich zugänglichen Methoden eines Objektes als kritischer Bereich deklariert sind, kann dieses Objekt zu jedem Zeitpunkt nur eine einzige, externe Beauftragung bearbeiten.

Bei Java wird standardmäßig eine Schnittstelle angeboten, die es erlaubt, dass ein ablaufender leichtgewichtiger Prozess blockiert, bzw. vom blockierten Zustand in den Zustand `bereit` versetzt werden kann. Falls ein Prozess, der in einem kritischen Bereich abläuft, über diese Schnittstelle blockiert wird, wird die Monitor-Sperre zurückgesetzt. Sie wird wieder gesetzt, wenn der Prozess wieder abläuft.

Speicherverwaltung

Eine JVM kapselt die Speicherverwaltung, so dass in einem Programm nicht direkt auf einen physikalischen Speicherbereich, sondern nur über logische Referenzen auf Objekte zugegriffen werden kann. Objekte können dynamisch, d. h. zur Laufzeit, erzeugt werden, wobei die JVM dann den dafür benötigten physikalischen Speicherbereich belegt. Ein Attribut eines Objekts kann im Wesentlichen entweder den Wert eines primitiven Datentyps annehmen oder einer Referenz auf ein weiteres Objekt entsprechen. So kann der Datenzustand in einem ablaufenden Programm durch einen zusammenhängenden Graphen mit Objekten als Knoten und Referenzen als gerichtete Kanten dargestellt werden. Die Speicherverwaltung der JVM kann feststellen, ob in diesem Graphen keine Kante mehr auf einen Knoten gerichtet ist und man deshalb nicht mehr auf ein Objekt über eine Referenz zugreifen kann. In einem solchen Fall wird der vom Objekt belegte Speicherbereich von der Speicherverwaltung wieder freigegeben (engl. *Garbage Collection*, GC).

Schnittstellen zu weiteren Betriebssystemdiensten

Es werden standardmäßig noch weitere Betriebssystemdienste angeboten, z. B. zum Abruf der Systemzeit oder zur Verwaltung von Dateien und des Zugangs zu IP-, UDP- und TCP-Diensten. Da Java in dem Sinne plattformunabhängig ist, dass die Programme dieser Sprache nur in einer abstrakten JVM ausgeführt werden können, stehen standardmäßig keine plattformspezifische Zugriffe auf die Peripherie eines Systems zur Verfügung. Gegebenenfalls müssen diese zusätzlich implementiert werden. Bei Java wird diese Implementierung durch sogenannte JNI-Mechanismen (*Java Native Interface*) unterstützt.

Schwache Migration bei Java

Bei Java wird die Realisierung Schwacher Migration, d. h. das Verschieben des Datenzustandes und des Codes einer Komponente, durch Java-eigene Mechanismen und standardmäßig vorhandene Bibliotheken, die u. a. auch den RMI-Mechanismus realisieren, unterstützt.

Ein Datenzustand wird durch Objekte gehalten, die mit ihren Referenzen einen zusammenhängenden Graphen aufspannen. Es existiert eine Standardbibliothek, die einen Objektgraphen in eine serialisierte Byte-Folge bzw. diese wieder in den Objektgraphen umwandelt. Auch bei der Realisierung des Methodenfernaufrufs RMI wird dieser Serialisierungsmechanismus verwendet. Ein Objektgraph wird dann als Übergabe- oder Rückgabeparameter per Kopie in serialisierter Form zwischen Client und Server ausgetauscht. Fehlen bei den Zielablaufumgebungen die zu den Objekten gehörenden Klassenbeschreibungen, die den Code der Komponente darstellen, können diese beispielsweise per HTTP auf die Zielablaufumgebung nachgeladen werden.

Wie die folgende Überlegung zeigt, lässt sich eine Schwache Migration einer nach dem Client/Server-Prinzip arbeitenden Server-Komponente ohne größeren Aufwand zu einer Starken Migration erweitern: Man kann eine Server-Komponente mit Hilfe der Middleware-Technologie RMI auf einfache Weise realisieren. Ein RMI-Server hält einen leichtgewichtigen Empfangsprozess bereit, der nur beim Empfang einer Server-Anfrage läuft und ansonsten wartet (d. h. blockiert ist). Nach dem Empfang einer Beauftragung erzeugt er einen Bearbeitungsprozess und lässt ihn ablaufen. Folglich lässt sich der Aktivitätszustand eines untätigen RMI-Servers, der keine Anfrage bearbeitet, durch einen wartenden Empfangsprozess beschreiben. Es kann auf triviale Weise der Aktivitätszustand der migrierten, untätigen Server-Komponente auf der Zielablaufumgebung auch bei Schwacher Migration wiederhergestellt werden: Der Empfangsprozess der ansonsten inaktiven RMI-Server muss bei der Prozessverwaltung der Zielablaufumgebung angemeldet und in den wartenden Zustand überführt werden. Die Konsistenzbetrachtungen in Abschnitt 3.1.2.2 zeigen die Relevanz dieser Überlegung, da Migration von aktiven Komponenten Inkonsistenzen mit sich bringen können und somit nur Migrationen von nicht an Transaktionen beteiligten Komponenten sinnvoll sind.

3.2 Konfiguration bei Kompositionen von benutzergesteuerten Anwendungen

In diesem Abschnitt werden Kompositionen von benutzergesteuerten Anwendungen untersucht. Unter dem Begriff Anwendungskomposition wird in dieser Arbeit die Gesamtheit aller Komponenten, die sich zu einer Anwendung zusammensetzen, verstanden. Die Software in den im Folgenden betrachteten Verteilten Systemen besteht gemäß dem im Abschnitt 2.1.4.3 vorgestellten mehrstufigen Komponenten-Software-Modell aus Anwendungs- und Systemdienstimplementierungen. Eine (bindungs-)strukturelle Konfiguration eines Verteilten Systems ergibt sich aus allen Anwendungskompositionen, allen Systemdienste realisierenden Komponenten und allen Bindungen.

Zunächst wird in diesem Abschnitt der Aufwand, der bei einer Kooperation zwischen Komponenten entsteht, abgeschätzt. Insbesondere wird der Mehraufwand betrachtet, der die Verwendung einer in Komponenten zerlegten Anwendung anstatt einer monolithischen Anwendung mit sich bringt. Es wird gezeigt, dass eine Zerlegung in Komponenten trotzdem lohnenswert sein kann. Nach diesen Untersuchungen wird eine oftmals vorhandene Eigenschaft von benutzergesteuerten Anwendungen bzw. Anwendungskompositionen vorgestellt: Das Programm wird in vielen Fällen nur durch einen einzigen Kontrollfluss, der durch Benutzeraktionen gesteuert wird, durchlaufen. Zu jedem Zeitpunkt ist je Anwendungskomposition nur eine einzige durch eine Benutzeraktion ausgelöste Abfolge von verschachtelten Komponentenbeauftragungen existent. Diese Betrachtungen bilden die Grundlage einer im nächsten Kapitel vor-

gestellten Methode zur Modellierung von Konfigurationen Verteilter Systeme, die solche Anwendungskompositionen beinhalten.

3.2.1 Untersuchung des durch Middleware entstehenden Mehraufwands

Der durch den Einsatz von Middleware zustandekommende Mehraufwand bei einer Kooperation zwischen Komponenten wird in diesem Abschnitt durch einen Vergleich zwischen einem Methodenfernaufruf und einem gewöhnlichen Methodenaufruf bestimmt. Er drückt sich in einer Ausdehnung der Blockierungsdauer bei Aufrufen aus. Diese kommt durch die Serialisierung und Deserialisierung der auszutauschenden Daten und durch deren Übertragung in Form einer seriellen Byte-Folge zustande. Da Middleware das Verteilen von Komponenten unterstützt, kann sich jedoch ihr Einsatz trotz dieses Mehraufwands lohnen. Dies wird in diesem Abschnitt an einer exemplarischen Anwendungskomposition veranschaulicht, bei der durch ein optimales Verteilen der Komponenten die Antwortzeiten bei Benutzeraktionen minimiert werden.

3.2.1.1 Mehraufwand bei verschiedenen Middleware-Produkten

In der CORBA-Spezifikation [92] sind sogenannte Auffangorte (engl. *Interceptors*) zwischen einem Client und einem Server definiert (siehe Bild 3.8), die die auszutauschenden Daten passieren müssen. Für einen Software-Entwickler sind diese Auffangorte als Schnittstellen sichtbar, die einen Zugang zu den diesen Ort passierenden Daten ermöglichen. So können z. B. dort Filter oder Verschlüsselungsinstanzen auf dem Weg zwischen einem Client und einem Server eingebaut werden. Bei einer im Umfeld dieser Arbeit durchgeführten Untersuchung [38] wurden an Auffangorten Messuhren eingebaut, die den Zeitpunkt festhalten, wann bestimmte Daten diese Orte passieren.

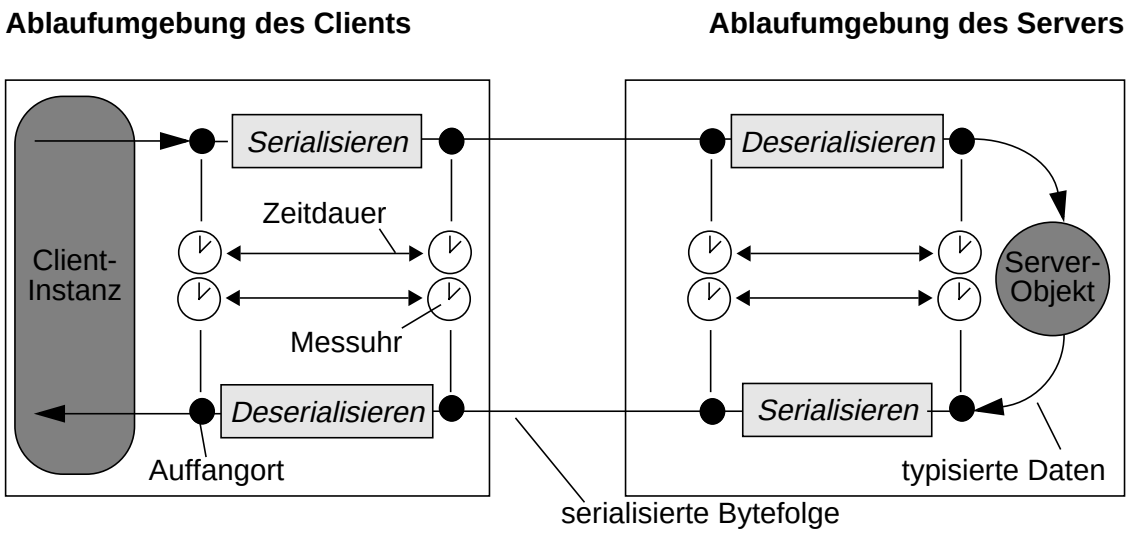


Bild 3.8: CORBA-Interceptoren als Orte von Messuhren

Die Messungen wurden für das CORBA-Produkt Orbix 2.3c für C++ und Visibroker für Java auf einem Pentium MMX 200 MHz-Rechner mit einem Windows NT 4.0-Betriebssystem durchgeführt. Die Zeit wurde durch die in [11] vorgestellte Software erfasst. Bei den Messungen wurde der Typ und die Menge der auszutauschenden Daten variiert.

Die CORBA-IDL-Typen der ausgetauschten Daten sind Octet (1 Byte), Char (1 Byte), Short (2 Byte), Long (4 Byte) und Double (8 Byte). Zusätzlich wurde eine 16 Byte große Datenstruktur vom Typ Struct untersucht, die sich aus den Feldern vom Typ Octet, Char, Short, Long und Double zusammensetzt.

Zudem wurden Messungen durchgeführt, die eine Bewertung der Leistungsfähigkeit der Java-Standardbibliothek zur Ein- und Ausgabeverwaltung bezüglich der dort bereitgestellten Funktionalität zum Serialisieren von Objekten in Java-spezifische Byte-Folgen und deren Deserialisieren für das Produkt JDK 1.1.7 für Linux der Firma Sun erlauben.

3.2.1.2 Ergebnisse von Messungen

Die zusätzliche Dauer der Blockierung bei einem Methodenfernaufruf gegenüber einem Methodenaufruf, bei dem Referenzen auf die Parameter übergeben werden, setzt sich aus den Summanden Serialisierungsdauer, Übertragungsdauer und Deserialisierungsdauer für übergebene bzw. zurückgegebene Parameter zusammen.

Aus den Messungen lassen sich folgende Ergebnisse ableiten:

- Die Serialisierungsdauer bzw. Deserialisierungsdauer hängt stark vom gewählten Middleware-Produkt ab. Unabhängig vom Produkt gilt, dass die Serialisierungsdauer bzw. Deserialisierungsdauer gegenüber der Übertragungsdauer nicht vernachlässigt werden kann.
- Bei allen Produkten wurde ein näherungsweise linearer Zusammenhang zwischen dem Umfang der zu verarbeitenden Daten und der Zeitdauer des Serialisierens bzw. Deserialisierens festgestellt.
- Die Serialisierungsdauer bzw. Deserialisierungsdauer hängt stark vom Datentyp ab. Beim Produkt Orbix ergab sich beispielsweise für die Serialisierung von Double-Werten eine Bitrate von 18,2 Mbit/s und für die Serialisierung von Octet-Werten eine Bitrate von nur 2,7 Mbit/s.
- Aufgrund der vorgenommenen Messungen ist die Zeitdauer zwischen dem Serialisieren und dem Deserialisieren, d. h. die Zeitdauer des Übertragens der seriellen Byte-Folge zwischen Client und Server, berechenbar. Diese Zeitdauer ist im Allgemeinen nur durch die Übertragungskapazität des verwendeten Bussystems begrenzt. Bei Daten eines CORBA-Struct-Typs ist diese Zeitdauer jedoch noch zusätzlich abhängig von der Reihenfolge der Felder,

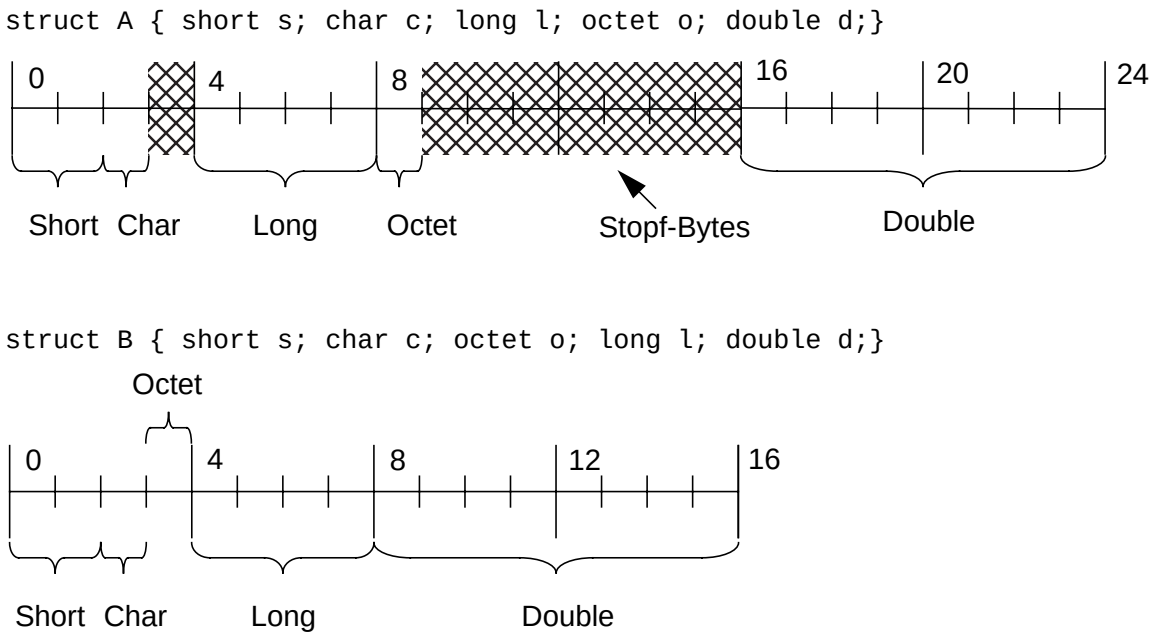


Bild 3.9: Serialisieren eines Struct-Datums in eine Byte-Folge gemäß der CDR

die vom Software-Entwickler bei der Struct-Deklaration gewählt wird. Dies liegt an der Spezifikation der CDR von CORBA. Gemäß dieser Transfer-Syntax darf ein Struct-Feld in der sequentiellen Byte-Folge nur an einer Stelle angeordnet werden, deren Position einem ganzzahligen Vielfachen der Länge des Datentyps des Feldes entspricht. Somit kann es erforderlich sein, dass in eine Byte-Folge zusätzliche Stopf-Bytes eingefügt werden müssen. Bild 3.9 veranschaulicht diesen Zusammenhang: Durch Verändern der Reihenfolge, indem man bei der Deklaration das Feld vom Typ Octet von der 4. Stelle auf die 3. Stelle vorzieht, werden keine Stopf-Bytes benötigt. Die Länge der seriellen Byte-Folge reduziert sich von 24 Byte auf 16 Byte. Dadurch wird die Übertragung auf dem Bussystem um ein Drittel schneller.

3.2.2 Modellierung von Anwendungskompositionen

In diesem Abschnitt wird anhand einer exemplarisch ausgewählten Anwendung eine Zerlegung in Komponenten veranschaulicht und danach die Abfolge von verschachtelten Komponentenbeauftragungen beschrieben.

Bei einer Orientierung an der in Abschnitt 2.1.4.3 eingeführten mehrstufigen Architektur erfolgt in diesem Beispiel eine Zerlegung in drei Komponenten, die jeweils die graphische Benutzerschnittstelle (Komponente A), die Ablaufsteuerung (Komponente B) und die Datenhaltung (Komponente C) eventuell unter Nutzung von Systemdiensten repräsentieren. Die Komponente A ist durch die Bereitstellung der graphischen Benutzerschnittstelle (engl. *Graphical User Interface*, GUI) an den Ort gebunden, an dem der Benutzer mit der Anwendung wechselwirkt. Eine weitere Dekomposition dieser Komponente ist beispielsweise durch eine

Orientierung an dem MVC-Architekturmuster (*Model View Controller)* [74] möglich. Hier wäre dann die Benutzereingabe durch die *Controller*-Komponente und die Ergebnisausgabe durch den *View*-Komponente an den Ort der Benutzerinteraktion gebunden. Die *Model*-Komponente, welche die Verarbeitung repräsentiert, müsste dann mit den Komponenten B und C kooperieren bzw. auf diese beiden Komponenten abgebildet werden.

3.2.2.1 Zerlegung einer benutzergesteuerten Anwendung in Komponenten

Es wird exemplarisch die Zerlegung der im Informatikpraktikum für den Studiengang Elektrotechnik und Informationstechnik in Java zu programmierenden Anwendung zur Verwaltung von Daten über Bahnhöfe [77] in Komponenten veranschaulicht. Sie stellt ein grafisches Menü bereit, mit dem der Anwender verschiedene Aktionen auslösen kann. Er kann u. a. Daten über Bahnhöfe von einer Datei einlesen, auf dem Bildschirm ausgeben, nach der Entfernung zu einem Referenzbahnhof sortieren und nach dem zusätzlichen Einlesen von Verbindungsdaten die kürzeste Strecke zwischen zwei Bahnhöfen berechnen.

Es wird eine Dekomposition der Anwendung in eine Komponente (A), welche die graphische Benutzerschnittstelle repräsentiert, und in eine Komponente (B+C), die sowohl für die Ereignisverarbeitung als auch für die Datenhaltung zuständig ist, vorgenommen. Bei der ursprünglichen, monolithischen Anwendung ist eine Strukturierung in Software-Pakete gemäß einer mehrstufigen Architektur bereits vorhanden. Die Zerlegung erfolgt ausschließlich durch die Modifikation des Mechanismus, der die Kooperation zwischen Objekten aus verschiedenen Software-Paketen realisiert. Bei der monolithischen Anwendung kooperieren diese Objekte durch gewöhnliche Methodenaufrufe. Nach der Zerlegung kooperieren die Objekte durch Methodenfernaufrufe. Die Komponentenablaufumgebungen sind rudimentär durch die JVM Kaffe [64, 143] gegeben. Die Methodenfernaufrufe werden durch Ninja RMI [142] realisiert.

In Tabelle 3.1 sind Ergebnisse von Messungen bezüglich der Rechenzeit für die Komponentenrealisierung vorgestellt. Hierbei wird das Einlesen der Bahnhofsdaten von einer Datei, das Sortieren nach der Entfernung zu einem Referenzbahnhof und das Ausgeben der sortierten Bahn-

Bahnhöfe	**1**	**100**	**500**	**1000**	**2000**
A-Komponente	1,04	1,03	0,97	0,90	0,77
B+C-Komponente	0,14	0,15	0,18	0,24	0,38
RMI-Registratur	0,09	0,09	0,08	0,08	0,06
Gesamt	1,27	1,27	1,23	1,22	1,21

Tabelle 3.1: Messung der Rechenzeit für Einlesen, Sortieren und Ausgeben von Daten

hofsdaten auf einem Bildschirm betrachtet. Bei den Messungen variiert die Anzahl der Bahnhofsdaten. Die angegebenen Werte entsprechen dem Verhältnis der Rechenzeiten bei der Komponentenrealisierung zu den Rechenzeiten bei der monolithischen Anwendung. Man kann aus der Tabelle diese relativen Werte für die A-Komponente und die B+C-Komponente entnehmen. Zusätzlich sind Werte für eine RMI-Registratur angegeben.

Durch den zusätzlichen Middleware-Aufwand, der durch die Registrierung der Komponenten und das Serialisieren, Übertragen und Deserialisieren der zwischen den Komponenten ausgetauschten Daten entsteht, benötigt die Komponentenrealisierung in jedem Fall mehr Rechenzeit als die monolithische Lösung. Der Rechenaufwand im Benutzerendgerät kann allerdings reduziert werden, da bei der Komponentenrealisierung nur die A-Komponente, jedoch nicht die B+C-Komponente und die Registratur an den Benutzerort gebunden sind. Diese Aufwandsreduzierung ist umso größer, je mehr Daten von der Anwendung verarbeitet werden. Dies liegt in diesem Beispiel daran, dass die Rechenzeit für A sowohl von der Menge der innerhalb von A auf dem Bildschirm dargestellten Daten als auch von der Menge der durch Middleware zwischen B+C und A ausgetauschten Daten nur linear abhängt, während die Verarbeitung der Daten in B+C durch den verwendeten Sortieralgorithmus eine quadratische Abhängigkeit zwischen Rechenzeit und Datenmenge mit sich bringt. Bei sehr vielen Daten benötigt somit die ortsungebundene B+C-Komponente gegenüber der ortsgebundenen A-Komponente die meiste Rechenzeit.

3.2.2.2 Beauftragungsabfolgen bei Kompositionen von benutzergesteuerten Anwendungen

Bei benutzergesteuerten Anwendungen löst ein Benutzer die Erbringung eines Dienstes aktiv aus. So kann ein Benutzer z. B. durch einen Mausklick auf eine grafische Oberfläche mit einer benutzergesteuerten Anwendung interagieren.

Eine Komponente gibt in der Rolle des Clients während einer Dienstnutzung die Kontrolle an die diensterbringende Server-Komponente ab. Sie blockiert solange, bis ihr von der diensterbringenden Komponente die Kontrolle wieder zurückgegeben wird. Bei einer mehrstufigen Diensthierarchie kommt es zu Verschachtelungen, wenn eine Komponente während der Diensterbringung die Rolle eines Dienstnutzers annimmt.

Synchronisationsbedingungen

Es ist oftmals notwendig, den Benutzer während der Diensterbringung zu blockieren, d. h. ihm die Kontrolle für die Anwendung zu entziehen, um Inkonsistenzen zu vermeiden.

Eine Benutzerblockierung ist prinzipiell immer bei in Java programmierten GUI-Anwendungen durch den inhärenten Einsatz einer sogenannten AWT-Ereigniswarteschlange (*Abstract Windowing Toolkit*) gegeben. AWT bezeichnet die grundlegende Standardbibliothek für Gra-

fikprogrammierung unter Java. Mit Hilfe dieser Warteschlange wird jedes Ereignis nach dem FIFO-Prinzip durch einen AWT-Prozess abgearbeitet. Auf in die Warteschlange eingereihte Ereignisse (z. B. Benutzeraktionen) wird solange nicht reagiert, bis alle vorher eingereihten Ereignisse verarbeitet worden sind. Diese FIFO-Abarbeitungsdisziplin hat jedoch negative Auswirkungen auf das Anwendungsverhalten, wenn die Ereignisabarbeitung eine für den Benutzer wahrnehmbare Zeitspanne dauert. So werden Bildschirmfenster z. B. nach ihrem Verschieben nicht unmittelbar neu gezeichnet. Der Benutzer hat den Eindruck, dass die Anwendung „hängt“.

Deshalb wird bei der Entwicklung solcher Software sehr oft dafür gesorgt, dass die Benutzerblockierung nur kurze Zeit dauert. Dies geschieht durch eine sehr schnelle Verarbeitung des Ereignisses innerhalb des AWT-Prozesses, indem als einziger Schritt ein weiterer Prozess für die eigentliche Ereignisverarbeitung aktiviert wird, der dann simultan zum AWT-Prozess abläuft. Innerhalb dieses Prozesses findet dann z. B. die Kooperation der A- mit der B+C-Komponente statt. Diese faktische Aufhebung der Benutzerblockierung bringt jedoch die Gefahr von Inkonsistenzen mit sich. So müssen beispielsweise bei der Bahnhofverwaltungsanwendung die Bahnhofsdaten vollständig in eine Datenstruktur der B+C-Komponente eingelesen sein, bevor eine Sortierung vorgenommen werden kann. Es dürfen deshalb die durch eine Benutzeraktion aktivierten Kontrollflüsse, die in der A-Komponente durch die Ereignisbearbeitungsprozesse repräsentiert sind, den Code der B+C-Komponente nicht simultan durchlaufen. Außerdem ist beim Durchlaufen dieses Codes die Reihenfolge der vom Benutzer ausgelösten Aktivierungen relevant.

Synchronisation durch Überwacher

Die erforderliche Synchronisation der um B+C-Ressourcen konkurrierenden Kontrollflüsse kann prinzipiell an mehreren Stellen erfolgen:

Eine Möglichkeit ist, den Code der B+C-Komponente als kritischen Bereich zu deklarieren. Ein Überwacher sorgt dafür, dass dieser Bereich zu jedem Zeitpunkt nur von einem einzigen Kontrollfluss durchlaufen wird. Solche Überwacher sind durch die bei Java gegebene Umsetzung des Monitor-Konzepts (siehe Abschnitt 3.1.3.3) leicht zu realisieren. Es müssen von solchen Monitoren nur die Schnittstellen der Komponente überwacht werden.

Ein Nachteil dieser Lösung ist jedoch, dass nicht garantiert werden kann, dass die Reihenfolge der Kontrollflüsse beim Durchlaufen durch den Code der B+C-Komponente gleich der Reihenfolge ihrer Aktivierung ist, da es zu Überholungen innerhalb der A-Komponente kommen kann. Deshalb wird in diesem Beispiel und oftmals auch an anderer Stelle ein anderer Weg gewählt: Die Beauftragungen durch Benutzeraktionen, die eine Aktivierung der B+C-Komponente hervorrufen, werden nach dem Durchlaufen durch die AWT-Warteschlange nochmals in einer FIFO-Warteschlange gespeichert, falls während der Aktion die B+C-Komponente schon

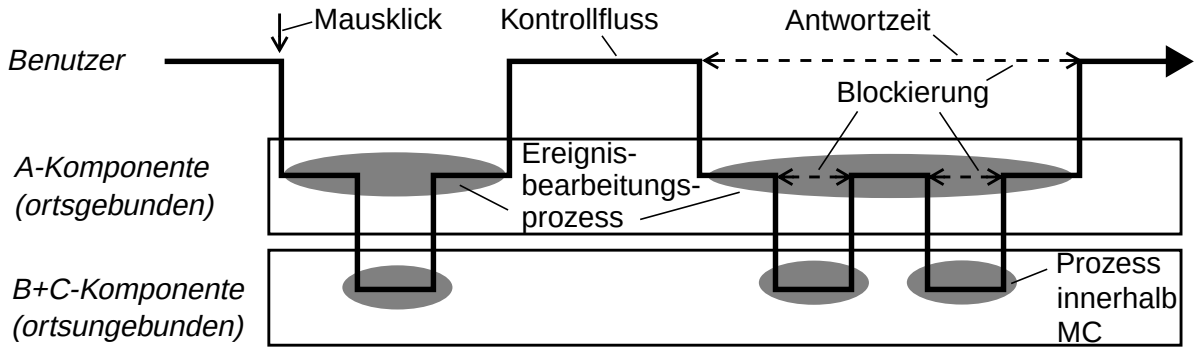


Bild 3.10: Beauftragungen bei einer benutzergesteuerten Anwendung

aktiv ist. Erst wenn die B+C-Komponente wieder inaktiv ist, wird die nächste Beauftragung aus der Warteschlange entfernt und ausgewertet. Es ist hierbei notwendig, dass ein Überwacher das Erzeugen von B+C-relevanten AWT-Ereignissen und die Aktivität einer B+C-Komponente feststellt. Solch ein Überwacher wird in dieser Arbeit auch als Benutzeraktionsüberwacher bezeichnet.

In Bild 3.10 wird ein möglicher Verlauf eines Kontrollflusses innerhalb einer solchen Anwendungskomposition dargestellt. Eine beteiligte Instanz (Benutzer oder eine Komponente) beauftragt gemäß einer gegebenen Diensthierarchie eine weitere Instanz und blockiert solange, bis der Auftrag bearbeitet worden ist. Die Antwortzeit einer Benutzeraktion entspricht der Zeitdauer der Benutzerblockierung. Es läuft nur ein Kontrollfluss durch die beteiligten Instanzen. Folglich existiert bei der Anwendungskomposition zu jedem Zeitpunkt nur maximal ein Prozess, der sich im Zustand `bereit` oder `laufend` befindet, wenn man den nahezu immer ablaufenden AWT-Prozess und Prozesse, die nur die A-Komponente durchlaufen, um z. B. einen Bildschirm zu vergrößern, außer acht lässt. Weitere existente Prozesse sind aufgrund des Einsatzes eines Benutzeraktionsüberwachers blockiert.

Beim betrachteten Beispiel wird folglich eine Synchronisation durch einen für die Anwendungskomposition verantwortlichen Benutzeraktionsüberwacher vorgenommen. Eine Synchronisation mehrerer ablaufender Prozesse durch Java-eigene Monitore kommt nicht zum Einsatz.

Ein von einem Benutzer gesteuerter Kontrollfluss durchläuft i. A. nicht nur Komponenten einer Anwendungskomposition sondern zudem Komponenten, die Systemdienste realisieren.

Kapitel 4

Modellierung von Konfigurationen

Die Betrachtungen in Abschnitt 3.2 bilden die Grundlage einer Methode zur Modellierung von Konfigurationen Verteilter Systeme, die Kompositionen von benutzergesteuerten Anwendungen beinhalten. In diesem Kapitel wird zunächst diese Modellierungsmethode, die eine Bewertung der Systemleistung mit Hilfe der Theorie über Warteschlangennetze ermöglicht, vorgestellt und validiert.

Zur Validierung der Modellierungsmethode werden Messergebnisse bezüglich der Synchronisation von Prozessen bei realen benutzergesteuerten Anwendungen vorgestellt. Die hierauf basierten Überlegungen münden in einer Bestätigung der Aussage von Abschnitt 3.2, dass in vielen Fällen eine benutzergesteuerte Anwendung nur einen einzigen Kontrollfluss beinhaltet. Die Gültigkeit dieser Aussage ist für die Anwendbarkeit der Methode notwendig.

Zum Abschluss dieses Kapitels werden die bei einer Komponentenmigration auftretenden Synchronisationsvorgänge mit Hilfe der Petri-Netzmodellierung betrachtet. Der Einfluss dieser Vorgänge auf die Antwortzeiten bei Benutzeraktionen wird bewertet.

4.1 Modellierung als Geschlossenes Warteschlangennetz

Eine benutzergesteuerte Anwendung wird bei der in diesem Kapitel vorgestellten Modellierungsmethode durch eine Kette in einem GPWN (*Geschlossenes Produktformwarteschlangennetz*) repräsentiert. Für eine vollständige Erläuterung der Methode ist es zunächst notwendig, auf die Theorien über GPWNs näher einzugehen. GWNs (*Geschlossene Warteschlangennetze*) sind Warteschlangennetze, die keinen externen Verkehr aufnehmen bzw. keinen internen Verkehr abführen. GPWNS sind GWNs, bei denen Produktformlösungen zur Berechnung von Leistungsgrößen anwendbar sind.

4.1.1 Geschlossene Produktformwarteschlangennetze (GPWNs)

In diesem Abschnitt werden die Theorien über GPWNs in der für diese Arbeit notwendigen Tiefe vorgestellt. In [10, 62, 81,137] findet man ausführliche und vollständige Beschreibungen dieser Theorien. Die in diesem Abschnitt vorgestellten Theorien werden in Anhang A veranschaulicht.

4.1.1.1 Leistungsgrößen in einem Bediensystem

Im Folgenden werden Komponentenablaufumgebungen als Bediensysteme mit einer Bedieneinheit und einer unbegrenzten Warteschlangenlänge modelliert. Der Zustand eines solchen Bediensystems kann durch Leistungsgrößen, wie die mittlere Ankunftsrate λ, die bei einem verlustfreien, stabilen System dem Durchsatz entspricht, die mittlere Aufenthaltsdauer $\bar{t}$ und die mittlere Anzahl von im System vorhandenen Anforderungen $\bar{k}$ beschrieben werden. Die Wahl der hier verwendeten Begriffe orientiert sich an [57]. Für ein Bediensystem gilt nach dem Theorem von Little die Gleichung

$$\bar{k} = \lambda \bar{t}. \tag{4.1}$$

Die mittlere Anzahl von Anforderungen $\bar{k}$ errechnet sich mit den Wahrscheinlichkeiten $p(k)$, dass k Anforderungen im Bediensystem sind, mit der Gleichung

$$\bar{k} = \sum_{k=1}^{\infty} k p(k). \tag{4.2}$$

Mit der mittleren Bedienrate μ errechnet sich die Auslastung ρ mit

$$\rho = \frac{\lambda}{\mu}. \tag{4.3}$$

Die Auslastung ρ errechnet sich zudem durch

$$\rho = 1 - p(0). \tag{4.4}$$

4.1.1.2 Zustandsraum eines Stochastischen Prozesses

Ein GWN ist ein Netz aus N Bediensystemen (Knoten), wobei einzelne Anforderungen nach ihrer Bedienung zu einem anderen Bediensystem gelangen. Die Anzahl der sich im Netz befindenden Anforderungen k ist konstant, da keine Anforderungen von außen ins Netz gelangen und keine das Netz verlassen.

Wenn k_i die Anzahl der Anforderungen im Knoten i beschreibt, dann gilt

$$k = \sum_{i=1}^{N} k_i . \tag{4.5}$$

Die in dieser Arbeit betrachteten Netze befinden sich zu jedem Zeitpunkt in einem Zustand, der durch einen Vektor $\underline{S}$ mit k_i als Elemente beschrieben werden kann. Es gilt

$$\underline{S} = (k_1, \ldots, k_N) . \tag{4.6}$$

Die Vorgänge, die Zustandsübergänge verursachen, können als Stochastische Prozesse aufgefasst werden. Ein Stochastischer Prozess $\{X(t), t \geq 0\}$ ist eine Familie von Zufallsvariablen *X*, welche vom Parameter *t*, d. h. der Zeit, abhängen. Er wird durch seinen Zustandsraum, seinen Parameter *t* und den Abhängigkeiten zwischen den Zufallsvariablen $X(t)$ für verschiedene *t* gekennzeichnet. In dieser Arbeit werden nur homogene, zeit- und zustandsdiskrete Stochastische Prozesse mit abzählbarem Zustandsraum betrachtet. Der Begriff homogen bedeutet, dass die Übergangsraten zeitlich konstant sind. Ein Stochastischer Prozess wird als Markoff-Prozess bezeichnet, wenn seine zukünftige Entwicklung nur vom gegenwärtigen Zustand abhängt. Die Abstände zwischen einander folgenden Zustandsänderungen sind negativ-exponentiell verteilt. Ein Markoff-Prozess mit abzählbarem Zustandsraum wird auch als Markoff-Kette bezeichnet. [47, 75]

Ein Markoff-Prozess heisst stationär, wenn sein Verhalten unabhängig gegenüber einer Zeitverschiebung ist. Wenn ein Zustand $\underline{S}^n$ mit einer Flussrate $\lambda(\underline{S}^n, \underline{S}^m)$ in den Zustand $\underline{S}^m$ übergeht, dann gelten mit

$$\sum_{\forall m \neq n} \lambda(\underline{S}^n, \underline{S}^m) = \sum_{\forall m \neq n} \lambda(\underline{S}^m, \underline{S}^n) \tag{4.7}$$

die Gleichungen für das globale Gleichgewicht.

Um Produktformlösungen anwenden zu können, müssen zusätzlich noch lokale Gleichgewichtsbedingungen gelten [78]: Die Flussrate von einem Zustand $\underline{S}$ zu einem anderen Zustand wegen einer Anforderung, die Knoten *j* verlässt, muss gleich der Flussrate zu dem Zustand $\underline{S}$ von einem anderen Zustand aufgrund einer Anforderung, die in Knoten *j* ankommt, sein.

4.1.1.3 Theorem von Gordon und Newell

Wenn p_{ij} die Wahrscheinlichkeit beschreibt, dass eine im Knoten *i* bediente Anforderung zum Knoten *j* gelangt, dann gilt für die Ankunftsrate λ_i von Knoten *i* die Gleichung

$$\lambda_i = \sum_{j=1}^{N} \lambda_j p_{ji} . \tag{4.8}$$

Für die folgenden Herleitungen wird ein Besuchskoeffizient v_i mit

$$v_i = c\lambda_i \tag{4.9}$$

eingeführt, wobei c einer Konstanten entspricht.

Mit $p(\underline{S})$ wird die Wahrscheinlichkeit, dass sich das Netz im Zustand $\underline{S}$ befindet, beschrieben. Nach dem Theorem von Gordon und Newell [42] lässt sich diese Wahrscheinlichkeit bei negativ-exponentiell verteilten Bedienzeiten und FCFS-Warteschlangendisziplinen (*First Come First Served*) in den Bedieneinheiten durch die Produktform

$$p(\underline{S}) = \frac{1}{G(k)} \prod_{i=1}^{N} \left(\frac{v_i}{\mu_i} \right)^{k_i} \tag{4.10}$$

berechnen. Die Normalisierungskonstante $G(k)$ ergibt sich aus der Bedingung, dass sich die Wahrscheinlichkeiten aller Systemzustände zu eins summieren. Es gilt

$$G(k) = \sum_{\sum_{i=1}^{N} k_i = k} \prod_{i=1}^{N} \left(\frac{v_i}{\mu_i} \right)^{k_i}. \tag{4.11}$$

In [10] wird beschrieben, wie $G(k)$ auch durch eine Faltungssumme berechnet werden kann.

Zum Lösen der Gleichung (4.10) muss man bei gegebenen μ_i die Gleichungen (4.8), (4.9) und (4.11) anwenden. Die Gleichung (4.8) beschreibt hierbei ein LGS (*Lineares Gleichungssystem*) mit $N - 1$ unabhängigen Gleichungen. Beim Auflösen der Gleichungen erhält man folglich keine Absolutwerte von v_i, sondern Verhältnisse von v_i zu v_j, die jedoch ein Lösen der Gleichung (4.10) erlauben.

Nun lassen sich die Wahrscheinlichkeiten aller Knotenzustände durch

$$p_i(k_i) = \sum_{\sum_{j=1, j \neq i}^{N} k_j = k - k_i} p(\underline{S}) \tag{4.12}$$

berechnen.

4.1.1.4 Theorem von Basket, Chandy, Muntz und Palacios (BCMP)

Bei den bisherigen Betrachtungen war bei einer Bedieneinheit die Verteilung der Bedienzeiten für jeden Auftrag gleich. Im Folgenden werden die Anforderungen R verschiedenen Klassen zugeordnet, wobei die Bedienzeitverteilungen je Bedieneinheit und Klasse verschieden sein können.

Beim BCMP-Theorem (*Basket, Chandy, Muntz und Palacios*, [2]) können verschiedene Anforderungsklassen in einem System existieren. Das Theorem ist gültig für Bediensysteme

- mit einer FCFS-Warteschlangendisziplin und negativ-exponentiell verteilten Bedienzeiten, wobei für alle Anforderungen gleiche mittlere Bediendauern vorliegen müssen (Typ A),
- mit einer PS-Warteschlangendisziplin (*Processor Sharing*) und verschiedenen Bedienzeitverteilungen je Auftragsklasse, wobei die Verteilung jeweils eine rationale Laplace-Transformierte besitzen muss (Typ B),
- mit einer LCFS-PR-Warteschlangendisziplin (*Last Come First Served with Preemptive Resume*) und Bedienzeitverteilungen wie im vorherigen Punkt, wenn eine ankommende Anforderung die momentan bediente unterbricht (Typ C), oder
- mit so vielen Bedieneinheiten, dass ankommende Anforderungen unmittelbar bedient werden, und Bedienzeitverteilungen wie im vorherigen Punkt (Typ D).

Für Bediensysteme der Typen A, B und C mit einer Bedieneinheit gilt die Gleichung

$$p(\underline{S}) = \frac{1}{G(\underline{k})} \prod_{i=1}^{N} k_i! \prod_{r=1}^{R} \frac{1}{k_{ir}!} \left(\frac{\nu_{ir}}{\mu_{ir}} \right)^{k_{ir}}. \tag{4.13}$$

Für Bediensysteme des Typs D gilt die Gleichung

$$p(\underline{S}) = \frac{1}{G(\underline{k})} \prod_{i=1}^{N} \prod_{r=1}^{R} \frac{1}{k_{ir}!} \left(\frac{\nu_{ir}}{\mu_{ir}} \right)^{k_{ir}}. \tag{4.14}$$

Mit dem Vektor $\underline{S}$ wird wiederum der Netzzustand beschrieben. Der Zustand $\underline{S}_i$ eines Netzknotens i hängt nun von der Anzahl der Anforderungen k_{ir} aller Klassen r ab. Es gilt

$$\underline{S} = (\underline{S}_1, \ldots, \underline{S}_N) \tag{4.15}$$

und

$$\underline{S}_i = (k_{i1}, \ldots, k_{iR}). \tag{4.16}$$

Durch den Vektor $\underline{k}$ wird zudem die Anzahl der Anforderungen im Netz für jede Klasse beschrieben. Es gilt

$$\underline{k} = \left(\sum_{i=1}^{N} k_{i1}, \ldots, \sum_{i=1}^{N} k_{iR} \right) = (e(\underline{k}, 1), \therefore, e(\underline{k}, R)), \tag{4.17}$$

wobei $e(\underline{k}, r)$ eine Funktion ist, die das r-te Element des Vektors $\underline{k}$ zurückliefert.

Desweiteren gilt für die Anzahl der Anforderungen k_i im Knoten *i* die Gleichung

$$k_i = \sum_{r=1}^{R} k_{ir}. \tag{4.18}$$

Die Besuchskoeffizienten v_{ir} lassen sich über ein Gleichungssystem berechnen, bei dem $p_{ir,\,js}$ die Wahrscheinlichkeit darstellt, dass eine Anforderung der Klasse *r* nach ihrer Bedienung im Knoten *i* als Anforderung der Klasse *s* zum Knoten *j* gelangt.

Mit

$$v_{ir} = c\lambda_{ir} \tag{4.19}$$

gilt

$$v_{ir} = \sum_{j=1}^{N} \sum_{s=1}^{R} v_{js} p_{js,\,ir}. \tag{4.20}$$

Die Normalisierungskonstante berechnet man (für die Fälle A bis C) mit

$$G(\underline{k}) = \sum_{\sum_{i=1}^{N} \underline{S}_i = \underline{k}} \prod_{i=1}^{N} k_i! \prod_{r=1}^{R} \frac{1}{k_{ir}!} \left(\frac{v_{ir}}{\mu_{ir}}\right)^{k_{ir}}. \tag{4.21}$$

4.1.1.5 Produktformlösungen bei zustandsabhängigen Knotenabgangsraten

In [48, 62, 117] wird eine Produktformlösung für Netze vorgestellt, deren Zustandsentwicklung durch eine Markoff-Kette beschrieben wird und bei denen die Rate für das Bewegen von Anforderungen zwischen zwei Knoten vom Netzzustand abhängt. Im Folgenden wird sie auch als Produktformlösung nach Kelly bezeichnet.

Eine Anforderungsklasse

Die Produktformlösung für Netze mit einer Anforderungsklasse lautet

$$p(\underline{S}) = \frac{1}{G(k)} \Phi(\underline{S}) \prod_{i=1}^{N} \left(\frac{v_i}{\mu_i}\right)^{k_i}. \tag{4.22}$$

Der zustandsabhängige Parameter $\Phi(\underline{S})$ berechnet sich mit

$$\Phi(\underline{S}) = \frac{1}{\prod_{i=1}^{N} \prod_{q=1}^{k_i} \Phi_i(q)}. \tag{4.23}$$

Die Rate $\lambda(\underline{S}, \underline{S} - \underline{1}_i + \underline{1}_j)$ ist die Übergangsrate vom Zustand $\underline{S}$ zum Zustand $\underline{S} - \underline{1}_i + \underline{1}_j$, wobei der Zustandswechsel durch das Bewegen einer Anforderung vom Knoten *i* zu Knoten *j* zustandekommt. Es gilt die Produktformlösung, wenn sich diese Übergangsrate durch die Gleichung

$$\lambda(\underline{S}, \underline{S} - \underline{1}_i + \underline{1}_j) = \Phi_i(k_i)\frac{\mu_i}{\nu_i} \tag{4.24}$$

berechnen lässt. Man sieht, dass für alle Netze bei denen das Gordon und Newell-Theorem gilt, der Parameter $\Phi(\underline{S})$ konstant eins ist.

Mehrere Anforderungsklassen

In [48] wird eine Erweiterung für Netze mit mehreren Anforderungsklassen vorgestellt. Es gilt

$$p(\underline{S}) = \frac{1}{G(\underline{k})}\Phi(\underline{S})\prod_{i=1}^{N}\prod_{r=1}^{R}\left(\frac{\nu_{ir}}{\mu_{ir}}\right)^{k_{ir}}. \tag{4.25}$$

Diese Produktformlösung existiert, wenn man $\Phi(\underline{S})$ für jede Klasse *r* mit

$$\Phi(\underline{S}) = \frac{1}{\prod_{i=1}^{N}\prod_{q=0}^{k_{ir}-1}\Phi_{ir}(\underline{S}_i - q\underline{1}_r)} \tag{4.26}$$

beschreiben kann. Außerdem muss

$$\lambda(\underline{S}, \underline{S} - \underline{1}_{ir} + \underline{1}_{jr}) = \Phi_{ir}(\underline{S}_i)\frac{\mu_{ir}}{\nu_{ir}} \tag{4.27}$$

gelten.

Veranschaulichung durch ein Beispiel

Um diesen Produktformansatz zu veranschaulichen, wird ein einfaches Beispiel betrachtet: Ein Netz besteht aus den beiden Bediensystemen 1 und 2, die jeweils mit einer PS-Warteschlangendisziplin arbeiten. Es existieren zwei Klassen 1 und 2, wobei die Klasse 1 eine Anforderung und die Klasse 2 zwei Anforderungen beinhaltet. Alle Anforderungen werden nach Fertigbedienung in einem Bediensystem zum anderen Bediensystem gereicht. In Bild 4.1 ist der dazugehörige Zustandsraum dargestellt. Bemerkenswert ist das Vorhandensein einer zustandsabhängigen Knotenabgangsrate bezüglich der Anforderungen einer Klasse. So verlässt z. B. bei $\underline{S}_1 = (1, 0)$ die Anforderung der Klasse 1 den Knoten 1 mit einer Rate μ_{11}, bei $\underline{S}_1 = (1, 1)$ mit einer Rate $\mu_{11}/2$ und bei $\underline{S}_1 = (1, 2)$ mit einer Rate $\mu_{11}/3$.

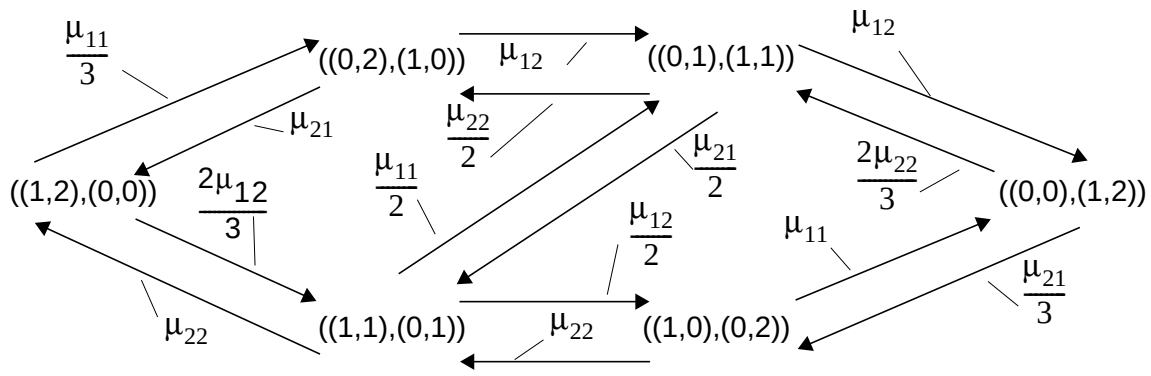


Bild 4.1: Zum Beispielnetz gehörender Zustandsraum der Markoff-Kette

Es gilt

$$\Phi((1,2),(0,0)) = \frac{1}{\Phi_{11}(1,2)} = \frac{1}{\frac{1}{3}} = \frac{1}{\Phi_{12}(1,1)\Phi_{12}(1,2)} = \frac{1}{\frac{1}{2}\frac{2}{3}} = 3,$$

$$\Phi((0,2),(1,0)) = \frac{1}{\Phi_{21}(1,0)} = \frac{1}{\Phi_{12}(0,1)\Phi_{12}(0,2)} = 1 \text{ und}$$

$$\Phi(((0,1)),(1,1)) = \frac{1}{\Phi_{21}(1,1)} = \frac{1}{\frac{1}{2}} = \frac{1}{\Phi_{12}(0,1)\Phi_{12}(1,1)} = \frac{1}{\frac{1}{2}} = 2.$$

Aus Symmetriegründen erhält man zudem

$$\Phi((0,0),(1,2)) = 3,$$

$$\Phi((1,0),(0,2)) = 1 \text{ und}$$

$$\Phi((1,1),(0,1)) = 2.$$

Wenn man die Werte für $\Phi(\underline{S})$ in Gleichung (4.25) einsetzt, erhält man die gleichen Zustandswahrscheinlichkeiten, wie wenn man mit Gleichung (4.13) das BCMP-Theorem anwendet.

4.1.1.6 Mittelwertanalyse (MVA)

Durch das MVA-Verfahren (*Mean Value Analysis*, [107]) lassen sich sämtliche Leistungsgrößen durch die Betrachtung von Mittelwerten ohne vorherige Bestimmung der Zustandswahrscheinlichkeiten berechnen.

Wenn keine Anforderungen im Netz sind, dann gilt für die mittlere Anzahl von Anforderungen der Klasse *r* im Knoten *i*

$$\bar{k}_{ir}(\underline{0}) = 0. \tag{4.28}$$

Die mittlere Aufenthaltsdauer läßt sich für Knoten der Typen A, B und C durch

$$\bar{t}_{ir}(\underline{k}) = \frac{1}{\mu_{ir}}(1 + \bar{k}_i(\underline{k} - \underline{1}_r)) \tag{4.29}$$

berechnen. Für Knoten des Typs D gilt

$$\bar{t}_{ir}(\underline{k}) = \frac{1}{\mu_{ir}}. \tag{4.30}$$

Die mittlere Gesamtanzahl von Anforderungen $\bar{k}_i(\underline{k})$ in einem Knoten *i* ergibt sich durch

$$\bar{k}_i(\underline{k}) = \sum_{r=1}^{R} \bar{k}_{ir}(\underline{k}). \tag{4.31}$$

Nach Littles Theorem (4.1) und Gleichung (4.3) gilt

$$\bar{k}_{ir}(\underline{k}) = \lambda_{ir}\bar{t}_{ir}(\underline{k}) = \frac{v_{ir}\bar{t}_{ir}(\underline{k})}{c}. \tag{4.32}$$

Für die konstante Anzahl $e(\underline{k}, r)$ der sich im Netz befindenden Anforderungen $\underline{k}$ einer Klasse *r* gilt zu jedem Zeitpunkt

$$e(\underline{k}, r) = \sum_{i=1}^{N} k_{ir} = \sum_{i=1}^{N} \bar{k}_{ir}(\underline{k}) = \sum_{i=1}^{N} \frac{v_{ir}\bar{t}_{ir}(\underline{k})}{c}. \tag{4.33}$$

Durch Substitution der Konstanten *c* aus den Gleichungen (4.32) und (4.33) erhält man

$$\bar{k}_{ir}(\underline{k}) = \frac{\bar{t}_{ir}(\underline{k}) \sum_{j=1}^{N} k_{jr}}{\sum_{j=1}^{N} \frac{v_{jr}}{v_{ir}} \bar{t}_{jr}(\underline{k})}. \tag{4.34}$$

Die mittleren Aufenthaltsdauern für eine gegebene Anzahl von Anforderungen sind nach (4.29) rekursiv berechenbar, indem solange die Anzahl der Anforderungen einer Klasse um eins reduziert wird, bis kein Auftrag mehr im Netz existiert. Es existieren weitere Verfahren wie z. B. LBANC (*Local Balance Algorithm for Normalizing Constants*), die der Mittelwertanalyse ähnlich sind.

4.1.1.7 MVA-basierte Heuristiken

Die vorgestellten Algorithmen besitzen i. A. eine so hohe Komplexität bezüglich der Rechenzeit und des Speicherplatzbedarfs, dass ihre Anwendbarkeit innerhalb eines zur Systemlaufzeit ablaufenden Verfahrens schon bei kleinen Netzen auszuschließen ist. Wie u. a. auch in [80]

gezeigt wird, gilt für die Anzahl der Rechenschritte $\Re$ beim MVA-Algorithmus, bei dem rekursiv aus einer Anforderungspopulation immer eine Anforderung herausgenommen wird, bis das Netz leer ist, die Beziehung

$$\Re \sim \prod_{r=1}^{R} (e(\underline{k}, r) + 1) . \tag{4.35}$$

Selbst bei nur einer Anforderung je Klasse (d. h. $e(\underline{k}, r) = 1, \forall r$) gilt

$$\Re \sim 2^R . \tag{4.36}$$

Im Folgenden werden Heuristiken vorgestellt, mit denen eine Näherungslösung berechnet werden kann. Der Vorteil ihres Einsatzes liegt darin, dass die Komplexität bezüglich der Rechenzeit und des Speicherplatzbedarfs beherrschbarer ist.

Core-Algorithmus (Bard-Schweitzer-Algorithmus)

Bei dem in [113] vorgestellten Core-Algorithmus schätzt man die mittlere Anzahl von Anforderungen je Knoten und Klasse in einem ersten Schritt grob ab und versucht dann, iterativ den exakten Werten möglichst nahe zu kommen. Für die erste Abschätzung gilt

$$\bar{k}_{ir}(\underline{k}) = \frac{e(\underline{k}, r)}{N_r} . \tag{4.37}$$

Der Parameter N_r beschreibt hierbei die Anzahl der Knoten, in denen sich mindestens eine Anforderung der Klasse *r* befinden kann.

Weiter schätzt man die mittlere Anzahl von Anforderungen einer Klasse *r* im Knoten *i* ab, indem man ein System betrachtet, bei dem eine Anforderung herausgenommen wurde. Für die Abschätzung gilt:

$$\bar{k}_{ir}(\underline{k} - \underline{1}_s) = \begin{cases} \dfrac{(e(\underline{k}, r) - 1)\bar{k}_{ir}(\underline{k})}{e(\underline{k}, r)} & r = s \\ \bar{k}_{ir}(\underline{k}) & r \neq s \end{cases} . \tag{4.38}$$

Man kann nun die mittlere Anzahl von Anforderungen und die mittleren Aufenthaltsdauern näherungsweise durch Iterationen berechnen: Für jedes $\bar{k}_{ir}(\underline{k})$ werden mit (4.38) alle möglichen $\bar{k}_{ir}(\underline{k} - \underline{1}_s)$ abgeschätzt, die für die Berechnung von $\bar{t}_{ir}(\underline{k})$ durch (4.29) benötigt werden. Mit (4.34) werden dann alle $\bar{k}_{ir}(\underline{k})$ für den nächsten Iterationsschritt berechnet.

Linearizer-Algorithmus

In [15] wird eine Verbesserung des Core-Algorithmus vorgeschlagen. Es wird ein Quotient $f_{ir}(\underline{k})$ und ein Differenzquotient $d_{irs}(\underline{k})$ eingeführt, so dass

$$f_{ir}(\underline{k}) = \begin{cases} \dfrac{\bar{k}_{ir}(\underline{k})}{e(\underline{k}, r)} & e(\underline{k}, r) \neq 0 \\ 0 & e(\underline{k}, r) = 0 \end{cases} \tag{4.39}$$

und

$$d_{irs}(\underline{k}) = f_{ir}(\underline{k} - \underline{1}_s) - f_{ir}(\underline{k}) \tag{4.40}$$

gilt.

Wenn man diese beiden Gleichungen entsprechend umformt, dann erhält man

$$\bar{k}_{ir}(\underline{k} - \underline{1}_s) = \begin{cases} (e(\underline{k}, r) - 1)\left(\dfrac{\bar{k}_{ir}(\underline{k})}{e(\underline{k}, r)} + d_{irr}(\underline{k})\right) & r = s \\ \bar{k}_{ir}(\underline{k}) + e(\underline{k}, r) d_{irs}(\underline{k}) & r \neq s \end{cases} . \tag{4.41}$$

Gleichung (4.41) entspricht Gleichung (4.38), wenn $d_{irs}(\underline{k}) = 0, \forall i, r, s$ ist. Der Linearizer-Algorithmus funktioniert folgendermaßen:

1. Zunächst wird die mittlere Anzahl von Anforderungen je Knoten und Klasse mit Hilfe der Gleichung (4.37) abgeschätzt. Am Anfang gilt $d_{irs}(\underline{k}) = 0, \forall i, r, s$.
2. Es wird der Core-Algorithmus ausgeführt, wobei die Gleichung (4.38) durch die Gleichung (4.41) ersetzt wird.
3. Es werden alle $d_{irs}(\underline{k})$ mit den Gleichungen (4.39) und (4.40) neu berechnet und Schritt 2 wiederholt.

4.1.2 Modellierung von Anwendungen als Ketten in GPWNs

In diesem Abschnitt wird eine Methode zur Modellierung von Anwendungen als sogenannte Ketten in GPWNs vorgestellt.

4.1.2.1 Verfahren zur Erlangung einer Warteschlangennetzbeschreibung

In [41, 61, 82, 118] wird mit bekannten Modellierungsmethoden die Leistungsfähigkeit von komponentenbasierten Verteilten Systemen bewertet. In einigen dieser Arbeiten wird die strukturelle Konfiguration des Verteilten Systems mit Hilfe von UML beschrieben und diese

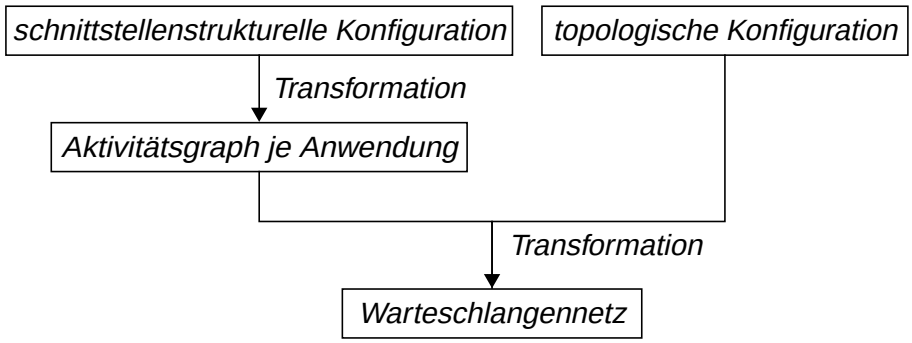


Bild 4.2: Transformation von Konfigurationsbeschreibungen

Beschreibung zusätzlich mit Leistungsgrößen angereichert. Sie wird dann in eine Warteschlangennetzbeschreibung transformiert, um daraus die gesuchten Leistungsgrößen zu berechnen. In [61] werden auf diese Weise CORBA-basierte Verteilte Systeme modelliert und bewertet. In [41] wird von einer ADL-Beschreibung ausgegangen, die in eine UML-Beschreibung überführt wird. Die meisten Arbeiten haben die Erstellung eines Software-Werkzeuges zum Ziel, so dass mit Hilfe einer grafischen Oberfläche das System beschrieben und automatisch in eine andere Beschreibungsform transformiert werden kann. Über die grafische Oberfläche können die jeweiligen Beschreibungen mit zusätzlicher Information angereichert werden. In einem letzten Schritt wird die vorliegende Beschreibung in eine Warteschlangennetzbeschreibung transformiert und eine Leistungsbewertung vorgenommen.

Im Gegensatz dazu wird im Folgenden ein Verfahren zur Berechnung eines Warteschlangennetzes aus der vorhandenen Konfiguration vorgestellt, das auch zur Systemlaufzeit arbeiten kann. Dieses Verfahren soll hierbei einer zentralen Konfigurationsverwaltung ein Abbild der strukturellen und topologischen Konfiguration liefern. In Bild 4.2 ist der prinzipielle Ansatz zur Transformation dieser Konfigurationsabbilder zu einem Warteschlangennetz dargestellt. Mit dem Aktivitätsgraphen wird hierbei eine Zwischenstufe bei der Transformation eingeführt. Er beschreibt eine Anwendung und beinhaltet Informationen über die Belastung der Komponenten einer Anwendung durch ihre Beauftragungen.

4.1.2.2 Aktivitätsgraph

Wie in Bild 4.3 dargestellt ist, kann für jede Anwendung ein Graph mit den Komponenten als Knoten und deren möglichen Bindungen als Kanten bestimmt werden. Wenn innerhalb einer Anwendung Systemdienste genutzt werden, dann wird der Aktivitätsgraph um zusätzliche Knoten, die Systemdienste anbietende Komponenten repräsentieren, erweitert. Wenn die Dienste gemäß einer vorgegebenen Dienstehierarchie genutzt werden, dann ist dieser Graph zyklenfrei.

Jede Kante des Graphen beschreibt eine Schnittstelle. Sie ist gerichtet von einem Knoten, der eine bezüglich dieser Schnittstelle in der Client-Rolle agierende Komponente repräsentiert, zu einem Knoten, der eine, diese Schnittstelle anbietende Server-Komponente repräsentiert. Jede

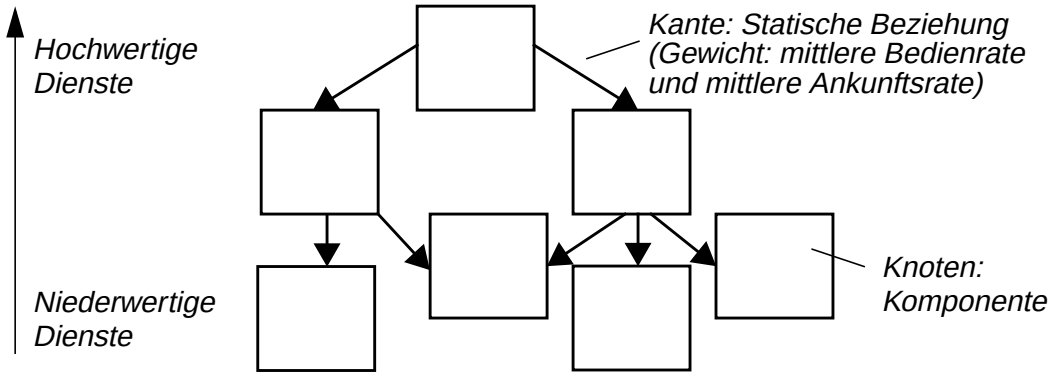


Bild 4.3: Aktivitätsgraph

Kante besitzt ein Gewicht, welches die mittleren Bedienraten und mittleren Ankunftsraten von Server-Beauftragungen repräsentiert.

4.1.2.3 Betrachtung einer benutzergesteuerten Anwendung

Zur Veranschaulichung wird wiederum die in Abschnitt 3.2.2.1 exemplarisch eingeführte Dekomposition einer benutzergesteuerten Anwendung betrachtet. Im Allgemeinen kann zu jedem Zeitpunkt je Anwendung maximal nur ein Prozess im Verteilten System ablaufen, d. h. nur ein Prozess befindet sich entweder im Zustand `bereit` oder `laufend`, und alle anderen existierenden Prozesse sind blockiert. Eine Instanz delegiert die Bearbeitung einer Aufgabe an eine andere Instanz und blockiert solange, bis die Bearbeitung abgeschlossen ist.

Zu dieser Delegationsabfolge gehört die Kette durch den in Bild 4.4 auf der linken Seite dargestellte Aktivitätsgraphen, die der auf der rechten Seite dargestellten Kette mit einer zirkulierenden Anforderung durch ein Warteschlangennetz entspricht. Eine Anforderung wird in dieser Kette zwischen dem Benutzer, der A-, B- und der C-Komponente weitergereicht. Im GWN wird diese Kette durch Bediensysteme gelegt, die die Ablaufumgebungen repräsentieren. Als Denkphase ist ein Zeitraum definiert, in dem der Benutzer die Kontrolle über die Anwendung besitzt. Sie endet, wenn er eine Aktion tätigt. Sie wird durch ein Bediensystem des Typs D

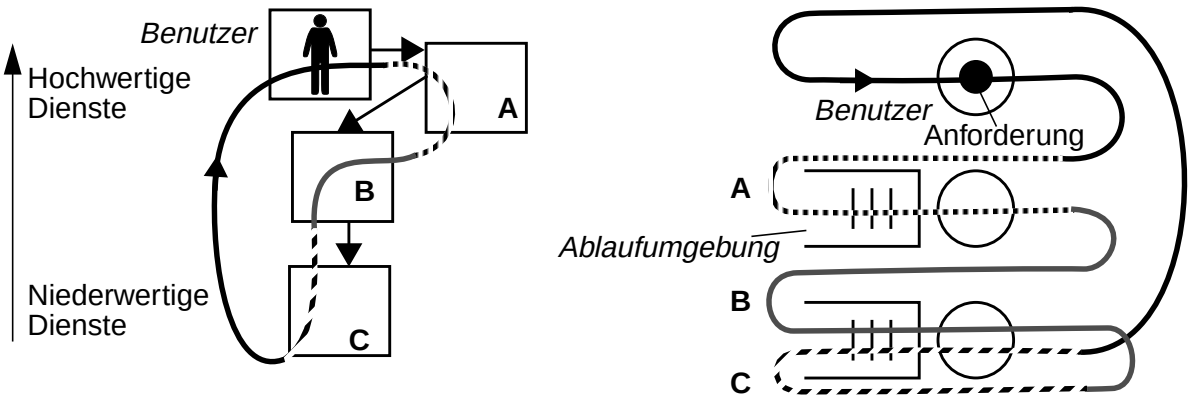


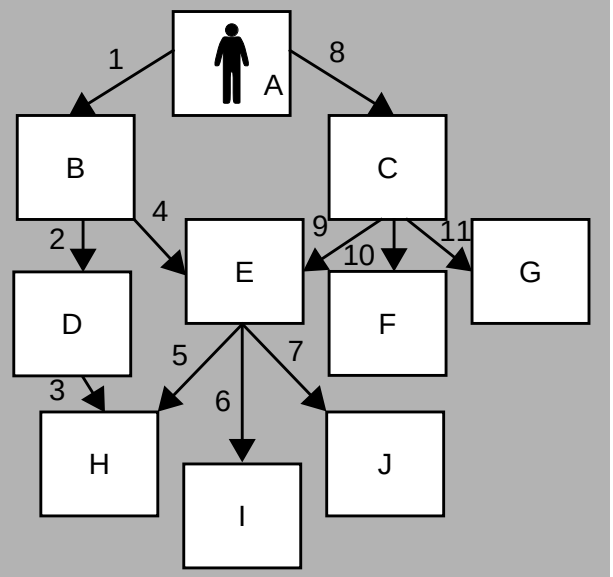
Bild 4.4: Aktivitätsgraph und dazugehörige Kette im Warteschlangennetz

modelliert. Das GWN kann als GPWN angesehen werden, wenn zudem alle Bediensysteme, die eine Ablaufumgebung repräsentieren, vom Typ B sind.

4.1.2.4 Transformation eines Aktivitätsgraphen in eine Kette

In diesem Abschnitt wird ein Algorithmus vorgestellt, der die Transformation eines Aktivitätsgraphen in eine im GPWN liegende Kette ermöglicht. Die Grundidee ist, dass jede Kante des Aktivitätsgraphen eine Client/Server-Schnittstelle repräsentiert, die im GPWN genau einmal durch eine Teilkette berücksichtigt wird. Eine Anwendung wird dann durch eine geschlossene Gesamtkette repräsentiert, die sich aus allen diesen Teilketten zuzüglich einer den Benutzer repräsentierenden Teilkette zusammensetzt. Die Reihenfolge der Teilketten spielt bei ihrer Verknüpfung zur geschlossenen Gesamtkette keine Rolle.

1. Aktivitätsgraph:



2. Tiefensuche:

```
Besuche A
   Speichere (A→B)
   Besuche B
      Speichere (B→D)
      Besuche D
         Speichere (D→H)
         Besuche H
      Speichere (B→E)
      Besuche E
         Speichere (E→H)
         Speichere (E→I)
         Besuche I
         Speichere (E→J)
         Besuche J
   Speichere (A→C)
   Besuche C
      Speichere (C→E)
      Speichere (C→F)
      Besuche F
      Speichere (C→G)
      Besuche G
```

3. Verkettung:

Anforderung in A wegen Denkpause des Benutzers
Anforderung in B mit v_{AB} und μ_{AB} wegen Übergang (A→B)
Anforderung in D mit v_{BD} und μ_{BD} wegen Übergang (B→D)
Anforderung in H mit v_{DH} und μ_{DH} wegen Übergang (D→H)
Anforderung in E mit v_{BE} und μ_{BE} wegen Übergang (B→E)
Anforderung in H mit v_{EH} und μ_{EH} wegen Übergang (E→H)
Anforderung in I mit v_{EI} und μ_{EI} wegen Übergang (E→I)
Anforderung in J mit v_{EJ} und μ_{EJ} wegen Übergang (E→J)
Anforderung in C mit v_{AC} und μ_{AC} wegen Übergang (A→C)
Anforderung in E mit v_{CE} und μ_{CE} wegen Übergang (C→E)
Anforderung in F mit v_{CF} und μ_{CF} wegen Übergang (C→F)
Anforderung in G mit v_{CG} und μ_{CG} wegen Übergang (C→G)
Anforderung kehrt nach A zurück

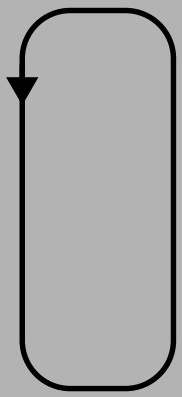


Bild 4.5: Kette durch einen Aktivitätsgraphen (Tiefensuche)

Eine Teilkette, die eine Client/Server-Schnittstelle repräsentiert, wird im GPWN durch das Bediensystem gelegt, das die Ablaufumgebung der Server-Komponente repräsentiert. Es wird jeder Gesamtkette genau eine einzige, in ihr zirkulierende Anforderung zugeordnet.

Das Kantengewicht, das die Häufigkeit und die mittlere Bedienrate von Server-Beauftragungen repräsentiert, gibt an, wie oft es innerhalb einer Teilkette zu Bedienungen kommt und wie lange sie dauern. Hieraus kann ein Besuchskoeffizient *v* und eine mittlere Bedienrate μ bezüglich der zirkulierenden Anforderung je Teilkette berechnet werden.

Dass bei der Verknüpfung der Teilketten zur geschlossenen Gesamtkette die Reihenfolge prinzipiell für die Berechnung von Leistungsgrößen keine Rolle spielt, sieht man an den MVA-Gleichungen (4.29) und (4.31): Es werden nur die Beeinflussungen durch konkurrierende Instanzen aufsummiert. Das Ergebnis der Aufsummierung hängt in keiner Weise von Reihenfolgen ab. Um systematisch jede Kante einmal zu berücksichtigen, können Algorithmen zur Traversierung wie z. B. die Tiefen- (DFS, *Depth First Search*) oder Breitensuche (BFS, *Breadth First Search*) angewendet werden.

In Bild 4.5 ist ein Beispiel für den beschriebenen Algorithmus dargestellt.

4.1.2.5 Besuchshäufigkeit – Zu- und Abflüsse

Die Häufigkeit des Besuchs von Teilketten soll in diesem Abschnitt noch näher untersucht werden. Die Betrachtungen werden exemplarisch anhand des in Bild 4.6 dargestellten Aktivitätsgraphen vorgenommen. Jede Komponente der durch den Aktivitätsgraph repräsentierten Anwendung sei hierbei der Einfachheit halber einer eigenen Ablaufumgebung zugeordnet. Eine mögliche, durch ein GPWN gelegte Kette, die nach einer Traversierung gebildet worden ist, ist ebenfalls in Bild 4.6 dargestellt.

Man sieht, dass man die Reihenfolge der Kettenglieder beliebig vertauschen kann, wenn die Ankunftsraten zu den Bediensystemen immer gleich bleiben. Dies wird durch die gestrichelt

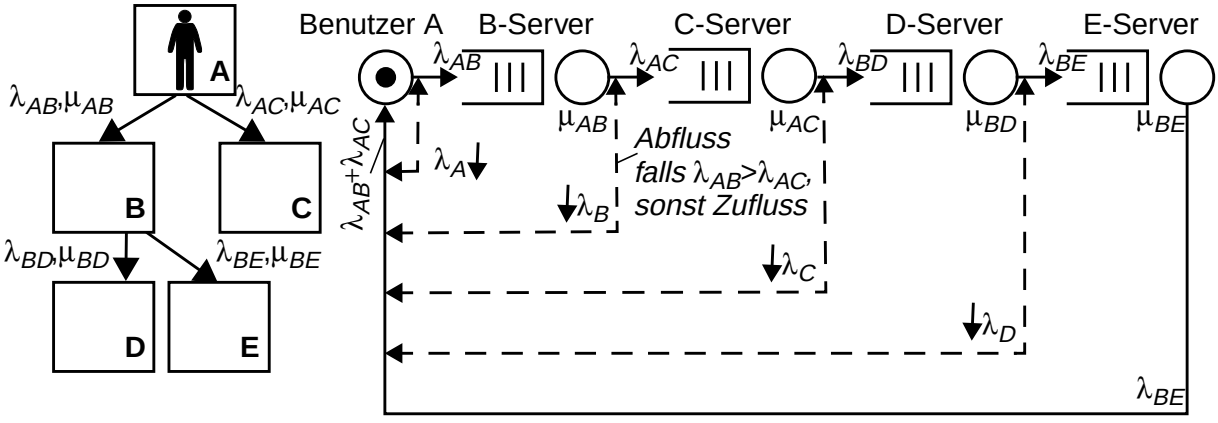


Bild 4.6: Beachtung der Besuchshäufigkeit durch Modellierung von Zu- und Abflüssen

dargestellten Zu- und Abflüsse ermöglicht, die je nach gewählter Kettengliedreihenfolge unterschiedliche Werte annehmen.

Für das betrachtete Beispiel gilt das Gleichgewicht

$$\lambda_{AB} + \lambda_{AC} = \lambda_A + \lambda_B + \lambda_C + \lambda_D + \lambda_{BE}$$

mit

$$\lambda_A = \lambda_{AC},$$

$$\lambda_B = \lambda_{AB} - \lambda_{AC},$$

$$\lambda_C = \lambda_{AC} - \lambda_{BD} \text{ und}$$

$$\lambda_D = \lambda_{BD} - \lambda_{BE}.$$

Nach Gleichung (4.9) können anstatt der Ankunftsraten auch dazu korrespondierende Besuchskoeffizienten berücksichtigt werden. Mit diesen Besuchskoeffizienten sind die bisherigen Gleichungen zur Leistungsgrößenberechnung anwendbar.

4.1.2.6 Komponenten derselben Anwendung auf derselben Ablaufumgebung

Betrachtet man nochmals das Beispiel aus Abschnitt 4.1.2.3, so sieht man, dass die Komponenten A, B und C prinzipiell der gleichen Ablaufumgebung zugeordnet sein können, wobei die Komponenten im Normalfall unterschiedliche mittlere Bedienzeiten besitzen. Folglich muss eine zirkulierende Anforderung ihre Klassenzugehörigkeit wechseln können.

Die Anzahl der Anforderungen bezüglich einer Klasse ist nun nicht mehr zu jedem Zeitpunkt konstant. Wenn eine Anforderung einer bestimmten Klasse zugeordnet werden kann, dann befindet sie sich in einer zur Klasse gehörenden Teilkette innerhalb der die Anwendung repräsentierenden Gesamtkette. Auf der rechten Seite von Bild 4.4 sind die Teilketten in unterschiedlicher Schattierung dargestellt. Der Parameter $\underline{k}$ beschreibt nun die konstante Anzahl der Anforderungen aller Ketten, die jeweils eine Anwendung repräsentieren. Es gilt immer

$$\underline{k} = (1, \ldots, 1). \tag{4.42}$$

Es gibt R Ketten und Z Klassen (d. h. Teilketten) im Netz ($Z \geq R$). Es wird eine Funktion ζ eingeführt, so dass eine Teilkette z Teil der Kette r ist, wenn die Beziehung

$$\zeta(z) = r \tag{4.43}$$

gilt.

4.1.2.7 Ermittlung der Parameter

Als Kantengewicht des Aktivitätsgraphen ist ein Tupel (μ, λ) gegeben, wobei μ die mittlere Bedienrate und λ die mittlere Ankunftsrate von Beauftragungen beschreibt.

Mittlere Ankunftsrate von Beauftragungen

Es sei nochmals erwähnt, dass innerhalb einer Anwendung i. A. den Komponenten unterschiedliche Ankunftsraten ihrer Beauftragungen zuzuordnen sind. So ist es beim Beispiel aus Abschnitt 3.2.2 möglich, dass nicht bei jeder Beauftragung der für die GUI verantwortlichen Komponente A durch den Benutzer eine Beauftragung der Komponenten B und C erfolgt, da die Beauftragung z. B. nur ein Neuzeichnen der grafischen Oberflächen veranlassen kann. Mit Hilfe der Gleichung (4.19) können hierbei aus λ die Besuchskoeffizienten v_{ir} berechnet werden, die dann bei der Berechnung von Leistungsgrößen berücksichtigt werden müssen (siehe auch Abschnitt 4.1.2.5).

Mittlere Bedienrate von Beauftragungen

Die Bearbeitung einer Beauftragung in einer Komponente wird in der Regel wegen Unterbeauftragungen mehrmals blockiert. Als Bedienzeit einer Beauftragung gilt die Zeit, in der der Bearbeitungsprozess sich im Zustand `laufend` befindet.

4.2 Betrachtungen zur Nebenläufigkeit und Synchronistion

Bei der vorgestellten Modellierungsmethode wird von einer Synchronisation durch spezielle Benutzeraktionsüberwacher innerhalb einer benutzergesteuerten Anwendung ausgegangen, so dass ihr Programm nur von einem Kontrollfluss durchlaufen wird. Im Folgenden wird die Relevanz und Anwendbarkeit der Modellierungsmethode nachgewiesen. Es wird in diesem Abschnitt gezeigt, dass die Anwendung der Modellierungsmethode auch bei einer verfeinerten Betrachtung von Synchronisationsvorgängen gerechtfertigt ist. Zum Abschluss wird der Vorgang einer Komponentenmigration näher betrachtet, da hier Synchronisationsvorgänge eine entscheidende Rolle spielen. Es wird bewertet, welchen Einfluss eine Komponentenmigration auf die Leistungsfähigkeit des Systems ausübt.

4.2.1 Anwendbarkeit der vorgestellten Modellierungsmethode

Bei der Betrachtung der exemplarischen, benutzergesteuerten Anwendung in Abschnitt 3.2.2 wurde die Synchronisation durch Überwacher vorgestellt, wobei prinzipiell die Überwacher mit unterschiedlicher Granularität arbeiten können. So können Monitore die Eingänge von Objekten, d. h. deren öffentliche Schnittstellen, überwachen. Ebenso können Monitore die Eingänge von Komponenten überwachen. In vorherigen Kapitel wurde ein Konzept erläutert, bei

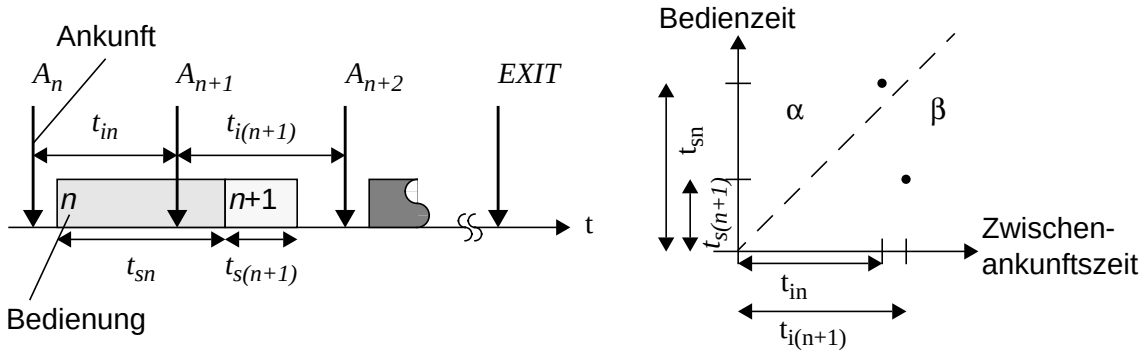


Bild 4.7: Beziehung zwischen einem Zwischenankunftsabstand und einer Bedienzeit

dem ein Benutzeraktionsüberwacher den Eingang einer graphischen Benutzerschnittstelle überwacht. Damit wird gewährleistet, dass nur ein einziger, vom Benutzer gesteuerter Kontrollfluss je Anwendung existiert. Eine Anwendung kann dann durch eine einzige, in einer Kette zirkulierende Anforderung modelliert werden. In diesem Abschnitt wird durch Messungen an existierenden Anwendungen die Relevanz des Konzeptes nachgewiesen.

4.2.1.1 Validierung durch Messungen

Im Rahmen dieser Arbeit wurde eine Messumgebung [64] für eine JVM namens Kaffe [143] implementiert, um die zeitliche Charakteristik von Prozessbereitstellungsabständen und Prozessbedienzeiten von benutzergesteuerten Java-Anwendungen zu messen. Die bei den Messungen verwendete Prozessverwaltung innerhalb der JVM entspricht einem reinen unterbrechenden Prioritätensystem. Es wird auf den Einsatz eines Zeitscheibenverfahrens verzichtet. Die aufgezeichneten Reihen bestehen aus den in Bild 4.7 auf der linken Seite dargestellten Wertepaaren (t_{in}, t_{sn}). Der Parameter t_{in} entspricht dem *n*-ten Wert einer Reihe und steht für den Ankunftsabstand zwischen zwei Ankünften A_n und A_{n+1}, die jeweils eine Prozessbereitstellung verursachen. Der Parameter t_{sn} bildet den zweiten Wert des Paares, und entspricht der zur Anforderung A_n gehörenden Bedienzeit.

Die Ankunft des in Bild 4.7 eingeführten *EXIT*-Signals beendet den Programmablauf. Wenn man die Wertepaare in einem Koordinatensystem wie in Bild 4.7 auf der rechten Seite darstellt, dann kann man Punkte im Bereich α als ein Anzeichen für die Büschelhaftigkeit der Pozessbereitstellungsankünfte einer Anwendung werten. Wenn bei dieser Wertepaardarstellung, die im Folgenden auch als Scatter Plot bezeichnet wird, keine Punkte im Bereich α liegen, dann gilt immer $t_{in} \geq t_{sn}$, d. h. es wird immer mit einer neuen Bereitstellung eines Prozesses gewartet, bis der aktuelle Prozess fertig bedient ist.

4.2.1.2 Vorstellung und Bewertung von Messergebnissen

Im Folgenden werden Messergebnisse vom Ablauf der in Abschnitt 3.2.2.1 eingeführten Anwendung zur Verwaltung von Bahnhofsdaten und zusätzlich vom Ablauf eines WWW-Browsers namens NetClue [141] vorgestellt.

Bei den in Bild 4.8 dargestellten Scatter Plots für jeweils einen Ablauf der beiden monolithischen Anwendungen ist ersichtlich, dass durchaus Punkte im Bereich α liegen. Die Gründe hierfür liegen ausschließlich in der der Realisierung der verwendeten Java-Standard-Grafikbibliothek Swing:

- Realisierung von Timern

  Bei jeder Anwendung mit einer Swing-GUI existiert eine Swing-Timer-Warteschlange, in der Timer sortiert nach ihrem Ablaufzeitpunkt eingetragen sind. Durch diese Timer-Warteschlange können mehrere Timer auf einen einzigen innerhalb der JVM realisierten Timer abgebildet werden. Ein für die Timer-Warteschlange verantwortlicher Prozess wird hierbei immer für die Zeitdauer des Ablaufs des JVM-eigenen Timers blockiert. Die Ablaufzeit dieses JVM-eigenen Timers entspricht der Ablaufzeit des an der ersten Stelle gespeicherten Timers. Bei jedem Timer-Ablauf wird der erste Eintrag aus der Timer-Warteschlange entfernt.

  Der für die Timer-Warteschlange zuständige Prozess ist nebenläufig, besitzt die gleiche Priorität wie der AWT-Ereignisverarbeitungsprozess und kann diesen deshalb, wenn kein Zeitscheibenverfahren bei der Prozessverwaltung angewendet wird, nicht unterbrechen. Bei der Verwendung einer Swing-basierten GUI wird sehr oft ein Timer eingesetzt, um zu entscheiden, ob zwei hintereinander getätigte Mausklicks als Doppelklick zu werten sind.

- Realisierung modaler Dialogboxen

  Modale Dialogboxen erzwingen eine Benutzereingabe. Der Programmablauf wird solange angehalten, bis die angeforderten Eingaben getätigt und danach alle durch die Eingabe festgelegten Aktionen durchgeführt wurden und schließlich die Dialogbox wieder geschlossen wurde. Bei der Anwendung zur Verwaltung von Bahnhofsdaten werden modale Dialogboxen für die Eingabe von Dateinamen verwendet, wenn eine Datei mit einem Bahnhofsdatensatz eingelesen werden soll. Bei dem WWW-Browser NetClue werden modale Dialogboxen eingesetzt, um z. B. die Adresse einer HTTP-Instanz einzugeben und diese zu laden.

  Nebenläufigkeiten kommen vor, wenn aufgrund der Eingabe eine Aktion, wie z. B. eine Datei oder eine HTTP-Instanz einzulesen, getätigt wird, aber simultan das Auslösen weiterer Vorgänge, die unabhängig vom eigentlichen Programmablauf sind, (z. B. das Verändern der Größe oder das Neuzeichen des Bildschirmfensters) erlaubt ist.

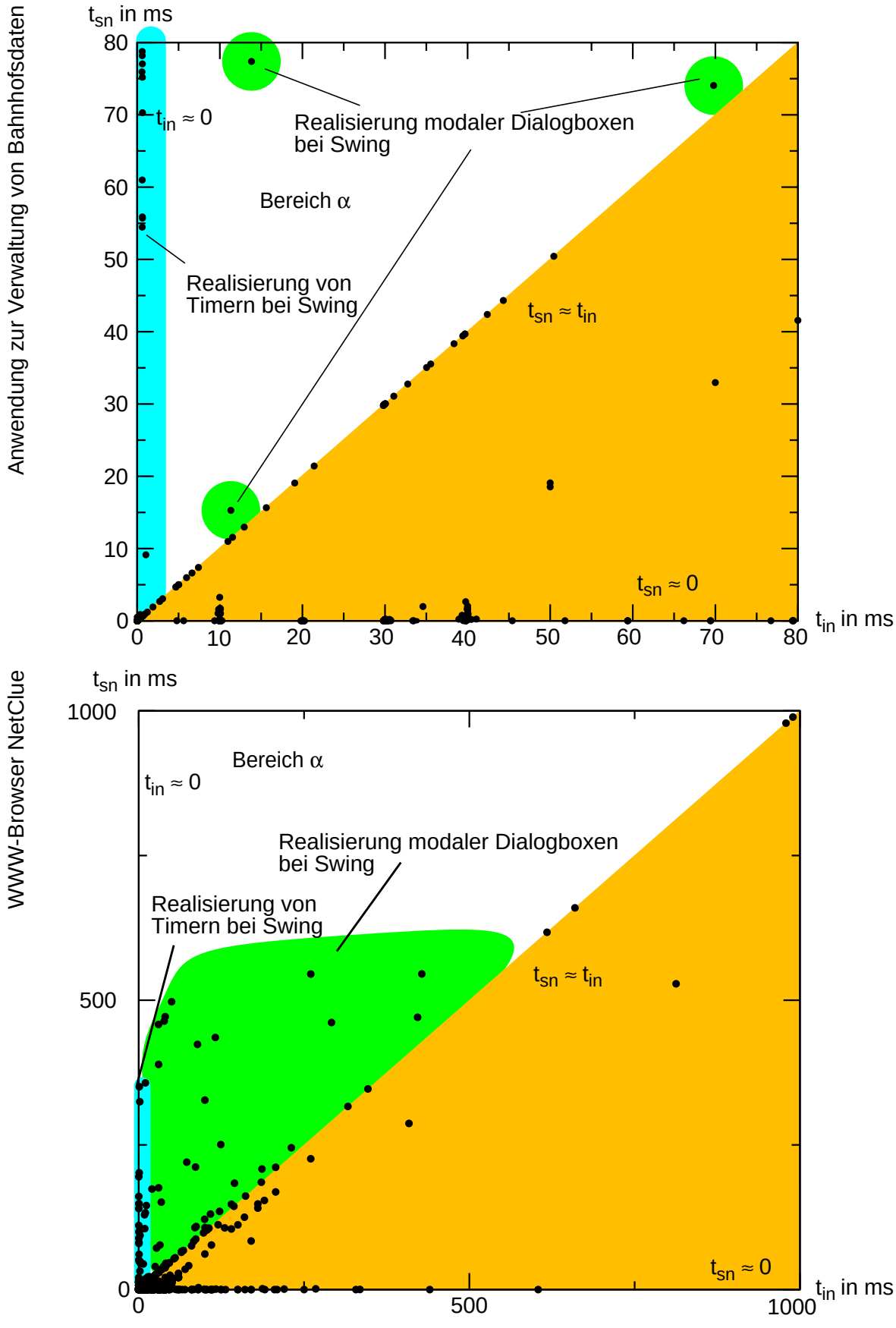


Bild 4.8: Scatter Plots für exemplarische Anwendungen

Bei einer (gedachten) Dekomposition kommen die festgestellten Nebenläufigkeiten nur innerhalb der für die GUI verantwortlichen Komponente A durch die Nutzung der Swing-Bibliothek vor. Sie haben keine Auswirkungen auf die Beauftragungen zwischen den Komponenten. Bei der Bahnhofsverwaltungsanwendung muss dies so sein, da diese mit einem Benutzeraktionsüberwacher, der maximal nur einen Kontrollfluss durch die Komponente B+C zulässt, programmiert wurde. Das Beispiel des WWW-Browsers NetClue, dessen Quell-Code nicht eingesehen und nicht verändert wurde, zeigt jedoch, dass dieses Überwacherprinzip universeller eingesetzt wird bzw. dass man es universeller einsetzen kann.

4.2.2 Synchronisation von Prozessen

Im Folgenden werden noch weitere Synchronisationsaspekte betrachtet:

- Es wird untersucht, wie sich die Konkurrenz um die Prozessor-Ressource auswirkt. Es wird geprüft, ob die bei der vorgestellten Modellierungsmethode gemachten Annahmen erlaubt sind.
- Ein weiterer Schwerpunkt bildet die Berechnung des Aufwandes, der durch die Synchronisation, die zur Erlangung der Migrationsfähigkeit einer Komponente vorgenommen werden muss, zustande kommt. Sie wird mittels Stochastischer Petri-Netze durchgeführt.

4.2.2.1 Konkurrenz und Kooperation

In dieser Arbeit werden Mechanismen berücksichtigt, die eine Kooperation zwischen Objekten innerhalb einer Komponente durch blockierende Methodenaufrufe und eine Kooperation zwischen Komponenten durch blockierende Methodenfernaufrufe realisieren. Zudem wird mit der Vorstellung eines Benutzeraktionsüberwachers ein Mechanismus berücksichtigt, der eine relevante Konkurrenz zwischen simultanen Vorgängen innerhalb einer Anwendungskomponente ausschließt. Im Allgemeinen beinhaltet eine Anwendung zu jedem Zeitpunkt maximal nur einen ablaufenden Prozess.

Durch die Modellierungsmethode wird die Konkurrenz um Prozessor-Ressourcen grundsätzlich berücksichtigt, da die Leistungsgrößenberechnungen auf den in Abschnitt 4.1.1 vorgestellten Algorithmen für GPWNs beruhen. Es wird hierbei eine vereinfachte Betrachtung der realen Java-Prozessverwaltung in Kauf genommen, um den produktformbasierten Berechnungen Processor Sharing (PS) zugrunde legen zu können. Ein JVM-Scheduler berücksichtigt jedoch Prioritäten und arbeitet zusätzlich oftmals mit einem Zeitscheibenverfahren. Es ist deshalb zu beachten, dass wenn Prozesse unterschiedliche Prioritäten besitzen oder kein sich näherungsweise wie PS verhaltendes Zeitscheibenverfahren verwendet wird, dann die durch das Modell berechneten Leistungsgrößen im Vergleich zu den realen Größen verfälscht sind.

Es wird davon ausgegangen, dass Systemdienste durch RMI-Server-Komponenten realisiert werden. Je Client-Beauftragung existiert ein Bearbeitungsprozess, der eventuell mit anderen Bearbeitungsprozessen darum konkurriert, vom Zustand `bereit` in den Zustand `laufend` zu gelangen, wobei die Prozessor-Ressource nach der PS-Strategie zugeteilt wird.

4.2.2.2 Beschränkte Stochastische Petri-Netze und ihre Verwandtschaft zu GWNs

Petri-Netze sind ein mächtiges Hilfsmittel, mit denen man Synchronisationsabläufe modellieren kann, um über sie qualitative und quantitative Aussagen zu erhalten. Da dieses Hilfsmittel im Folgenden eingesetzt wird, werden zunächst Konzepte und Methoden im Zusammenhang mit Petri-Netzen vorgestellt.

Ein Petri-Netz besteht aus N_p Plätzen (Stellen), N_t Transitionen und gerichteten Verbindungen zwischen den Plätzen und den Transitionen. Es kann als Graph mit den Plätzen und den Transitionen als Knoten und den Verbindungen als gerichtete Kanten beschrieben werden. Eine Kante ist entweder von einem als Vorplatz bezeichneten Knoten auf einen, eine Transition repräsentierenden Knoten oder von einem Transitionsknoten zu einem Knoten, der Nachplatz heisst, gerichtet. Im Gegensatz zu Warteschlangennetzen, bei denen die Knoten Bediensysteme repräsentieren, die in Warteschlangen und Bedieneinheiten verfeinert werden können, existieren bei Petri-Netzen mit den Plätzen und Transitionen zwei Typen von Knoten. Bei Warteschlangennetzen werden Anforderungen den sie bearbeitenden Knoten zugeordnet. Bei Petri-Netzen gibt es ähnliche Elemente, die Marken genannt werden. Marken werden Plätzen zugeordnet. Ein Netzzustand $\underline{S}$ ist durch eine vollständige Markierung, die die Markenanzahl je Platz angibt, eindeutig beschrieben. Die Elemente des Vektors $\underline{S}$ beschreiben die Anzahl der Marken je Platz p_i.

Wenn eine Transition t_i aktiviert ist, dann befinden sich in ihren Vorplätzen genug Marken, um einen Schaltvorgang, der auch als Feuern bezeichnet wird, durchzuführen. Beim Feuern wird an jedem Vorplatz eine durch das Kantengewicht festgelegte Anzahl von Marken entfernt und an jedem Nachplatz eine wiederum durch das Kantengewicht festgelegte Markenanzahl erzeugt. Durch das Feuern wird i. A. eine Zustandsänderung vorgenommen. Marken, die eine Eingabemenge $\underline{M}_e(i)$ bilden, werden beim Feuern einer Transition t_i in eine Ausgabemenge $\underline{M}_a(i)$ transformiert. Die Vektoren $\underline{M}_e(i)$ und $\underline{M}_a(i)$ sind Teilmarkierungen.

Ein Petri-Netz ist verklemmungsfrei, wenn jeder erreichbare Netzzustand durch das Feuern einer Transition wieder verlassen werden kann. Ein Petri-Netz ist reversibel, wenn von jedem erreichbaren Netzzustand der Initialzustand wieder erreicht werden kann. Ein Petri-Netz ist lebendig, wenn ausgehend vom Initialzustand durch eine Abfolge von Feuervorgängen erreicht werden kann, dass jede Transition mindestens einmal feuert. Ein Petri-Netz ist *n*-beschränkt, wenn jeder Platz nie mehr als *n* Marken aufnehmen muss bzw. kann.

Bei SPNs (*Stochastische Petri-Netze*) wird der Transition t_i eine zustandsabhängige Rate $\lambda_i(\underline{S})$ zugewiesen, die die Dauer festlegt, die zwischen der Aktivierung und dem Feuern vergeht. Diese Dauer kann aufgrund einer Zufallsverteilung bestimmt sein. In dieser Arbeit wird von negativ-exponentiellen Verteilungen ausgegangen, so dass den Transitionen eine mittlere Schaltrate zugeordnet werden kann. Der Stochastische Prozess entspricht dann einer Markoff-Kette [88]. Bei sogenannten Verallgemeinerten SPNs sind zwei Typen von Transitionen möglich. Neben Transitionen mit einer Schaltrate existieren weitere Transitionen, die immer unmittelbar nach ihrer Aktivierung feuern. Sie besitzen Priorität beim Feuern gegenüber Transitionen mit Schaltrate.

Unterschiede zwischen GWNs und SPNs

Unterschiede zwischen GWNs und SPNs sind:

- Bei SPNs wird durch das Feuern einer Transition t_i ein Zustandsübergang verursacht. Bei diesem Vorgang wird $\underline{M}_e(i)$ in $\underline{M}_a(i)$ transformiert. Die Größe dieser Eingabe- und Ausgabemengen kann variieren, muss jedoch mindestens eins sein. Bei GWNs werden Zustandsübergänge durch die Bewegung einzelner Anforderungen verursacht. Dies bedeutet, dass hier die Eingabe- und Ausgabemenge immer nur aus einem Element besteht.
- Bei GWNs existiert eine konstante Anzahl von Anforderungen im Netz. Bei SPNs kann die Anzahl schwanken. Vergleichbar mit GWNs sind beschränkte SPNs, bei denen die Anzahl der Marken eine obere Schranke besitzt.

Produktformlösungen

Über das durch die globalen Zustandsgleichungen gegebene LGS können wie bei GWNs die Wahrscheinlichkeiten für die Systemzustände berechnet werden. In [1, 3, 21, 49] werden Produktformlösungen für SPNs und Verallgemeinerte SPNs entwickelt, die auf der durch die Gleichung (4.22) gegebenen, zustandsabhängigen Produktformlösung basieren. In [116] wird zudem ein MVA-Algorithmus vorgestellt.

Folgende hinreichenden Bedingungen, deren Erfüllung für das Anwenden von den in [21, 116] entwickelten Produktformlösungen für SPNs notwendig ist, werden in [49] vorgestellt:

- Es dürfen keine zwei Transitionen t_i und t_j existieren, die die gleiche Eingabemenge besitzen. Es muss $\underline{M}_e(i) \neq \underline{M}_e(j), \forall i, j$ gelten, wenn $i \neq j$ ist.
- Jede Eingabemenge $\underline{M}_e(i)$ einer Transition t_i muss der Ausgabemenge $\underline{M}_a(j)$ einer Transition t_j entsprechen. Es muss für jedes i ein j gefunden werden, so dass $\underline{M}_e(i) = \underline{M}_a(j)$ gilt. Es sei angemerkt, dass eine Transition grundsätzlich in mehreren Modi arbeiten kann, so dass man ihr mehrere Ausgabemengen zuordnen kann. In dieser Arbeit werden nur Transitionen, die in einem Modus arbeiten, betrachtet. In [21] wird der Allgemeinfall behandelt.

- Die Rate $\lambda_i(\underline{S})$ muss durch folgende Funktion darstellbar sein

$$\lambda_i(\underline{S}) = \frac{\Psi(\underline{S} - \underline{M}_e(i))\mu_i}{\Phi(\underline{S})}. \tag{4.44}$$

  Der Parameter $\Phi(\underline{S})$ ist durch Gleichung (4.23) gegeben. In [48, 117] wird der Parameter $\Psi(\underline{S} - \underline{M}_e(i))$ eingeführt. Sein Wert ist in dieser Arbeit immer eins, da nur Petri-Netze mit einzeln bewegten Marken betrachtet werden.

Eine Produktformlösung für SPNs erhält man nach [21, 116], indem man zunächst die Inzidenzmatrix

$$\underline{I} = \begin{bmatrix} \underline{M}_a(1) - \underline{M}_e(1) \\ \cdots \\ \underline{M}_a(N_t) - \underline{M}_e(N_t) \end{bmatrix} \tag{4.45}$$

des Petri-Netzes erstellt. Es wird eine Funktion $\varepsilon(j)$ eingeführt, so dass für jedes $\underline{M}_e(i) = \underline{M}_a(j)$ die Funktion $\varepsilon(j)$ den Wert *i* liefert. Man definiert weiter einen Vektor

$$\underline{C} = \begin{pmatrix} \log\left(\frac{\mu_{\varepsilon(1)}}{\mu_1}\right) \\ \cdots \\ \log\left(\frac{\mu_{\varepsilon(N_t)}}{\mu_{N_t}}\right) \end{pmatrix}. \tag{4.46}$$

Schließlich erstellt man das Gleichungssystem

$$(-\underline{I}) \begin{pmatrix} \log(y_1) \\ \cdots \\ \log(y_{N_p}) \end{pmatrix} = \underline{C}. \tag{4.47}$$

Für die Zustandswahrscheinlichkeiten gilt die der Gleichung (4.22) entsprechenden Produktform

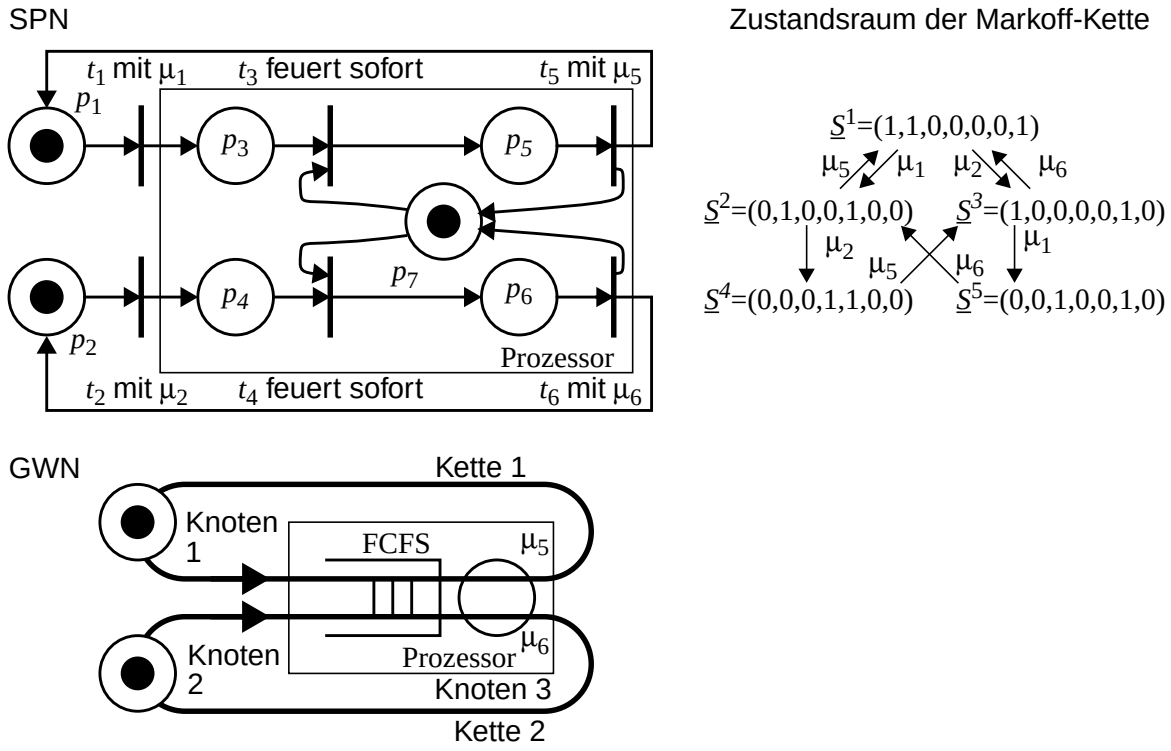
$$p(\underline{S}) = \frac{1}{G(k)} \Phi(\underline{S}) \prod_{i=1}^{N_p} y_i^{k_i}. \tag{4.48}$$

In Anhang B wird das Anwenden dieser Produktformlösung bei Stochastischen Petri-Netzen anhand eines Beispiels veranschaulicht.

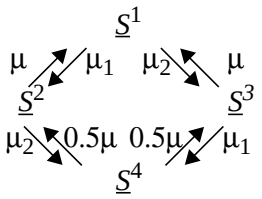
GWNs und SPNs mit gleicher Markoff-Kette

In manchen Fällen lässt sich ein Problem sowohl mit Hilfe eines GWNs als auch mit Hilfe eines (Verallgemeinerten) SPNs so modellieren, dass beide Modellierungen die gleiche Markoff-Kette beinhalten.

In Bild 4.9 ist dieses Prinzip exemplarisch veranschaulicht. In dem dort dargestellten Beispiel nutzen zwei Clients 1 und 2 wiederholt nach Abwarten einer Zeitspanne (Pause) einen Prozessor. Dieses Verhalten ist durch das sich oben auf der linken Seite befindende Petri-Netz so modelliert, dass während einer Pause des Clients *i* sich eine Marke auf dem Platz p_i befindet.



Vereinfachung des Zustandsraums aus Symmetriegründen ($\mu_5 = \mu_6 = \mu$)



$\underline{S}^1 = (1,1,0,0)$
$\underline{S}^2 = (0,1,1,0)$
$\underline{S}^3 = (1,0,0,1)$
$\underline{S}^4 = (0,0,1,1)$

Codierung für das SPN

$\underline{S}^1 = ((1,0),(0,1),(0,0))$
$\underline{S}^2 = ((0,0),(0,1),(1,0))$
$\underline{S}^3 = ((1,0),(0,0),(0,1))$
$\underline{S}^4 = ((0,0),(0,0),(1,1))$

Codierung für das GPWN

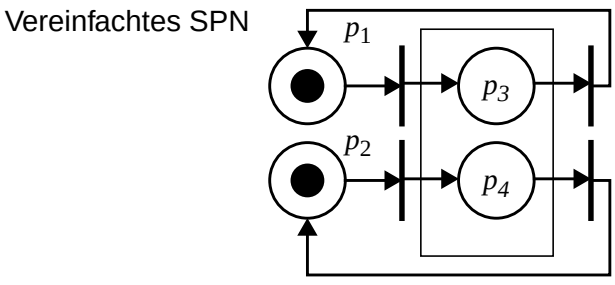


Bild 4.9: Beispiel eines GWNs und SPNs mit gleicher Markoff-Kette

Die Dauer der Pause ist durch die Schaltrate der Transition t_i festgelegt. Beide Clients konkurrieren um den Prozessor, so dass eine Synchronisation mit Hilfe des Prinzips des wechselseitigen Ausschlusses (engl. *Mutual Exclusion*) vorgenommen wird. Die Belegung des Platzes p_{i+4} mit einer Marke ist ein Zeichen, dass der Prozessor vom Client *i* belegt ist. Die Dauer der Belegung ist durch die Schaltrate der Transition t_{i+4} festgelegt. Die Belegung des Platzes p_{i+2} sagt aus, dass Client *i* auf die Nutzung des belegten Prozessors wartet. Die Transition t_{i+2} schaltet sofort nach ihrer Aktivierung. In Bild 4.9 ist oben rechts der aus 5 Zuständen bestehende Zustandsraum der dazugehörigen Markoff-Kette dargestellt.

Der wechselseitige Ausschluss kann bei zwei Clients auch durch das weiter unten dargestellte GWN modelliert werden. Der Prozessor wird hierbei als Bediensystem mit einer FCFS-Warteschlangendisziplin modelliert. Diesem GWN ist die gleiche Markoff-Kette wie dem SPN zugeordnet. Wenn die mittleren Belegungen gleich groß sind, d. h. wenn beim SPN die Schaltraten μ_{i+4} von t_{i+4} oder beim GWN die Bedienraten μ_{i+4} gleich groß (nämlich μ) sind, dann ergibt sich ein ebenfalls in Bild 4.9 dargestellter, vereinfachter Zustandsraum mit 4 Zuständen, da die Zustände $\underline{S}^4$ und $\underline{S}^5$ zusammengefasst werden können. In Bild 4.9 sind zudem die sowohl für das neu entstandene, ganz unten dargestellte SPN als auch für das neue GWN geltenden Zustands-Codierungen angegeben. Zum Berechnen der Zustandswahrscheinlichkeiten ist bei Beiden das Anwenden einer Produktformlösung erlaubt. Das GWN entspricht folglich einem GPWN.

Leistungsgrößen können nun auf unterschiedliche Arten berechnet werden:

- Für die Berechnung der Zustandswahrscheinlichkeiten für das SPN gilt

  $$\Psi(\underline{S}) = 1$$

  und

  $$\Phi(\underline{S}) = \begin{cases} 2 & \underline{S} = (0, 0, 1, 1) \\ 1 & sonst \end{cases} .$$

  Unter Anwendung des Theorems von [21] kann eine Berechnung wie folgt durchgeführt werden. Mit

  $$\underline{M}_e(1) = \underline{M}_a(3) = (1, 0, 0, 0),$$

  $$\underline{M}_e(2) = \underline{M}_a(4) = (0, 1, 0, 0),$$

  $$\underline{M}_e(3) = \underline{M}_a(1) = (0, 0, 1, 0) \text{ und}$$

  $$\underline{M}_e(4) = \underline{M}_a(2) = (0, 0, 0, 1)$$

erhält man unter Verwendung von (4.45), (4.46) und (4.47) zwei unabhängige Gleichungen:

$$\log(y_1) - \log(y_3) = \log\left(\frac{\mu}{\mu_1}\right) \text{ und}$$

$$\log(y_2) - \log(y_4) = \log\left(\frac{\mu}{\mu_2}\right).$$

Wenn für $y_1 = 1$ und für $y_2 = 1$ gewählt wird, dann ist

$$y_3 = \frac{\mu_1}{\mu} \text{ und}$$

$$y_4 = \frac{\mu_2}{\mu}$$

Wenn man schließlich die Gleichung (4.48) anwendet, erhält man

$$p(\underline{S}^1) = \frac{1}{G},$$

$$p(\underline{S}^2) = \frac{1}{G}\frac{\mu_1}{\mu},$$

$$p(\underline{S}^3) = \frac{1}{G}\frac{\mu_2}{\mu} \text{ und}$$

$$p(\underline{S}^4) = \frac{1}{G}\frac{2\mu_1\mu_2}{\mu^2}.$$

- Für die Berechnung der Zustandswahrscheinlichkeiten für das GWN gilt:

  Man kann z. B. über das BCMP-Theorem mit Hilfe von Gleichung (4.13) alle Zustandswahrscheinlichkeiten berechnen. Es gilt $N = 3$, $R = 2$ und $v_{ir} = 1, \forall i, r$. Man erhält die Zustandswahrscheinlichkeiten

  $$p(\underline{S}^1) = \frac{1}{G}\frac{1}{\mu_1\mu_2},$$

  $$p(\underline{S}^2) = \frac{1}{G}\frac{1}{\mu_2\mu},$$

  $$p(\underline{S}^3) = \frac{1}{G}\frac{1}{\mu_1\mu} \text{ und}$$

$$p(\underline{S}^4) = \frac{1}{G}\frac{2}{\mu^2}.$$

Der Vergleich der Ergebnisse zeigt nur einen Unterschied bei den Normierungskonstanten G:

$$G_{SPN} = \mu_1\mu_2 G_{GWN}.$$

Man sieht, dass über beide Wege die gleichen Zustandswahrscheinlichkeiten berechnet worden sind. Mit den Gleichungen (4.1), (4.2), (4.3) und (4.4) läßt sich dann die mittlere Aufenthaltsdauer $\bar{t}$ einer Anforderung bzw. Marke im Prozessor berechnen.

4.2.2.3 Betrachtung der Synchronisation zur Erlangung der Migrationsfähigkeit

Es wird zunächst exemplarisch ein Migrationsszenario einer Komponente, die einen Systemdienst realisiert, untersucht. Für dieses Szenario soll gelten, dass zwei Komponenten, die Teil verschiedener Anwendungen sind, eine statische Beziehung zu der Systemdienstkomponente besitzen. Es wird angenommen, dass das Migrationskonzept dem in Abschnitt 3.1.2.2 vorgestellten Vorschlag von Chen entspricht, bei dem eine Middleware-Technologie zur Verfügung steht, die eine Beauftragung während der Migration erlaubt, da Aufrufe in den Stubs zwischengespeichert werden.

Modellierung mit Hilfe eines Verallgemeinerten SPNs

Das Szenario wird durch das in Bild 4.10 dargestellte Verallgemeinerte SPN modelliert. Wenn der Platz p_1 bzw. p_8 eine Marke enthält, ist die den Systemdienst bereitstellende Komponente

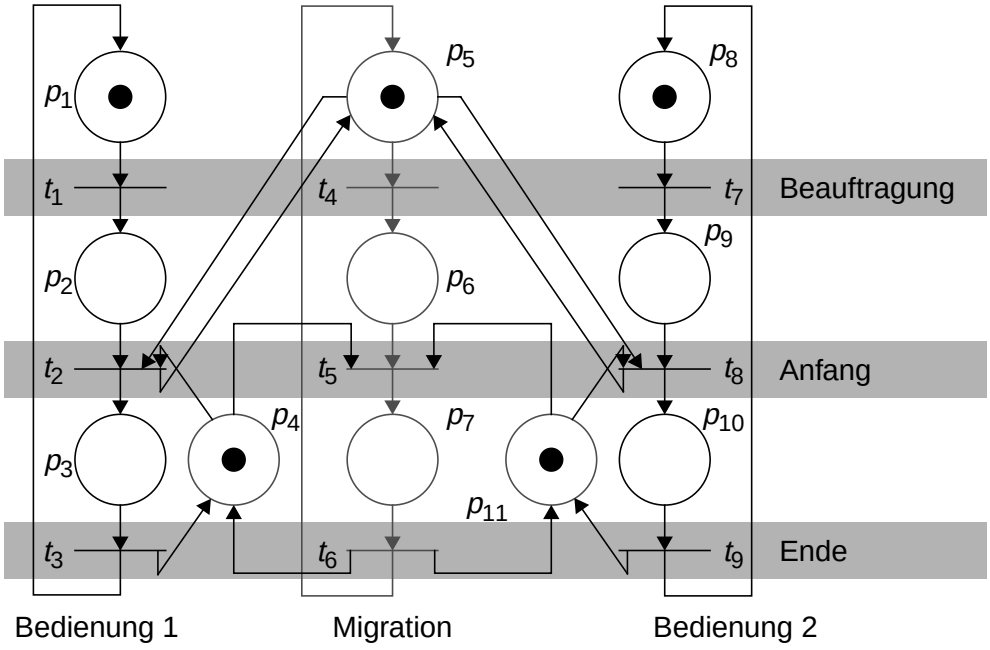


Bild 4.10: Petri-Netz, Szenario *Migration einer Komponente bei 2 Bedienungen*

nicht vom Client 1 bzw. Client 2 beauftragt. Die Transitionen t_1 und t_7 sind mit den Schaltraten λ_1 und λ_2 versehen und geben die mittlere Zeitdauer an, in dem der Dienst nicht beauftragt wird. Wenn der Platz p_2 bzw. p_9 eine Marke enthält, ist die Systemdienst-Komponente vom Client 1 bzw. Client 2 beauftragt. Erst wenn der Platz p_3 bzw. p_{10} eine Marke enthält, wird die Beauftragung bearbeitet. Es wird als Bearbeitungsdisziplin PS angenommen. Die Schaltraten der Transitionen t_3 bzw. t_9 sind deshalb markierungsabhängig. Sie sind bei Aktivierung, wenn p_3 und p_{10} besetzt sind, gleich $0.5\mu_1$ bzw. $0.5\mu_2$, ansonsten bei jeder anderen Aktivierung gleich μ_1 bzw. μ_2. Wenn p_5 eine Marke enthält, ist von einer Konfigurationsverwaltung keine Migration beauftragt. Die Schaltrate der Transition t_4 entspricht folglich der Migrationsbeauftragungsrate λ_m. Befindet sich eine Marke in p_6, dann ist eine Migration beauftragt, die vorgenommen wird, wenn p_7 eine Marke enthält. Die Schaltrate der Transition t_6 ist durch die mittlere Migrationsdauer μ_m bestimmt. Zwischen einer Migrationsbeauftragung und dem Ende der Migration kann eine Dienstnutzung beauftragt werden. Der beauftragende Client wird jedoch erst nach der Migration bedient. Eine Migration kann erst durchgeführt werden, nachdem gerade vorgenommene Bedienungen beendet worden sind. Zur Synchronisierung sind deshalb die Transitionen t_2, t_5 und t_8 eingeführt, die sofort nach ihrer Aktivierung feuern.

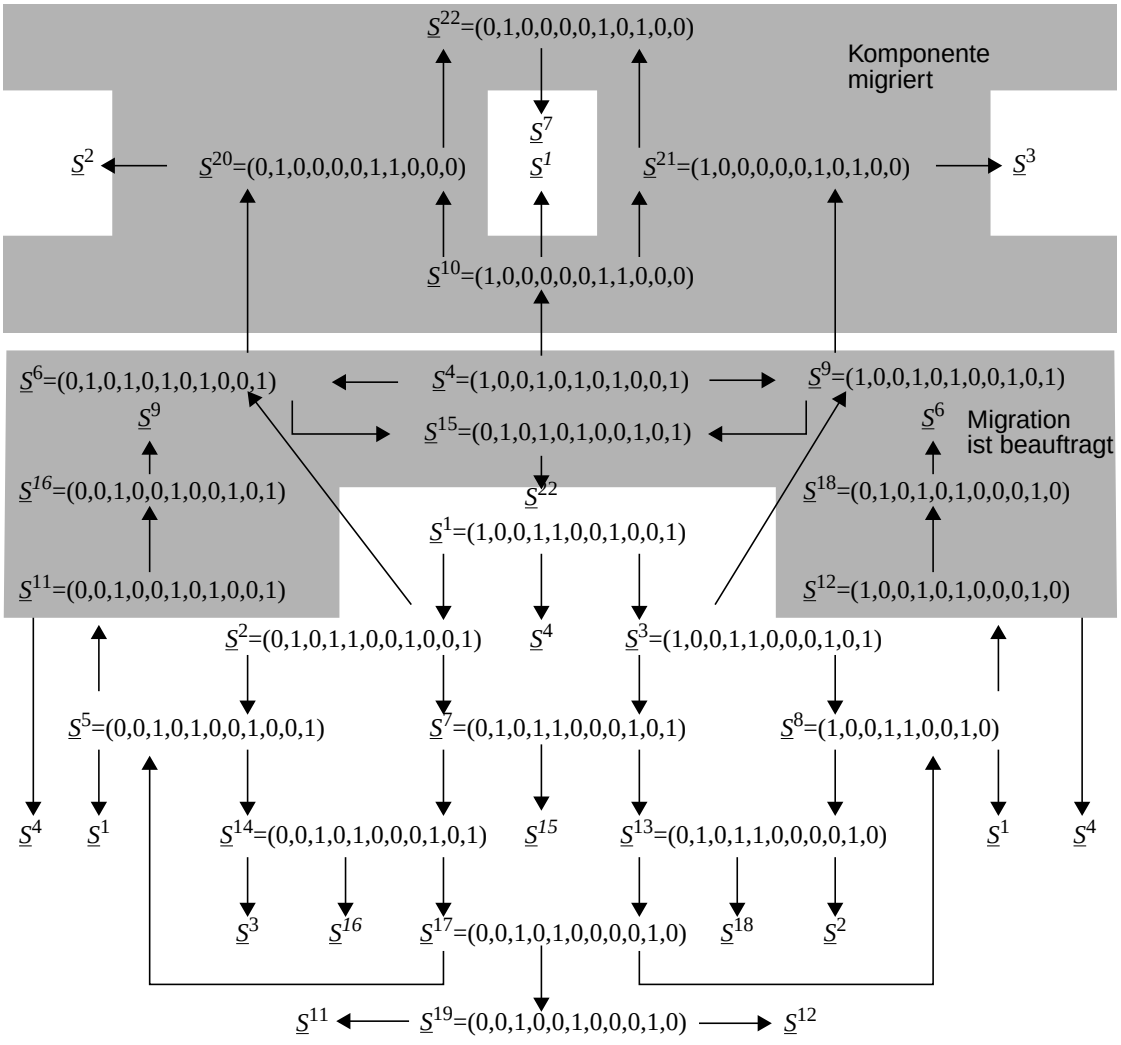


Bild 4.11: Zustandsraum, Szenario *Migration einer Systemdienstkomponente*

Berechnung von Leistungsgrößen

Das Petri-Netz aus Bild 4.10 ist verklemmungsfrei, reversibel, lebendig und 1-beschränkt. Bild 4.11 zeigt den Zustandsraum.

Da die Transitionen t_2, t_5 und t_8 nach ihrer Aktivierung sofort feuern, ist die Zustandswahrscheinlichkeit für die Zustände $\underline{S}^2$, $\underline{S}^3$, $\underline{S}^4$, $\underline{S}^6$, $\underline{S}^7$, $\underline{S}^9$, $\underline{S}^{13}$, $\underline{S}^{14}$ gleich null, da sofort ein Nachfolgezustand angenommen wird. Die Zustandswahrscheinlichkeit von Zustand $\underline{S}^{15}$ ist ebenfalls gleich null, da der Zustand nicht erreichbar ist. Man erhält den in Bild 4.12 dargestellten Zustandsraum mit 12 Zuständen.

Die Bedingung zur Berechnung der Zustandswahrscheinlichkeiten über Produktformlösungen ist nicht erfüllt. Beispielsweise kann die Eingabemenge (p_2, p_5), die die Transition t_2 zum Feuern benötigt, niemals durch eine Ausgabemenge erzeugt werden. Die Zustandswahrscheinlichkeiten müssen folglich über das LGS, das durch die globalen Gleichgewichtsgleichungen gegeben ist, berechnet werden.

Modellierung mit Hilfe eines GWNs

Das Problem, das durch das in Bild 4.10 dargestellte Verallgemeinerte SPN modelliert ist, lässt sich auch, wie Bild 4.13 auf der linken Seite zeigt, als GWN modellieren. Es ist ein GWN mit einem Bediensystem und zwei Warteschlangen. Die obere Warteschlange, in der die Client-Beauftragungen eingereiht werden, wird mit niederer Priorität abgearbeitet. Die untere Warteschlange ist nur dann belegt, wenn die Bearbeitung eines Migrationswunsches nicht unverzüglich vorgenommen werden kann, da die Bedieneinheit noch durch die Bearbeitung einer Cli-

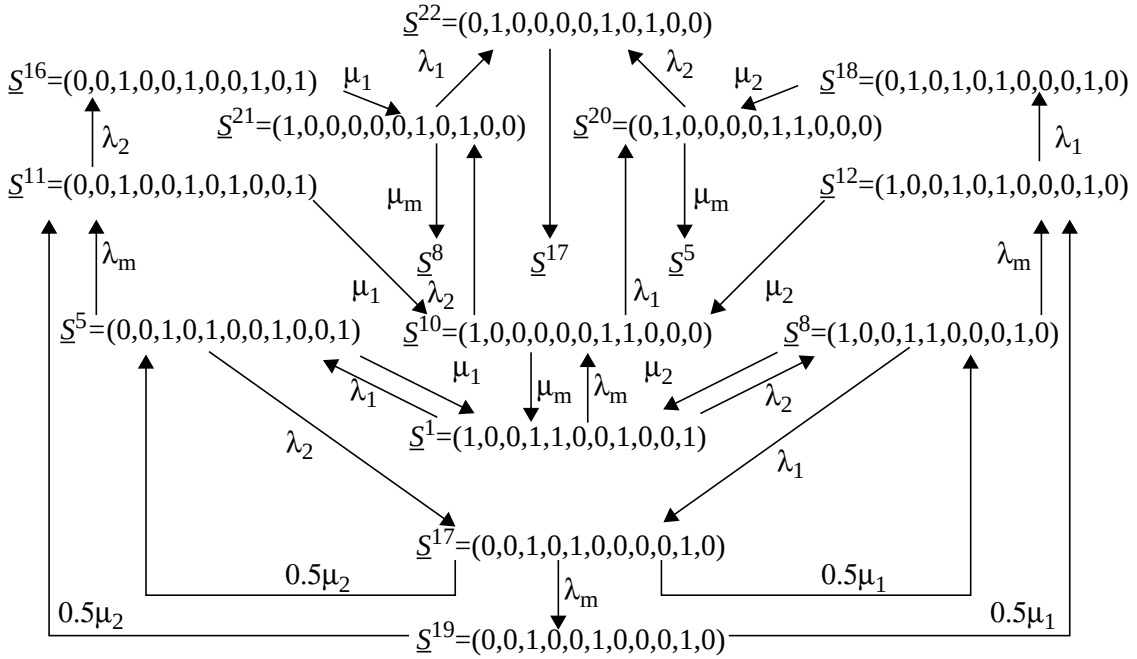


Bild 4.12: Reduzierter Zustandsraum

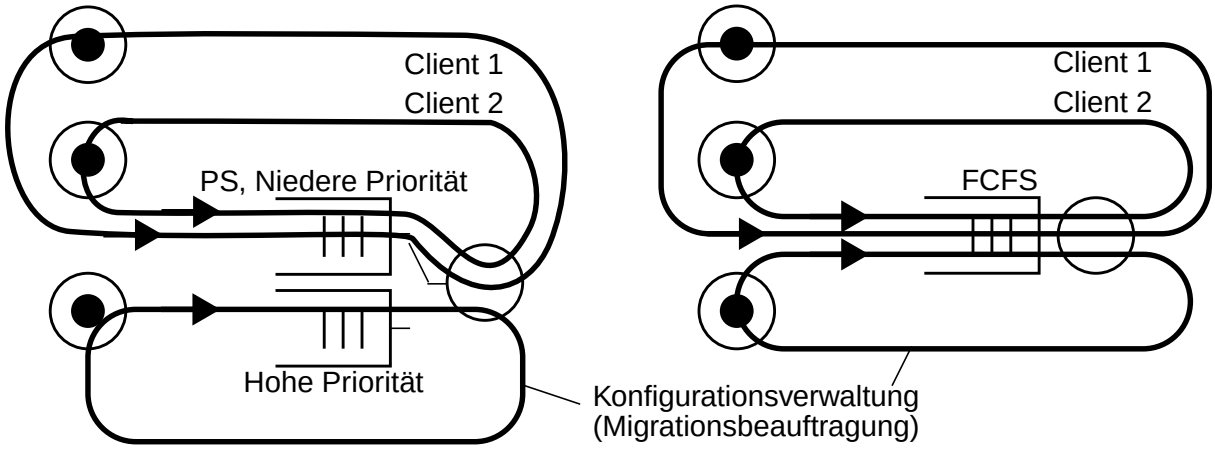


Bild 4.13: GWNs, Szenario *Migration einer Komponente bei 2 Bedienungen*

ent-Beauftragung beschäftigt ist. Zu diesem GWN gehört ebenso wie zum in Bild 4.10 dargestellten Verallgemeinerten SPN dieselbe Markoff-Kette, deren Zustandsraum in Bild 4.12 dargestellt ist.

Das Modell vereinfacht sich, wenn man einen Sonderfall betrachtet. Falls die Bedienraten μ_1, μ_2 und μ_m gleich sind, lässt sich das System als GWN, das in Bild 4.13 auf der rechten Seite dargestellt ist, modellieren. Da die Bedienraten gleich sind, lässt sich eine Produktformlösung zum Berechnen der Leistungsgrößen anwenden. Eine Produktformlösung ist zudem allgemein für *n* Clients bestimmbar.

Dass die Vereinfachung des Modells bei gleichen Bedienraten gültig ist, sieht man bei Betrachtung einiger Ankunftreihenfolgen und Bearbeitungsreihenfolgen von Anforderungen:

- Wenn die Ankunftsreihenfolge *Client 1 bzw. Client 2 – Migration – Client 2 bzw. Client 1* ist, dann wird genau in dieser Reihenfolge bedient. Dies bedeutet, dass nach dem FCFS-Prinzip bedient wird.
- Wenn die Ankunftsreihenfolge *Client 1 bzw. Client 2 – Client 2 bzw. Client 1 – Migration* ist, dann muss die Migration warten, bis die beiden Clients fertigbedient sind, da, auch wenn die Client-Warteschlange niedere Priorität besitzt, durch die PS-Strategie jeder Client simultan bedient wird. Dies bedeutet, dass das Verhalten auch hier durch das FCFS-Prinzip modelliert werden kann, da die PS-Strategie bei gleichen, negativ-exponentiell verteilten Bedienzeiten nach außen die gleichen Auswirkungen wie die FCFS-Strategie zeigt.

Auch die Abarbeitung jeder anderen Ankunftsreihenfolge wirkt sich bezüglich der Zustandswahrscheinlichkeiten und den daraus berechenbaren Leistungsgrößen wie eine FCFS-Abarbeitung aus.

Kapitel 5

Verfahren zur lastbezogenen Rekonfigurierung

In diesem Kapitel werden Verfahren zur lastbezogenen Rekonfigurierung vorgestellt und bewertet. Durch das Umordnen ortsungebundener Komponenten von benutzergesteuerten Anwendungen auf weniger belastete Ablaufumgebungen können hierbei die Antwortzeiten, die die Blockierungszeiten der Benutzer bestimmen und die sich aus den Summen der einzelnen Bearbeitungszeiten der beauftragten Komponenten zusammensetzen, reduziert werden. Die Verfahren sind innerhalb einer zur Systemlaufzeit arbeitenden Konfigurationsverwaltung einsetzbar.

Es werden zunächst allgemeine Kriterien betrachtet, mit denen anschließend relevante Lastverteilungsverfahren eingeordnet und bewertet werden.

5.1 Einführung in Lastverteilungsverfahren

In diesem Abschnitt werden zunächst Kriterien zur Klassifikation und Bewertung von Lastverteilungsverfahren vorgestellt. Anschließend werden mögliche Strategien betrachtet, die von zur Systemlaufzeit arbeitenden Lastverteilungsverfahren angewendet werden.

5.1.1 Lastverteilungsverfahren – Begriffsdefinition und Klassifikation

Nach der Erläuterung von wesentlichen Begriffen werden Kriterien zur Klassifikation und Bewertung von Lastverteilungsverfahren näher vorgestellt.

5.1.1.1 Last, lastverursachende Instanzen und lastaufnehmende Instanzen

Der Anteil der belegten Gesamtkapazität der Ressourcen eines Systems wird als Last des Systems bezeichnet. Sie wird durch eine lastverursachende Instanz generiert und durch eine lastaufnehmende Instanz in dem Sinne aufgenommen, dass deren Ressourcen belegt werden.

Unter einer lastverursachenden Instanz kann je nach Granularität der Betrachtungsweise Folgendes verstanden werden:

- Eine Komponente besitzt einen Daten- und Aktivitätszustand, deren Speicherung die Speicher-Ressourcen belastet. Sie kann bei prozessorientierten Systemen mehrere Prozesse beinhalten. Der Ablauf ihrer Prozesse belastet u. a. Prozessor- und Speicher-Ressourcen.
- Ein Objekt, das einen Datenzustand besitzt und somit Speicher-Ressourcen belegt, kann durch Methodenaufrufe bzw. allgemein durch das Auslösen von Transaktionen von anderen Objekten beauftragt werden. Durch die Abläufe bei der Abarbeitung von Beauftragungen, die bei prozessorientierten Systemen durch Prozesse realisiert werden können, werden Speicher- und Prozessor-Ressourcen beansprucht. Bei Servern – darunter werden je nach Granularität der Betrachtung Objekte oder Server-Komponenten verstanden – wird eine Belastung durch eine externe Beauftragung ausgelöst. Die Last, die durch solche Instanzen erzeugt wird, hängt folglich inhärent von der Anzahl der externen Beauftragungen ab.
- Einem einzelnen Prozess wird ein Speicherbereich zugeordnet. Neben Prozessor-Ressourcen müssen deshalb auch Speicher-Ressourcen betrachtet werden. Erst wenn man sich auf die eigentlichen Abläufe bezieht, die oftmals auch als Aktivitäten bezeichnet werden, wird nur noch die Belastung der Prozessor-Ressource betrachtet. Nach Erzeugung eines Prozesses wird in der Regel sofort die Prozessor-Ressource beansprucht. Diese aktive Inanspruchnahme kann durch Blockierungen unterbrochen werden und ist im Normalfall von endlicher Dauer.

Eine lastaufnehmende Instanz ist je nach Granularität der Betrachtungsweise in dieser Arbeit eine Komponentenablaufumgebung, ein Rechner oder ein Prozessor.

5.1.1.2 Lastverteilung

Durch den Einsatz eines Lastverteilungsverfahrens wird das Erreichen eines Ziels angestrebt, indem Last zu lastaufnehmenden Instanzen zugeordnet wird. Ein Ziel kann z. B. sein, Antwortzeiten zu minimieren, den Durchsatz zu maximieren und gegebenenfalls bei diesen Optimierungen auch Fairness-Aspekte zu berücksichtigen. Weitere Ziele können in der Gewährleistung von Echtzeitfähigkeit [106] oder in der Verbesserung der Zuverlässigkeit oder der Verfügbarkeit liegen. Die Leistung eines Systems wird verbessert, wenn durch den Einsatz des Verfahrens dem definierten Ziel näher gekommen wird.

Die Zuordnung von Last zu lastaufnehmenden Instanzen kann auch indirekt in dem Sinne geschehen, dass nicht die schon verursachte Last selbst sondern die lastverursachende Instanz einer lastaufnehmenden Instanz zugewiesen wird. Beispielsweise kann nach einer in Abschnitt 3.1.2.1 beschriebenen Starken oder Schwachen Migration eine Komponente einer neuen Ablaufumgebung zugeordnet werden. Die zukünftig von dieser Komponente verursachte Last

wird dann von dieser neuen Ablaufumgebung aufgenommen. Im Folgenden kann unter dem Begriff Last sowohl schon verursachte Last als auch lastverursachende Instanzen verstanden werden.

5.1.1.3 Klassifikation von Lastverteilungsverfahren

In [14] werden einige Klassifikationskriterien für Lastverteilungsverfahren vorgestellt, deren Zusammenhang in Bild 5.1 dargestellt ist.

Lastverteilungsverfahren unterscheiden sich grundlegend darin, ob durch sie Entscheidungen statisch vor dem Systembetrieb oder dynamisch während diesem getroffen werden. Der Vorteil statischer Verfahren gegenüber dynamischen Verfahren ist, dass die erlaubte Rechenzeit und der verfügbare Speicherplatz nicht durch den Betrieb eingeschränkt ist. So werden bei statischen Verfahren in der Regel aufwändige Berechnungen vorgenommen, um dem Optimum möglichst nahe zu kommen. Beispiele für statische Verfahren werden in [83, 125] vorgestellt. Dynamische Verfahren besitzen bezüglich ihrer erlaubten Rechenzeit und ihres verfügbaren Speicherplatzes vom Systembetrieb abhängige Grenzen. Deswegen werden bei den meisten dynamischen Verfahren schnelle und speicherplatzsparende Berechnungen (z. B. durch Heuristiken) durchgeführt, die unter Rücksichtnahme dieser Beschränkungen eine Lösung nahe dem Optimum berechnen sollen. Der Vorteil dynamischer gegenüber statischer Verfahren ist, dass auf die im Betrieb vorkommende Dynamik reagiert werden kann. Die Informationen, aufgrund denen Entscheidungen getroffen werden, sind i. A. aktueller als bei statischen Verfahren. Dynamische Verfahren sind adaptiv, wenn die verwendeten Algorithmen oder die verwendeten Parameter aufgrund des Systemverhaltens änderbar sind. Nicht adaptive Verfahren werden auch als starre Verfahren bezeichnet.

Es gibt dynamische Verfahren, bei denen Last nur einmal einer lastaufnehmenden Instanz zugeordnet wird. Desweiteren existieren dynamische Verfahren, bei denen schon zugeordnete Last wieder umgeordnet werden kann. Unter einem Transfer versteht man das Entfernen einer Last von einer lastaufnehmenden Instanz und ihr Hinzufügen zu einer anderen lastaufnehmen-

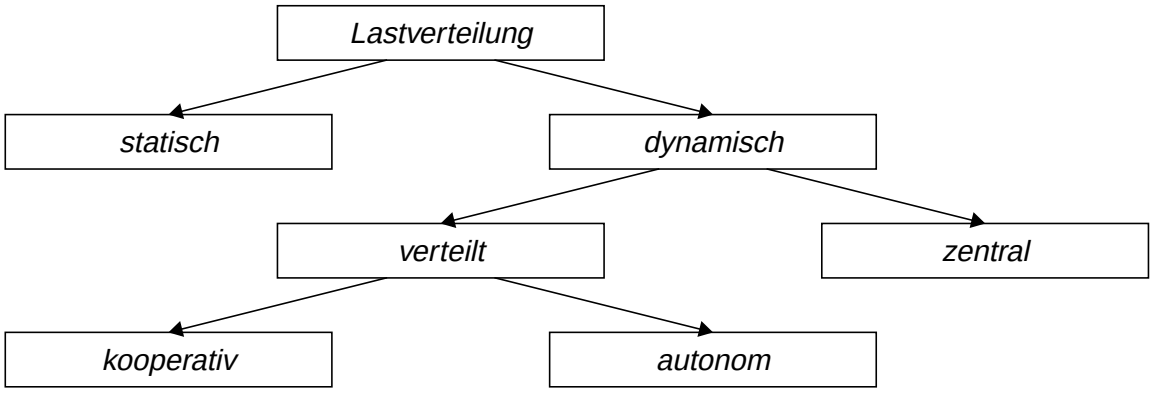


Bild 5.1: Klassifikation von Lastverteilungsverfahren

den Instanz zur Systemlaufzeit. Ein Transfer von Komponenten und Objekten wird durch Starke oder Schwache Migration realisiert. Ein Beispiel für Lastverteilung durch Transferieren von Objekten ist in [95] gegeben. Ein Transfer von Prozessen wird durch Prozessmigration, d. h. Starke Migration, ermöglicht.

Dynamische Verfahren unterscheiden sich darin, dass sie verteilt oder zentral ablaufen können. Es gibt verteilt ablaufende Verfahren, bei denen Instanzen Entscheidungen dezentral treffen. Hiervon unterscheiden sich verteilt ablaufende Verfahren, bei denen verteilte Instanzen zwar zur Entscheidungsfindung beitragen, aber nur eine zentrale Instanz berechtigt ist, Entscheidungen vorzunehmen. Bei dynamischen Verfahren sind Strukturen mit komplexen Kooperationshierarchien möglich. Ein Extremfall ist, dass Instanzen eines verteilt arbeitenden Lastverteilungsverfahrens autonom Entscheidungen treffen.

5.1.2 Strategien bei dynamischen Lastverteilungsverfahren

In diesem Abschnitt werden Strategien von dynamischen Verfahren vorgestellt. Die Betrachtungen orientieren sich an [12, 14]. Strategien legen einen Verhaltensplan fest, nachdem Entscheidungen getroffen werden. Mechanismen legen die Art und Weise der Umsetzung der Strategien fest.

5.1.2.1 Strategien zur Auswahl von Last und lastaufnehmenden Instanzen

Strategien zur Auswahl von Instanzen behandeln die Frage, welche lastaufnehmende Instanzen von welcher Last befreit bzw. welche lastaufnehmende Instanzen damit belastet werden sollen.

Bei der Zuordnung von Last zu lastaufnehmenden Instanzen existieren Strategien, bei denen ein gemessener Lastzustand berücksichtigt wird. Oftmals werden Schwellenwerte (engl. *Threshold*) z. B. für Prozessorwarteschlangenlängen berücksichtigt, deren Überschreiten oder Unterschreiten festlegen, ob die lastaufnehmende Instanz be- oder entlastet werden soll. Desweiteren ist eine weit verbereitete Strategie, nur die Instanz mit der kürzesteten Warteschlange (engl. *Shortest Queue*) zu belasten. In [23] werden Schwellenwertverfahren und Verfahren, die eine Optimierung aufgrund der kürzesten Warteschlange vornehmen, miteinander verglichen. Zudem existieren Strategien, die sich an den über den letzten Bearbeitungszeitraum gemittelten Auslastungsgrad der lastaufnehmenden Instanzen orientieren.

Es existieren weitere Strategien, bei denen Entscheidungen nicht aufgrund eines gemessenen Lastzustandes getroffen werden und die deshalb einfacher zu realisieren sind. Bei der Lastzuordnung können z. B. die lastaufnehmenden Instanzen zyklisch abgewechselt (engl. *Round Robin*, RR) werden oder sie können per Zufall (engl. *Random*) bestimmt werden.

Wenn das Lastverteilungsverfahren verteilt realisiert ist und ein Umordnen von Last möglich ist, dann wird oftmals eine Strategie angewendet, bei der die Initiative von einer überlasteten

Instanz (senderbasiertes Verfahren), die Last abgeben will, oder von einer unterlasteten Instanz (empfängerbasiertes Verfahren), die Last aufnehmen will, ausgeht. In [119] werden exemplarische Verfahren vorgestellt, die auf solchen Strategien beruhen.

Eine Lastteilungsstrategie (engl. *Load Sharing*) ist eine Lastverteilungsstrategie, bei der eine Belegung aller lastaufnehmenden Instanzen in einem Verteilten System angestrebt wird. Sie unterscheidet sich von einer Lastausgleichsstrategie (engl. *Load Balancing*), bei der versucht wird, alle lastaufnehmenden Instanzen gleichmäßig auszulasten.

5.1.2.2 Strategien zur Verarbeitung von Informationen

Strategien zur Verarbeitung von Informationen legen fest, aufgrund welcher Informationen Entscheidungen getroffen werden und wann bzw. wo diese Informationen gesammelt bzw. gehalten werden. Es können unterschiedliche Arten von Informationen gewonnen werden. Beispielsweise können solche Informationen Messergebnisse in Form von Warteschlangenlängen, Bedienzeiten oder Antwortzeiten sein. Diese Messergebnisse können als Momentanwerte gehalten werden, oder sie können z. B. mit Hilfe eines EMA-Algorithmus (*Exponential Moving Average*) aufbereitet werden.

Die Strategien legen zudem fest, in welchen Abständen neue Informationen gewonnen werden müssen bzw. wann Informationen als veraltet angesehen werden. Die Zeitpunkte für die Gewinnung neuer Informationen können z. B. periodisch oder aufgrund von Zustandsänderungen bestimmt werden. Desweiteren ist insbesondere bei sender- und empfängerbasierten Verfahren möglich, dass eine initiierende Instanz andere Instanzen nach Informationen befragt, und dass erst aufgrund dieser Anfragen Informationen gesammelt werden.

Informationen können an zentraler Stelle oder verteilt gehalten werden. Es sind verschiedene Strategien bekannt, wie Informationen zwischen verteilten Instanzen ausgetauscht werden. Periodisches Abfragen (engl. *Polling*) von verteilten Instanzen durch eine zentrale Instanz und Rundsenden (engl. *Broadcasting*) sind weit verbreitete Formen.

5.1.3 Tauglichkeitskriterien für Lastverteilungsverfahren

Bei sogenannten dynamischen Systemen spielen Kriterien wie Stabilität und Effektivität eine entscheidende Rolle. Wenn innerhalb eines dynamischen Systems durch den Einsatz eines Lastverteilungsverfahren ein solches Kriterium verletzt wird, dann ist es hierfür untauglich.

5.1.3.1 Stabilität

Ein System ist stabil, wenn eine begrenzte Einwirkung eine begrenzte Auswirkung zur Folge hat. Jede Einwirkung verursacht eine Veränderung des Systemzustandes. Beispielsweise kann man eine Anforderungsankunft als Einwirkung auf ein Bediensystem und die Aufenthalts-

dauer einer Anforderung als Auswirkung ansehen. Wenn der mittlere Ankunftsabstand nicht größer als die mittlere Bedienzeit ist, dann kann keine endliche, mittlere Aufenthaltsdauer für die Anforderungen bestimmt werden. Das System ist instabil, da die Auswirkung nicht begrenzt ist.

Ein weiteres, oft genanntes Beispiel für Instabilität ist das Prozessorflattern (engl. *Processor Thrashing*). Bei diesem Phänomen wird eine Anforderung zwischen Bediensystemen umhergereicht, da jedes Bediensystem überlastet ist. Das System ist instabil, da die Anforderung nie bedient wird. Prozessorflattern zeichnet sich dadurch aus, dass der Überlastzustand niemals verlassen wird, da freie Kapazität nicht mehr für das Bearbeiten von Anforderungen, sondern für deren Umherreichen verbraucht wird. In [13, 124] findet man eine detaillierte Betrachtung von Stabilität in Verteilten Systemen mit dynamischen Lastverteilungssverfahren.

5.1.3.2 Effektivität

Eine grundlegende Bedingung für die Tauglichkeit eines Verfahrens ist die Einhaltung des Stabilitätskriteriums. Dies reicht jedoch nicht aus. Das Verfahren muss zudem effektiv arbeiten, d. h. sein Einsatz muss die Leistung des Systems verbessern [119].

Gründe, warum ein Lastverteilungsverfahren ineffektiv arbeiten kann, sind:

- Rekonfigurierungskosten

  Eine Rekonfigurierung zum Zwecke der Lastverteilung wird durch Transfers von Last ermöglicht. Es treten zum einen Kosten auf, da Transfers das System zusätzlich belasten. So muss z. B. bei Schwacher Migration von Komponenten deren Code und Datenzustand zwischen zwei Ablaufumgebungen unter Inanspruchnahme von Prozessor- und Speicher-Ressourcen ausgetauscht werden. Zum anderen muss beachtet werden, dass erst nach einem Lasttransfer wieder Last erzeugt und bearbeitet werden kann. In Abschnitt 4.2.2.3 ist dargestellt, dass aus diesem Grund bei der Migration einer Komponente die Antwortzeit erhöht wird.

- Informationsgewinnungskosten

  Ein Verfahren benötigt Informationen, aufgrund dessen es Entscheidungen vornehmen kann. Die Gewinnung, Verarbeitung und Speicherung von Informationen erfordert zusätzliche Rechenzeit und Speicherplatz. Ein Verfahren arbeitet ineffektiv, wenn durch den Aufwand der Informationsgewinnung sich die Leistung des Systems verschlechtert.

- Informationsalterung

  Gewonnene Informationen können in dem Sinne veraltet sein, dass die wegen ihnen getroffenen Entscheidungen dafür sorgen, dass die Systemleistung verschlechtert wird.

5.1.3.3 Weitere Kriterien – Skalierbarkeit, Robustheit und Konvergenz

Ein Lastverteilungsverfahren ist skalierbar, wenn seine Effektivität nicht von der Größe des Verteilten Systems, d. h. von der Anzahl der lastverursachenden und lastaufnehmenden Instanzen, abhängt.

Ein Verfahren gilt als robust, wenn es auftretende Fehler korrekt behandelt. Hierbei kann auch ein quantitativer Aspekt gemeint sein. So kann z. B. gefordert sein, dass bei einem gegebenen Fehler trotzdem noch eine definierte Leistung erbracht werden muss. So ist ein Verfahren als nicht robust anzusehen, wenn es bei einem derartigen Fehler (z. B. Rechnerausfall) ineffektiv arbeitet oder gar instabil wird.

Ein Verfahren ist konvergent, wenn bei einer gegebenen Störgröße sich ein Systemgleichgewicht einstellt. Unter Systemgleichgewicht wird verstanden, dass die Zuordnungen von Last zu lastaufnehmenden Instanzen unverändert bleibt. Ein interessanter Parameter ist hierbei die Zeitdauer, bis sich das Systemgleichgewicht einstellt. Konvergenz ist eigentlich keine Tauglichkeitsanforderung. So ist prinzipiell möglich, dass ein effektives und stabiles Verfahren nicht konvergent ist. Nicht-Konvergenz ist jedoch durchaus als Hinweis zu werten, dass ein Verfahren nicht sonderlich effektiv arbeitet.

5.2 Vorstellung von Lastverteilungsverfahren für die Rekonfigurierung

In diesem Abschnitt werden relevante Verfahren für den Einsatz in komponentenbasierten Verteilten Systemen mit benutzergesteuerten Anwendungen vorgestellt. Durch die Verfahren soll das Problem, die Summe der Antwortzeiten aller Benutzeraktionen zu minimieren, so gelöst werden, dass die Lösungen möglichst nahe dem Optimum sind. Zusätzlich soll die Berücksichtigung von Fairnesskriterien möglich sein. In der vorliegenden Arbeit werden zum einen Standardverfahren berücksichtigt, die ihre Entscheidungen nicht inhärent aufgrund der in Kapitel 4 entwickelten Modellierungsmethode treffen. Zum anderen werden MVA-basierte Verfahren betrachtet, die ihren Berechnungen die Beschreibung einer Anwendung durch eine Kette innerhalb eines GPWNs zugrunde legen.

5.2.1 Einsatz von Standardverfahren

Es werden folgende Standardverfahren berücksichtigt:

Round Robin-Verfahren

Bei den Zuordnungen von lastverursachenden Instanzen werden beim Round Robin-Verfahren die lastaufnehmenden Instanzen zyklisch abgewechselt. Beim sogenannten komponentenbezo-

genen Round Robin-Verfahren werden Komponenten als lastverursachende Einheiten angesehen, die den Ablaufumgebungen als lastaufnehmende Instanzen zugeordnet werden. Beim sogenannten anwendungsbezogenen Round Robin-Verfahren werden komplette Anwendungen als lastverursachende Instanzen angesehen und deshalb immer alle Komponenten einer Anwendung derselben Ablaufumgebung zugeordnet. Ein Round Robin-Verfahren arbeitet immer bei der Aufnahme neuer Anwendungen bzw. Komponenten ins System. Zudem kann z. B. durch die Veränderung der Kapazität einer Ablaufumgebung die komplette Neuanordnung der lastverursachenden Instanzen mit Hilfe eines Round Robin-Verfahrens angestoßen werden. Wenn die Ablaufumgebungen im Verteilten System unterschiedliche Kapazitäten besitzen, ist der Einsatz eines gewichteten Round Robin-Verfahrens sinnvoll.

Random-Verfahren

Bei der Zuordnung von neu ins System kommende Anwendungen werden die Ablaufumgebungen per Zufall gewählt. Als lastverursachende Instanz wird hier immer eine komplette Anwendung angesehen, d. h. es werden alle Komponenten einer Anwendung derselben Ablaufumgebung zugeordnet.

Shortest Queue-Verfahren

Alle Komponenten einer neu ins System kommenden Anwendung werden der Ablaufumgebung zugeordnet, die im Moment der Zuordnung die kürzeste Prozessorwarteschlangenlänge aufweist.

Auslastungsbasiertes Verfahren

Alle Komponenten einer neu ins System kommenden Anwendung werden der Ablaufumgebung mit der geringsten Prozessorauslastung zugeordnet. Die Auslastung eines Prozessors kann über ein EMA-Verfahren bestimmt werden.

Es gilt gemäß den Gleichungen (4.3), (4.32) und (4.34) für die Auslastung ρ_i im Knoten *i* bei *R* Anwendungen die Beziehung

$$\rho_i = \sum_{r=1}^{R} \Delta(i,r)\rho_{ir} = \sum_{r=1}^{R} \Delta(i,r)\frac{\lambda_{ir}}{\mu'_{ir}} = \sum_{r=1}^{R} \frac{\Delta(i,r)}{(t_{ir} + \tau_r)\mu'_{ir}}. \tag{5.1}$$

Die Funktion $\Delta(i, r)$ ergibt hierbei nur dann den Wert eins, wenn die Anwendung *r* der Ablaufumgebung *i* zugeordnet ist. Ansonsten ergibt sie null. Der Parameter τ_r entspricht der mittleren Dauer der Benutzerdenkpause für die Anwendung *r*. Da alle Komponenten einer Anwendung *r* auf einer Ablaufumgebung *i* lokalisiert sind, können sie wie eine betrachtet werden. Die Bedienzeit $1/\mu'_{ir}$ für diese Aggregationskomponente der Anwendung *r* berechnet sich mit

$$\frac{1}{\mu'_{ir}} = \sum_{s = 1, \zeta(s) = r}^{Z} \frac{1}{\mu_{is}} \quad . \tag{5.2}$$

5.2.2 Einsatz von MVA-basierten Lastverteilungsverfahren

Im Folgenden werden Verfahren vorgestellt, die Berechnungen mit Hilfe der Mittelwertanalyse vornehmen. Diesen Berechnungen wird hierbei die in Kapitel 4 entwickelte Modellierungsmethode zugrundegelegt. Zunächst wird auf die Strategien zur Auswahl von Instanzen und zur Verarbeitung von Informationen eingegangen. Es wird danach genauer dargestellt, wie die durch Messungen gewonnenen Informationen an eine zur Systemlaufzeit arbeitende, zentrale Konfigurationsverwaltung weitergereicht werden, die hierdurch ein aktuelles Konfigurationsabbild gewinnt.

Die MVA-basierten Verfahren sind dynamisch, da sie zur lastbezogenen Rekonfigurierung innerhalb einer Konfigurationsverwaltung zur Systemlaufzeit ausgeführt werden. Informationen werden zwar verteilt gesammelt, jedoch dann einer zentralen Stelle zugeführt, von der aus alle Entscheidungen getroffen werden. Sie können deshalb in Kombination mit der von Chen vorgestellten Migrationsverwaltung (siehe Abschnitt 3.1.2.2) eingesetzt werden. Bei dieser Migrationsverwaltung werden Informationen über alle getätigten Beauftragungen ebenfalls an zentraler Stelle gehalten, um Verklemmungssituationen zu erkennen. Die Lastverteilungsverfahren sind nicht adaptiv, da immer derselbe Rekonfigurierungsalgorithmus unabhängig vom Systemzustand angewendet wird.

Die Verfahren berücksichtigen nicht die Auslastung der Bussysteme. Es wird davon ausgegangen, dass zwar die Komponentenablaufumgebungen überlastet sein können, die Übertragungskapazitäten der Bussysteme jedoch nie ausgeschöpft werden. Diese Annahme wird darin begründet, dass die relevanten Bussysteme zukünftig eine solch große Übertragungskapazität besitzen, dass Verzögerungsschwankungen von Antwortzeiten aufgrund von Änderungen der Bussystemlast vernachlässigbar sind. Beispielsweise wird ein IEEE 1394-Bussystem in Zukunft eine Übertragungskapazität von mehr als einem Gbit/s besitzen. Desweiteren wird angenommen, dass bei einem Datenaustausch die in Abschnitt 3.2.1 beschriebenen Serialisierungs- und Deserialisierungszeiten gegenüber den Übertragungszeiten dominieren. Für die Bestimmung der Verzögerungszeiten ist hierdurch die Entfernung der am Datenaustausch beteiligten Instanzen nicht relevant, da sie maßgeblich durch die entfernungsunabhängigen Serialisierungs- und Deserialisierungszeiten bestimmt werden.

5.2.2.1 Strategien der MVA-basierten Verfahren

Den MVA-basierten Verfahren lassen sich folgende Strategien zuordnen:

Strategien zur Verarbeitung von Informationen

Der Konfigurationsverwaltung muss gemäß Abschnitt 3.1.1.3 die statischen Abhängigkeiten jeder Komponente und somit die schnittstellenstrukturelle Konfiguration des Verteilten Systems bekannt sein. In Abschnitt 4.1.2 wird erläutert, wie dementsprechend ein Aktivitätsgraph je Anwendungskomposition aufgebaut wird.

Zur Systemlaufzeit werden Messungen durchgeführt, um die Stärke der Belastung einer diensterbringenden Komponente durch eine dienstnutzende Komponente zu ermitteln. Relevante Messgrößen sind die mittleren Bedienraten und die mittleren Ankunftsraten von Beauftragungen. Die Mittelwerte werden durch einen EMA-Algorithmus innerhalb der Prozessverwaltungen der Ablaufumgebungen näherungsweise berechnet und zur zentralen Konfigurationsverwaltung gesendet, wo sie zentral in Form von Aktivitätsgraphen gehalten werden. Es werden nur Informationen gesendet, wenn ein Schwellenwert über- bzw. unterschritten wird. Innerhalb eines gegebenen Zeitintervalls können Informationen nur einmal versendet werden. Zudem werden Änderungen über die Aktivität einer Anwendung an die zentrale Konfigurationsverwaltung gesendet.

Strategien zur Auswahl von Instanzen

Es werden in der vorliegenden Arbeit zwei MVA-basierte Verfahren mit unterschiedlichen Strategien zur Auswahl von Instanzen vorgestellt:

- Zunächst wird ein Verfahren berücksichtigt, bei dem die zu transferierenden Instanzen mit Hilfe des heuristisch arbeitenden, MVA-basierten Core-Algorithmus (siehe Abschnitt 4.1.1.7) ausgewählt werden.
- Desweiteren wird ein Verfahren vorgestellt, bei dem die Instanzen mit Hilfe von MVA (siehe Abschnitt 4.1.1.6) ausgewählt werden, d. h. dass für die gegebenen Mittelwerte exakte Leistungsgrößen berechnet werden.

Jede Anwendung wird mit all ihren Komponenten erstmalig einer Ablaufumgebung nach dem Round Robin-Verfahren bzw. einem gewichteten Round Robin-Verfahren zugeordnet, da eine Zuordnung mit den MVA-Algorithmen wegen fehlender Messdaten noch nicht möglich ist. Folgender Algorithmus wird periodisch angewendet:

- Es werden alle Ketten ausgewählt, die momentan inaktive Anwendungen repräsentieren. Bei solchen Anwendungen befindet sich der Benutzer nach dem Empfang einer Antwort in einer Denkphase.

- Beim Core-basierten Verfahren gilt, dass bei jeder gewählten Anwendungskomposition eine Komponente per Zufall ermittelt wird. Es werden mit Hilfe des Core-Algorithmus N Berechnungen hinsichtlich des Zielkriteriums durchgeführt, wobei sich die jeweiligen Konfigurationen in der Lokalität der ausgewählten Komponente unterscheiden. Es wird die hinsichtlich des Zielkriteriums beste Konfiguration ermittelt. Somit steht fest, ob und wohin die ausgewählte Komponente migriert.
- Beim Verfahren mit dem exakt rechnenden MVA-Algorithmus wird aus Komplexitätsgründen eine gesamte Anwendung und nicht eine einzelne Komponente als lastverursachende Instanz betrachtet. Mit Hilfe des MVA-Algorithmus werden wiederum N Berechnungen durchgeführt und die beste Konfiguration ermittelt. Das Verteilte System wird aufgrund der Berechnungen rekonfiguriert.

5.2.2.2 Dynamische Leistungsgrößenberechnung durch die Core-Heuristik

Jede Anwendung im Verteilten System hat im GPWN ihre Entsprechung durch eine Kette. Eine relevante Leistungsgröße ist die Antwortzeit, die aus der Summation aller Aufenthaltsdauern einer in der Kette zirkulierender Anforderung (außer der Aufenthaltsdauer beim Benutzer) berechnet wird.

In diesem Abschnitt wird vorgestellt, dass die Anwendung der Core-Heuristik zufriedenstellende Ergebnisse bei der Berechnung von Antwortzeiten liefert. Bei den hier untersuchten GPWNs mit jeweils einer Anforderung je Klasse entspricht der Linearizer-Algorithmus exakt dem Core-Algorithmus, da $d_{irs}(\underline{k}) = 0, \forall i, r, s$ gilt.

Betrachtung der Beeinflussung durch konkurrierende Instanzen

Eine Aufenthaltsdauer hängt wegen der sequentiellen Beauftragungsabfolge niemals von den anderen Komponenten ab, die mit der betrachteten Komponente zu einer Anwendung gehören. Es lässt sich die Gleichung (4.29) mit (4.31) beim Vorhandensein von Z Komponenten (d. h. Teilketten) zu

$$\tilde{t}_{iz}(\underline{k}) = \frac{1}{\mu_{iz}}\left(1 + \sum_{s=1,\, \zeta(s) \neq \zeta(z)}^{Z} \bar{k}_{iz}(\underline{k} - \underline{1}_{\zeta(s)})\right) \tag{5.3}$$

erweitern. Der Parameter $\bar{k}_{iz}$ ist hierbei die mittlere Anzahl von Anforderungen in einer Teilkette z, die einem Knoten i zugeordnet ist.

Die Core-Heuristik und ihre Modifikation

Da $e(\underline{k}, r) = 1, \forall r \in (1, \ldots, R)$ gilt, wird die Gleichung (4.38) zu

$$\bar{k}_{iz}(\underline{k} - \underline{1}_s) = \begin{cases} 0 & \zeta(z) = s \\ \bar{k}_{iz}(\underline{k}) & \zeta(z) \neq s \end{cases} \tag{5.4}$$

umgeformt. Mit der Gleichung (4.34), (5.3) und (5.4) erhält man dann die Gleichung

$$\dot{t}_{iz}(\underline{k}) = \frac{1}{\mu_{iz}} \left(1 + \sum_{s=1,\, \zeta(s) \neq \zeta(z)}^{Z} \frac{\dot{t}_{is}(\underline{k})}{\sum_{j=1}^{N} \frac{v_{jz}}{v_{iz}} \dot{t}_{js}(\underline{k})} \right). \tag{5.5}$$

Um den Core-Algorithmus anzuwenden, muss man mit Hilfe der Gleichungen (4.29), (4.31) und (4.37) eine initiale Berechnung von $\dot{t}_{iz}(\underline{k})$ vornehmen und dann iterativ die Gleichung (5.5) anwenden, bis sich $\dot{t}_{iz}(\underline{k})$ nur noch vernachlässigbar ändert. Für GPWNs, die mit der vorgestellten Methode modelliert werden, ist eine einfachere Berechnung der Startwerte als bei der ursprünglichen Core-Heuristik möglich. Anstatt die Gleichung (4.29), (4.31) und (4.37) anzuwenden, bietet sich für die Initialisierung eine Berechnung durch Gleichung (4.30) an. Bei der Startwertberechnung entspricht dann der Aufenthaltsdauer die Bedienzeit. Diese ist somit einfach als Teil eines Kantengewichts einem Aktivitätsgraphen zu entnehmen, wenn die Heuristik innerhalb einer Konfigurationsverwaltung ausgeführt wird.

Für die Anzahl der Rechenschritte gilt nun (im aufwändigsten Fall) die Beziehung

$$\Re \sim IZ^2, \tag{5.6}$$

wobei I der Anzahl der Iterationen entspricht. Schon für einen Iterationsschritt (d. h. für $I = 1$) werden, wie in Bild 5.2 dargestellt ist, zufriedenstellende Ergebnisse erzielt. Für die Anzahl der Rechenschritte gilt nun unter Berücksichtigung der Gleichung (5.6) mit $I = 1$ die Beziehung

$$\Re \sim Z^2. \tag{5.7}$$

Bei den Untersuchungen, deren Ergebnisse in Bild 5.2 dargestellt sind, werden bei einem Szenario Aufenthaltsdauern $\bar{t}_{core}$ mit dem Core-Algorithmus berechnet und mit den exakt berechneten Aufenthaltsdauern $\bar{t}_{exact}$ verglichen. Alle Anwendungskompositionen sind auf einer einzigen Ablaufumgebung lokalisiert. In Bild 5.2 sind die Verläufe der Wahrscheinlichkeitsverteilungsfunktion $P(X \leq x)$ dargestellt, wobei sich die Zufallsvariable X aus

$$X = 1 - \left| \frac{\bar{t}_{core}}{\bar{t}_{exact}} \right| \tag{5.8}$$

berechnet.

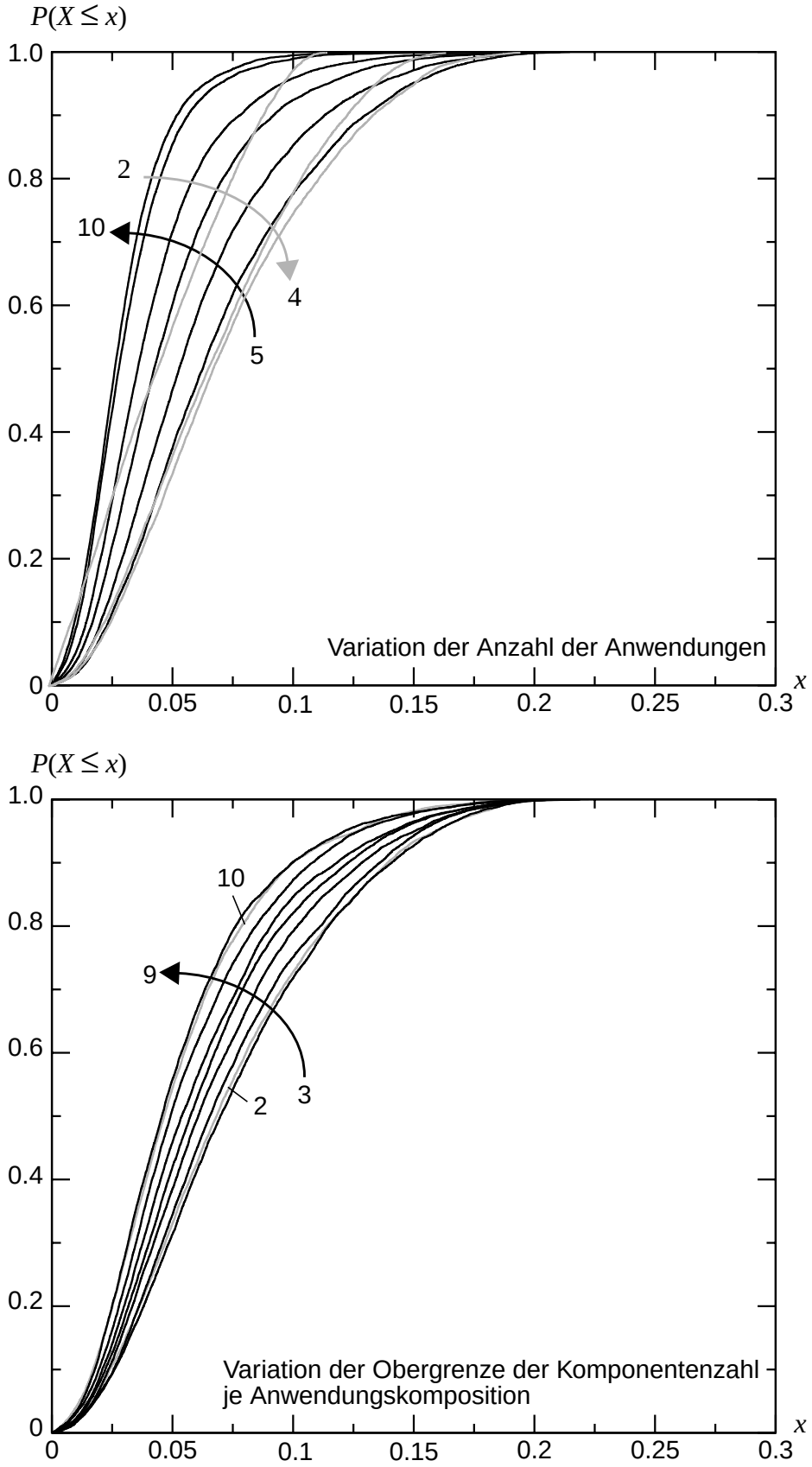


Bild 5.2: Kurvenscharen für $P(X \leq x)$ für $I = 1$

Zur Bestimmung von $P(X \leq x)$ werden hinreichend viele Szenarien betrachtet, bei denen die mittlere Denkzeit eines Benutzers und die mittlere Bedienzeit einer Komponente zufallsabhängig nach einer negativ-exponentiellen Verteilung bestimmt sind. Die obere Grafik zeigt eine Kurvenschar, bei der die Anzahl der Anwendungen variiert wird. Die Anzahl der Komponenten und damit die Anzahl der Teilketten je Anwendung ist aufgrund einer Gleichverteilung bestimmt und schwankt zwischen eins und fünf.

Die untere Grafik zeigt eine Kurvenschar, bei der die maximale Anzahl der Komponenten je Anwendung von eins bis zehn variiert. Die Anzahl der Anwendungen ist bei dieser Betrachtung immer fünf.

Da die exakt berechneten Aufenthaltsdauern $\bar{t}_{exact}$ durch den MVA-Algorithmus nur unter enormen Zeitaufwand berechnet werden können, wurden sie mit Hilfe der Simulationstechnik hinreichend genau bestimmt. Die Ergebnisse zeigen, dass die mit Hilfe des modifizierten Core-Algorithmus berechneten Aufenthaltsdauern zumeist diesen Werten sehr nahe sind.

Veranschaulichung der Effektivität bei nur einem Iterationsschritt

Die Betrachtung eines in Bild 5.3 gegebenen Beispiels zeigt die Umordnung von Komponenten aufgrund des Einsatzes eines Lastverteilungsverfahrens. Die linke Seite zeigt die Ausgangslage, bei der drei Anwendungen durch das Round Robin-Verfahren abwechselnd zwei Ablaufumgebungen zugeordnet sind. Auf der rechten Seite wird eine optimale Konfiguration mit der minimalen Gesamtantwortzeit dargestellt. Die mittleren Bedienzeiten für die Komponenten sind ebenfalls in Bild 5.3 angegeben. Dem Benutzer wird eine mittlere Denkzeit von 10.0 Zeiteinheiten zugeordnet.

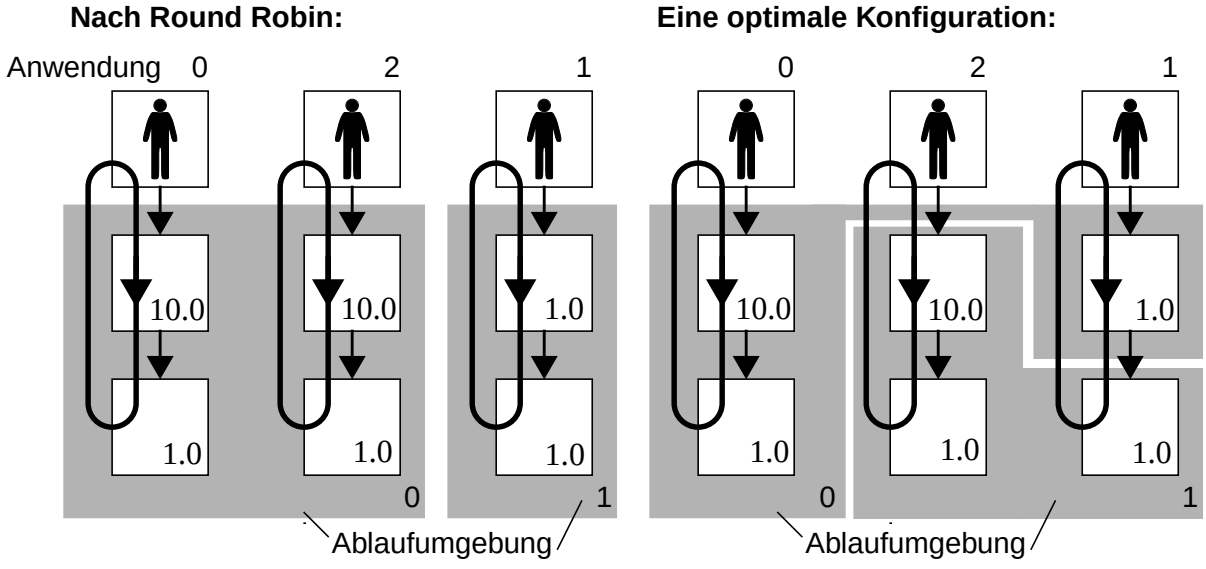


Bild 5.3: Exemplarische Betrachtung eines Szenarios vor und nach einer Rekonfigurierung

Anw.	Komp.	Umg.	I=0	I=1	I=10	exakt	Umg.	I=0	I=1	I=10	exakt
0	0	0	10.00	15.23	16.44	15.22	0	10.00	10.83	11.18	10.79
	1	0	1.00	1.52	1.64	1.52	0	1.00	1.08	1.12	1.08
1	0	1	1.00	1.00	1.00	1.00	0	1.00	1.54	1.55	1.52
	1	1	1.00	1.00	1.00	1.00	1	1.00	1.52	1.55	1.52
2	0	0	10.00	16.26	16.44	15.22	1	10.00	11.17	11.18	10.79
	1	0	1.00	1.63	1.64	1.52	1	1.00	1.12	1.12	1.08
		∑	24.0	36.64	38.16	35.48	∑	24.0	27.26	27.70	26.79
		Ausgangsszenario (nach Round Robin)					Optimale Konfiguration				

Tabelle 5.1: Berechnungen mit Hilfe von Core und MVA

Tabelle 5.1 zeigt die Berechnungen mit Hilfe des Core- und des exakt rechnenden MVA-Algorithmus für das Ausgangsszenario und das Szenario mit der optimalen Konfiguration. In jeder Zeile der Tabelle sind die berechneten Daten für eine Komponente angegeben. Die Ergebnisse eines Szenarios sind in der Tabelle mit Hilfe von 5 Spalten dargestellt. In der jeweils ersten Spalte sind die Umgebungen angegeben, zu denen eine Komponente zugeordnet ist. In der jeweils zweiten, dritten und vierten Spalte sind die Ergebnisse für die Core-Heuristik bei unterschiedlichen Iterationen angegeben. In der jeweils fünften Spalte sind die Ergebnisse bei exakter Berechnung durch MVA dargestellt. Die Ergebnisse zeigen, dass die mit $I = 1$ berechneten Werte sehr nahe den exakten Werte sind. Sie zeigen zudem, dass für $I = 0$ die Bediendauern als Aufenthaltsdauern angenommen werden und somit immer gleiche, konfigurationsunabhängige Gesamtantwortzeiten berechnet werden.

Eine weitere Veranschaulichung der Effektivität der modifizierten Core-Heuristik findet sich in Anhang A.

5.3 Bewertung der Tauglichkeit der Lastverteilungsverfahren

In diesem Abschnitt wird die Tauglichkeit der vorgestellten Lastverteilungsverfahren betrachtet. Insbesondere wird ihre Effektivität mit Hilfe von Methoden zur ereignisgesteuerten Simulation und zur mathematischen Analyse bewertet.

5.3.1 Methoden zur Bewertung der Effektivität

Es wird eine Bewertung der Effektivität der Verfahren vorgenommen, indem die Gesamtantwortzeiten von Anwendungen mit Hilfe der ereignisgesteuerten Simulation und der mathematischen Analyse quantitativ bestimmt werden. Migrationskosten, Informationsgewinnungskosten und Einflüsse durch die Informationsalterung werden erst in einer späteren Betrachtung

qualitativ mitberücksichtigt. Bei Verfahren, die Messungen über mittlere Ankunftsraten und mittlere Bedienzeiten in ihren Berechnungen einfließen lassen, werden diese Mittelwerte als gegebene Parameter angenommen. Ihre Näherung durch ein EMA-Verfahren wird bei den folgenden Betrachtungen nicht berücksichtigt. Insbesondere beim auslastungsbasierten Verfahren wird die Auslastung mit Hilfe der Gleichung (5.1) ermittelt.

5.3.1.1 Ermittlung von Leistungsgrößen durch eine ereignisgesteuerte Simulation

Bei den Simulationen werden hinreichend viele Szenarien betrachtet, um allgemeine Aussagen über Leistungsgrößen zu erhalten. Der Ablauf eines Szenarios besteht in der Regel aus zwei Phasen. In einer ersten Phase ändert sich die Systemlast aufgrund der Aufnahme von neuen Anwendungen ins Verteilte System, bis eine Endkonfiguration erreicht worden ist. Die Anwendungen werden hierbei nicht vom Verteilten System entfernt. In dieser dynamischen Phase arbeitet das zu untersuchende Lastverteilungsverfahren. In der zweiten, statischen Phase wird die Endkonfiguration nicht mehr geändert und das zu untersuchende Lasterteilungsverfahren arbeitet nicht mehr. Für diese Phase wird eine Leistungsbewertung durch eine ereignisgesteuerte Simulation durchgeführt, indem die mittleren Antwortzeiten je Anwendung berechnet werden. Durch die Bestimmung der Gesamtantwortzeit bzw. durch eine zusätzliche Bewertung von Fairness-Aspekten lässt sich dann die Effektivität des zu untersuchenden Verfahrens quantitativ bestimmen.

Ein Szenario wird durch folgende Parameter beschrieben: Das Verteilte System besteht aus `NUM_ENVS` Ablaufumgebungen (`NUM_ENVS` = N), auf denen Komponenten von `NUM_APPS` Anwendungen (`NUM_APPS` = R) zugeordnet werden. Eine Anwendung setzt sich aus maximal `MAX_NUM_COMPS` Komponenten zusammen. Die Komponenten beinhalten eine mittlere Bedienzeit, die auf einer idealen Ablaufumgebung gemessen wird. Zudem wird jeder Anwendung eine mittlere Benutzerdenkzeit zugeordnet. Die tatsächlichen Bedienzeiten und Denkzeiten sind zufallsbestimmt und negativ-exponentiell verteilt.

5.3.1.2 Ermittlung von Leistungsgrößen durch eine Mittelwertanalyse

Eine mathematisch analytische Leistungsgrößenberechnung ist in einigen Fällen durchaus möglich. So lässt sich der schlechteste Fall (engl. *Worst Case*) mittels MVA betrachten. Hier findet keine Lastverteilung statt und alle Komponenten sind nur einer Ablaufumgebung zugeordnet. Dieser Fall kann bei bezüglich aller Komponenten konstanten, mittleren Bedienzeiten und bezüglich aller Anwendungen konstanten, mittleren Benutzerdenkzeiten und bei konstanter Anzahl von Komponenten je Anwendung, auf folgende Weise analysiert werden:

Ausgehend von der Gleichung (4.29), (4.31) und (4.34) gilt, wenn R Anwendungen einer Ablaufumgebung zugeordnet sind, für die mittlere Antwortzeit der Anwendung r die Gleichung

$$\bar{t}_{ir}(\underline{k}) = \frac{1}{\mu'_{ir}}\left(1 + \sum_{s=1, s \neq r}^{R} \frac{\bar{t}_{is}(\underline{k} - \underline{1}_r)}{\bar{t}_{is}(\underline{k} - \underline{1}_r) + \tau}\right). \tag{5.9}$$

Der Parameter τ entspricht hierbei der bezüglich aller Anwendungen konstanten, mittleren Benutzerdenkzeiten. Wegen der Gleichheit aller Anwendungen lässt sich Gleichung (5.9) zu

$$\bar{t}_R = \frac{1}{\mu'}\left(\frac{R\bar{t}_{R-1} + \tau}{\bar{t}_{R-1} + \tau}\right) \tag{5.10}$$

umformen. Der Parameter $\bar{t}_R$ entspricht der mittleren Antwortzeit einer Anwendung bei *R* existierenden Anwendungen. Die Konstante μ' ist die mittlere Bedienzeit, die ebenfalls für alle Anwendungen gleich ist. Es gilt

$$\bar{t}_0 = 0. \tag{5.11}$$

Bei $\bar{t}_{R-1} \gg \tau$, d. h. bei entsprechend großem *R*, gilt die Beziehung

$$\bar{t}_R \approx \frac{R}{\mu'}. \tag{5.12}$$

5.3.2 Effektivität bei unterschiedlich homogenen Verteilten Systemen

Bei Szenarien mit sogenannten fast homogenen Verteilten Systemen gilt:

- Alle Ablaufumgebungen, insbesondere auch die Referenzablaufumgebung, besitzen die gleiche Kapazität.
- Die Anzahl der Komponenten je Anwendung ist gemäß einer Gleichverteilung zufallsbestimmt. Eine Anwendung setzt sich aus mindestens einer und maximal `MAX_NUM_COMPS` Komponenten zusammen.
- Die mittleren Bedienzeiten und die mittleren Benutzerdenkpausen sind gemäß einer negativ-exponentiellen Verteilung zufallsbestimmt. Der Mittelwert aller mittleren Bedienzeiten entspricht `MEAN_SERVICES_TIME`. Der Mittelwert aller mittleren Benutzerdenkzeiten entspricht `MEAN_THINKINGS_TIME`.
- Es werden keine Komponenten, die Systemdienste anbieten, betrachtet.

Selbst bei diesen fast homogenen Verteilten Systemen treten Inhomogenitäten auf:

- Alle möglichen mittleren Bedienzeiten und alle möglichen mittleren Benutzerdenkzeiten sind gemäß einer negativ-exponentiellen Verteilung zufallsbestimmt. Dies bedeutet, dass die mittleren Bedienzeiten der betrachteten Komponenten bzw. die mittleren Benutzerdenkzeiten bei den betrachteten Anwendungen nicht konstant sind. Erst bei Betrachtung hinrei-

chend vieler Szenarien liegt der Mittelwert von allen mittleren Bedienzeiten hinreichend nahe an MEAN_SERVICES_TIME und der Mittelwert von allen mittleren Benutzerdenkzeiten hinreichend nahe an MEAN_THINKINGS_TIME.

- Es gibt Verfahren, die eine Anwendung mitsamt ihren Komponenten einer einzigen Ablaufumgebung zuordnen. Hierdurch entstehen Inhomogenitäten:
  - Zunächst ist eine ungleiche Belastung von Ablaufumgebungen mit Anwendungen gegeben, wenn die Modulo-Operation $R \; mod \; N \neq 0$ ist.
  - Desweiteren schwankt die Anzahl der Komponenten je Anwendung zwischen eins und MAX_NUM_COMPS.

5.3.2.1 Effektivität bei fast homogenen Verteilten Systemen

In Bild 5.4 sind Ergebnisse für die dort aufgeführten Parameter dargestellt. Es ist je Verfahren die mittlere Antwortzeit für eine Anwendung dargestellt, wobei die Anzahl der bei der Endkonfiguration sich im Verteilten System befindenden Anwendungen variiert wird.

Mit dem anwendungsbezogenem Round Robin-Verfahren, dem auslastungsbasierten Verfahren und den MVA-basierten Verfahren wird erreicht, dass die mittlere Antwortzeit einer Anwendung bei $R = 4$ gleich dem Wert 3.0 ist. Dies kann folgendermaßen erläutert werden: Jede Anwendung ist alleine einer Ablaufumgebung zugeordnet und ihre Antwortzeiten werden von keiner Störlast beeinflusst. Im Mittel besitzt eine Anwendung drei Komponenten, so dass sich ihre mittlere Antwortzeit zu 3 * MEAN_SERVICES_TIME = 3.0 ergibt.

Wegen der gegebenen Homogenität arbeitet das anwendungsbezogene Round Robin-Verfahren sehr gut. Es arbeitet besser als das Random- und das Shortest Queue-Verfahren. Darüberhinaus ist es auch effektiver als das komponentenbezogene Round Robin-Verfahren, da Komponenten einer Anwendung wegen ihrer sequentiellen Beauftragungen sich nicht gegenseitig stören und es sich somit lohnt, sie auf einer Ablaufumgebung anzuordnen.

Das auslastungsbasierte Verfahren arbeitet noch besser als das Round Robin-Verfahren. Es besitzt gegenüber dem Round Robin-Verfahren eine genauere Kenntnis über den Systemzustand. Es wird hierbei berücksichtigt, an welchen Orten eine zu transferierende Instanz am wenigsten Konkurrenz erfährt. Es wird jedoch nicht betrachtet, wie diese Instanz sich als Konkurrenz für die anderen Instanzen auswirkt. Dieser zusätzliche Aspekt wird bei den MVA-basierten Verfahren beachtet. Deshalb sind die Ergebnisse für das Core-basierte Verfahren noch besser. Am effektivsten arbeitet das Verfahren, das exakte MVA-Berechnungen vornimmt, obwohl es gegenüber dem Core-basierten Verfahren, das einzelne Komponenten transferiert, nur gesamte Anwendungen als lastverursachende Instanzen ansieht.

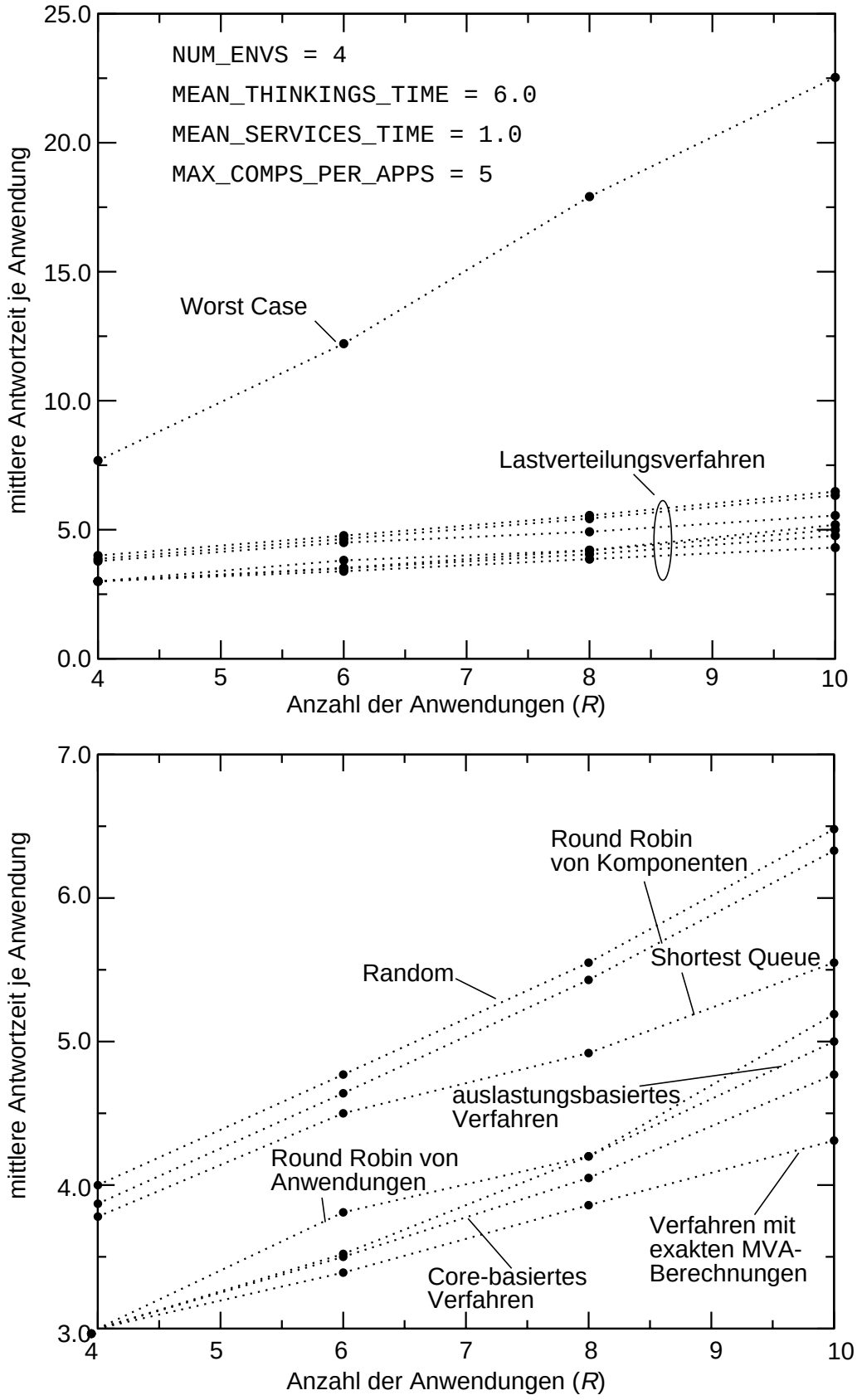


Bild 5.4: Vergleich der Lastverteilungsverfahren bei homogenen Verteilten Systemen

Die beiden MVA-basierten Verfahren können die vorkommenden Inhomogenitäten am besten ausgleichen, da sie diese in ihren Berechnungen mitberücksichtigen. Im völlig homogenen Fall bringen diese beiden Verfahren jedoch keine Verbesserungen gegenüber dem Round Robin-Verfahren und dem auslastungsbasierten Verfahren.

Im Folgenden wird von den MVA-basierten Verfahren nur noch das Core-basierte Verfahren betrachtet. Wie noch in Abschnitt 5.3.3 näher gezeigt wird, ist die Komplexität der exakten MVA-Berechnungen bei zur Systemlaufzeit arbeitenden Lastverteilungsverfahren nicht akzeptabel.

5.3.2.2 Effektivität bei inhomogeneren Verteilten Systemen

Mit den bisherigen Ergebnissen wurde gezeigt, dass die MVA-basierten Verfahren zwar am effektivsten arbeiten, dass jedoch wegen der betrachteten Homogenität das auslastungsbasierte Verfahren und das Round Robin-Verfahren ihnen sehr nahe kommen. Im Folgenden wird der Einfluss von Inhomogenitäten auf die Effektivität der Verfahren untersucht.

Effektivität bei Inhomogenitäten bei der Lastaufnahme

Im Folgenden werden Ablaufumgebungen betrachtet, die sich in ihrer Prozessorkapazität unterscheiden. Neben den schon eingeführten Verfahren wird nun zusätzlich ein gewichtetes Round Robin-Verfahren betrachtet. Bei der Wahl der Gewichte wird eine Strategie verfolgt, bei der eine Ablaufumgebung, die verglichen mit einer Referenzablaufumgebung *n*-fache Kapazität aufweist, *n* mal soviele Anwendungen aufnehmen soll. Wenn n_i die *n*-fache Kapazität einer Ablaufumgebung *i* gegenüber einer idealen Referenzablaufumgebung ist und a_i die Anzahl der Anwendungen, die dieser Ablaufumgebung schon während der Systemlaufzeit zugeordnet wurden, dann wird mit

$$z_i = \frac{n_i}{(a_i + 1)} \tag{5.13}$$

ihr eine Zahl z_i zugeordnet. Die nächste ins System kommende Anwendung wird der Ablaufumgebung *i* mit maximalem z_i zugeteilt. Es sei an dieser Stelle darauf hingewiesen, dass es zu diesem gewichteten Round Robin-Verfahren eine bekannte Analogie gibt. Das Höchstzahlenverfahren von d'Hondt zur Zuteilung von Sitzplätzen in einem Parlament an Parteien gemäß ihres bei einer Wahl erreichten Stimmenanteils arbeitet nach dem gleichen Prinzip.

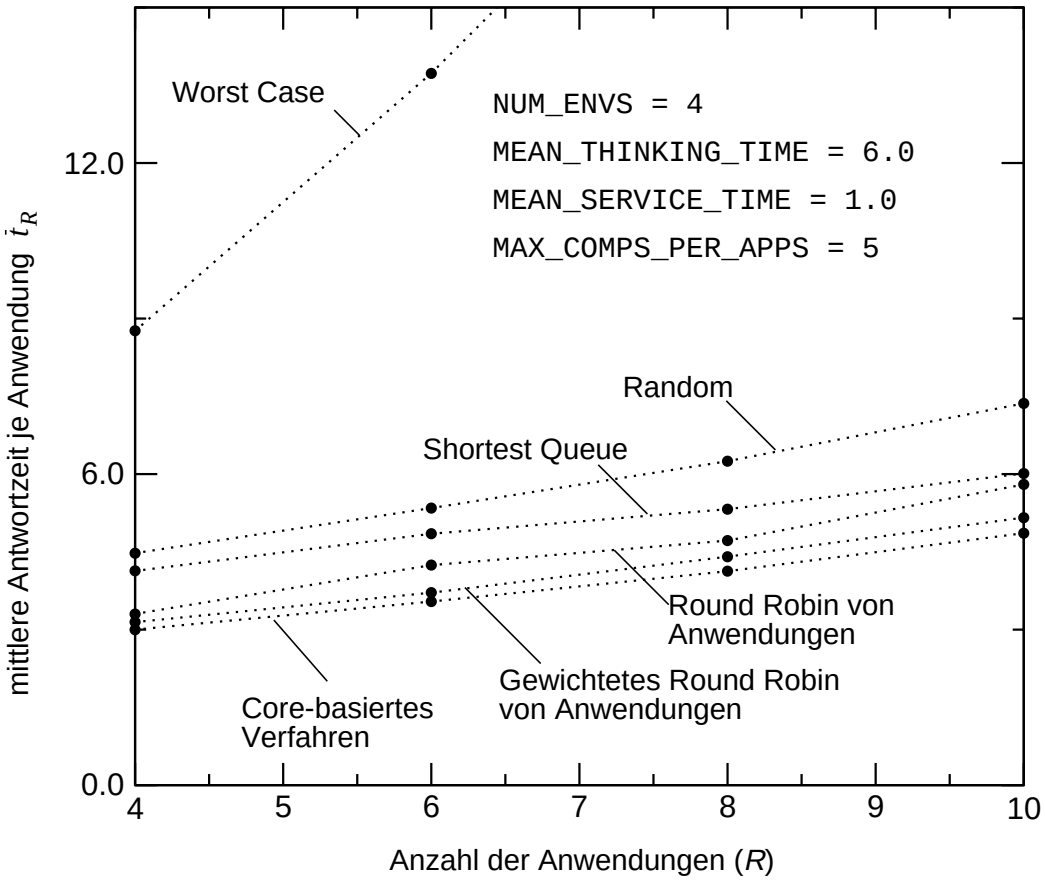


Bild 5.5: Inhomogenitäten durch Systemdienste realisierende Komponenten

Die Ergebnisse in Bild 5.5 beziehen sich auf Szenarien, bei denen die Kapazität der Ablaufumgebungen nach einer Gleichverteilung zufallsabhängig ist. Es gilt $1/2 \leq n_i < 3/2$ $\forall i = 1, \ldots, N$. Man sieht, dass das gewichtete Round Robin-Verfahren sehr gut arbeitet und die Inhomogenitäten bezüglich der Lastaufnahme sehr gut ausgleichen kann. Das Core-basierte Verfahren arbeitet am effektivsten.

Effektivität bei Inhomogenitäten bei der Lasterzeugung

Es werden nun Anwendungen betrachtet, die sich stärker als bisher in ihrer Lasterzeugung unterscheiden. Bei den bisherigen Betrachtungen sind die mittleren Bedienzeiten der Komponenten negativ-exponentiell verteilt, so dass der Mittelwert aller mittleren Bedienzeiten `MEAN_SERVICES_TIME` entspricht. Bei den folgenden Betrachtungen existieren mit `MEAN_SERVICES_TIME1` und `MEAN_SERVICES_TIME2` zwei solche Mittelwerte. Jeder Anwendung wird nun per Zufall einer der beiden Werte zugeordnet, so dass man wiederum ihren Komponenten eine mittlere Bedienzeit per Zufall gemäß einer negativ-exponentiellen Verteilung mit diesem Wert als Mittelwert zuordnen kann. Die Ablaufumgebungen unterscheiden sich nicht in ihrer Kapazität.

In Bild 5.6 sind Ergebnisse dargestellt, die einen Vergleich der Effektivität zwischen dem Round Robin-Verfahren und dem Core-basierten Verfahren zeigen. Es wird gezeigt, dass das Core-basierte Verfahren bei der gegebenen Art der Inhomogenität signifikant effektiver arbeitet als das Round Robin-Verfahren.

In Bild 5.6 ist als zusätzlicher Aspekt dargestellt, dass die Effektivität nach unterschiedlichen Kriterien bestimmt werden kann. Bei der ersten Untersuchung wird ein Verfahren als effektiv angesehen, wenn es wie bisher die Summe der Gesamtantwortzeiten minimiert. Bei einer weiteren Untersuchung wird jedoch ein Verfahren erst dann als effektiv angesehen, wenn es dazu noch möglichst fair arbeitet. In Bild 5.6 ist dargestellt, dass das Core-basierte Verfahren nach unterschiedlichen Zielkriterien erfolgreich optimieren kann.

Eine Optimierung nach einem Fairness-Kriterium wird hierbei wie folgt vorgenommen: Für eine Anwendung *r* ist der Fairness-Faktor q_r durch die Beziehung

$$q_r = \mu'_r \sum_{i=1}^{N} \sum_{z=1, r=\varsigma(z)}^{Z} \Delta(i,z)\dot{t}_{iz} \tag{5.14}$$

gegeben. Da die Kapazitäten aller Ablaufumgebungen gleich groß sind, gilt $\mu'_r = \mu'_{ir}$ $\forall i = 1, \ldots, N$.

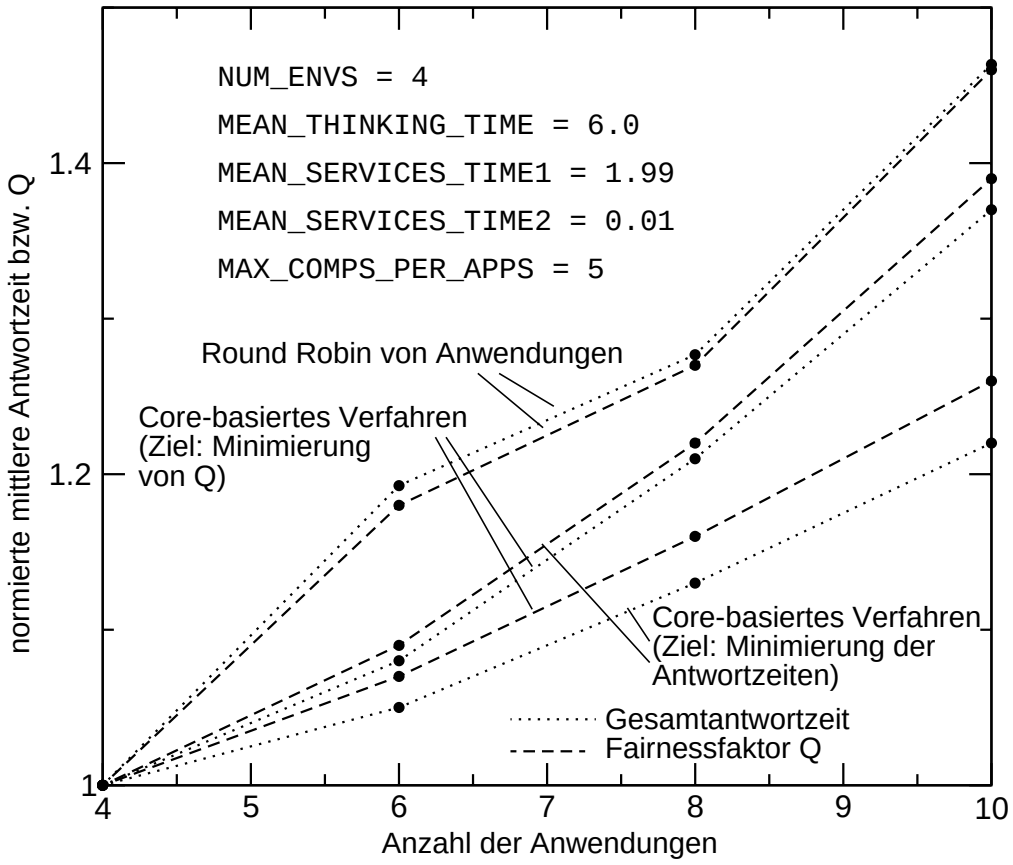


Bild 5.6: Inhomgenitäten bei der Lasterzeugung

Um jede Anwendung möglichst fair zu behandeln, soll die Summe

$$Q = \sum_{r=1}^{R} q_r \tag{5.15}$$

minimiert werden.

Effektivität bei Berücksichtigung von Systemdiensten

Bei den bisherigen Untersuchungen wurden keine Systemdienste berücksichtigt. Systemdienste realisierende Komponenten können einen Dienst für mehrere Anwendungen erbringen und deshalb ihre Ablaufumgebungen aufgrund der Vielfachnutzung stark belasten. Eine Inhomogenität im Verteilten System ist gegeben, wenn durch Systemdienste realisierende Komponenten die Ablaufumgebungen ungleich belastet werden.

In Bild 5.7 sind die Ergebnisse von Round Robin und dem Core-basierten Verfahren für ein exemplarisches Szenario dargestellt. Die Parameter entsprechen alle dem fast homogenen Szenario mit den gegebene Inhomogenitäten aus Abschnitt 5.3.2.1. Auf einer Ablaufumgebung ist jedoch eine Komponente lokalisiert, die einen Systemdienst realisiert, der von jeder Anwendung genutzt werden muss. Eine Anwendung kann folglich minimal eine Komponente und

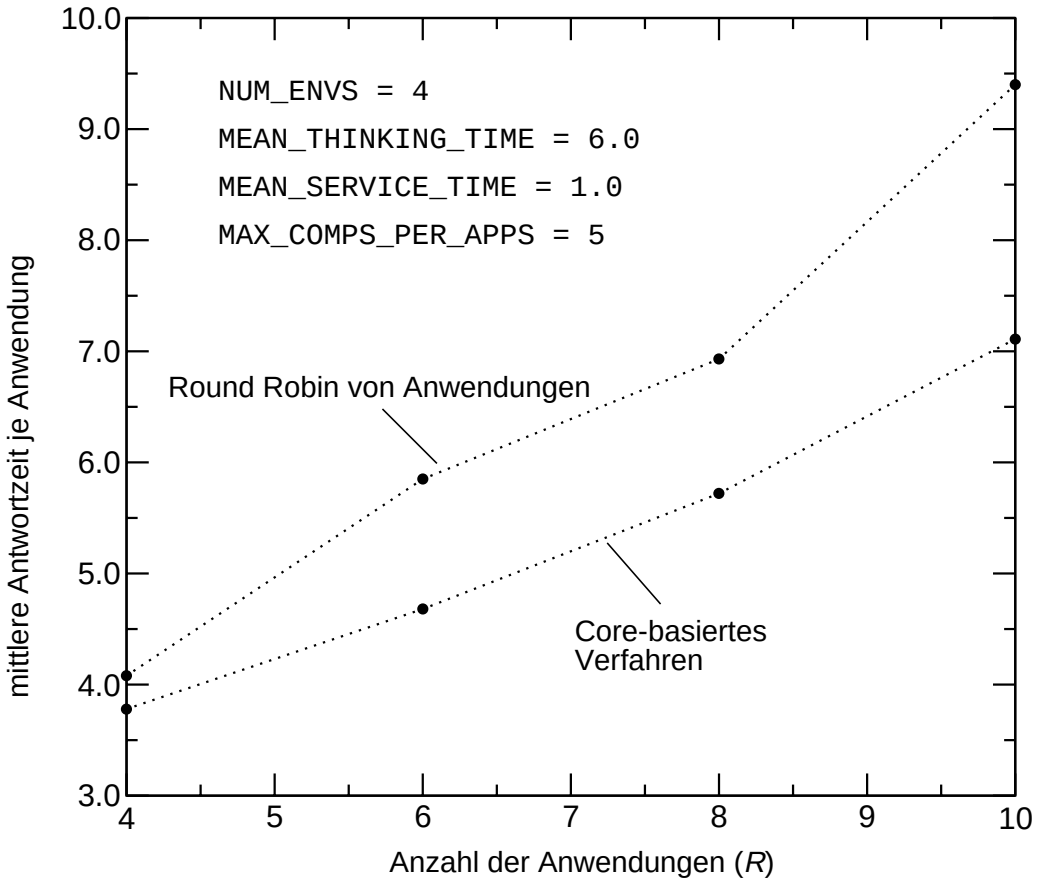


Bild 5.7: Inhomogenitäten durch Systemdienste realisierende Komponenten

maximal fünf Komponenten beinhalten, wobei eine Komponente immer derjenigen entspricht, die diesen Systemdienst realisiert. Die Ergebnisse zeigen, dass das Core-basierte Verfahren diese Art von Inhomogenitäten berücksichtigt und dass es gegenüber dem Round Robin-Verfahren die Systemleistung signifikant verbessert.

5.3.3 Begrenzende Einflüsse auf die Tauglichkeit der Verfahren

Von den bisher untersuchten Verfahren werden im Folgenden drei Verfahren genauer betrachtet: Das (gewichtete) Round Robin-Verfahren, das auslastungsbasierte Verfahren und das Core-basierte Verfahren. In diesem Abschnitt wird untersucht, wie noch nicht berücksichtigte Aspekte die Effektivität begrenzen. Der Abschnitt schließt mit der Betrachtung, ob das Core-basierte Verfahren konvergiert.

5.3.3.1 Informationsgewinnung und Informationsalterung

Die Gewinnung, Verarbeitung und Speicherung von Informationen kann die Effektivität beeinträchtigen, da durch diese Vorgänge zusätzlich Last erzeugt wird. Zudem wird durch die Alterung der gespeicherten Informationen der aktuelle Zustand des Verteilten Systems zum Zeitpunkt einer Entscheidungsfindung verfälscht wiedergegeben. Dies beeinträchtigt ebenfalls die Effektivität.

Für die betrachteten, relevanten Verfahren gilt:

- Round Robin-Verfahren: Der Aufwand zur Gewinnung der Informationen, die zur Entscheidungsfindung benötigt werden, ist gering. Beim gewichteten Round Robin-Verfahren müssen gemäß der Gleichung (5.13) nur die Parameter a_i und n_i ermittelt werden. Die Informationen werden nicht aufgrund von Messungen gewonnen und beschreiben deshalb i. A. nur sehr grob den Zustand des Verteilten Systems.

- Auslastungsbasiertes Verfahren: Der Aufwand zur Gewinnung von Informationen ist höher als beim Round Robin-Verfahren, da die Prozessorauslastung für jede Ablaufumgebung gemessen werden muss. Die Informationen können z. B. mit Hilfe des EMA-Algorithmus gewonnen werden, so dass je nach Parameterwahl entweder die Werte eher veraltet sind oder zu sehr einen Momentanwert und keinen Mittelwert beschreiben. Letzteres ist problematisch, da nur für kurze Zeit gültige Lastspitzen in die Entscheidungsfindung nicht einfliessen sollen. In [7] werden diese Aspekte näher untersucht.

- Core-basiertes Verfahren: Der Aufwand zur Gewinnung von Informationen ist hier am höchsten. Es müssen innerhalb der Prozessverwaltungen die mittleren Bedienzeiten und mittleren Beauftragungsraten bei Komponenten gemessen werden. Die Probleme durch den Einsatz eines EMA-Verfahrens zur Mittelwertbildung sind hier ebenfalls gegeben. Zudem muss berücksichtigt werden, dass nicht mehrmals innerhalb eines gegebenen Zeitintervalls

Informationen zu einer zentralen Instanz gesendet werden dürfen. Je größer dieses Intervall ist, desto größer ist auch die Gefahr, dass die Information zu sehr veraltet ist. Je geringer es ist, desto größer ist jedoch die Belastung für das Verteilte System durch die Informationsgewinnung.

5.3.3.2 Einfluss der Migration

Durch Aufrufblockierungen, die nach Chen (siehe auch Abschnitt 3.1.2.2) die für die Migration notwendige Inaktivität einer Komponente erzwingen, und durch die eigentliche Migration wachsen die Benutzerantwortzeiten während einer Rekonfigurierungsphase. Eine Gefahr besteht nun darin, dass das Verfahren soviele Rekonfigurierungen veranlasst, dass die im Gesamten betrachtete mittlere Antwortzeit höher als diejenige ist, die sich ohne Einsatz des Verfahrens ergeben hätte. Dies ist möglich, wenn das Anwachsen von Antwortzeiten aufgrund von Migrationsvorgängen die Oberhand gegenüber der Reduzierung von Antwortzeiten aufgrund der eigentlichen Lastverteilung gewinnt. Das Verfahren arbeitet in diesem Fall ineffektiv.

5.3.3.3 Skalierbarkeit der MVA-basierten Verfahren – Komplexität der Algorithmen

Für eine Bewertung der Skalierbarkeit der MVA-basierten Verfahren wird die Zunahme der algorithmischen Komplexität bei wachsender Größe des Verteilten Systems betrachtet:

- Für die Komplexität des Aufbaus eines Aktivitätsgraphen gilt:

  Die Schnittstellen, die eine neu ins System kommende Komponente nutzt oder anbietet, müssen ermittelt werden. Hierzu muss jeder Knoten von allen existierenden Aktivitätsgraphen untersucht werden, ob die durch ihn repräsentierte Komponente zur aktuell betrachteten Komponente schnittstellenkompatibel ist, d. h. ob beide eine Bindung eingehen können. Bei Z Komponenten und W Schnittstellenbeziehungen im Verteilten System gilt für die Anzahl der Rechenschritte bei einer vorzunehmenden Traversierung z. B. durch BFS oder DFS

  $$\Re \sim ZW\,. \tag{5.16}$$

- Für die Komplexität der Transformation eines Aktivitätsgraphen in Ketten gilt:

  Die Zeit für die Traversierung eines Aktivitätsgraphen durch DFS- bzw. BFS-Algorithmen wächst linear zur Anzahl der Knoten und Kanten. Für die Anzahl der Rechenschritte gilt wiederum die in der Gleichung (5.16) ausgedrückte Beziehung.

- Für die Komplexität der Auswahl von lastverursachenden und -aufnehmenden Instanzen beim Core-basierten Verfahren gilt:

Nach Auswahl einer Komponente werden mit Hilfe der Core-Heuristik jeweils für N Konfigurationen Berechnungen durchgeführt, um hieraus die beste Konfiguration zu ermitteln. Die Zeit für die Ausführung der Core-Heuristik wächst nach Gleichung (5.7) quadratisch mit der Anzahl der Komponenten im Verteilten System. Somit gilt insgesamt für die Anzahl der Rechenschritte

$$\Re \sim Z^2 N\,. \tag{5.17}$$

Das Core-basierten Verfahren ist wegen des quadratischen Anwachsens der Anzahl der Rechenschritte in seiner Skalierbarkeit begrenzt. Es ist jedoch durchaus verwendbar, wenn es innerhalb Verteilter Systeme eingesetzt wird, zu denen nur eine begrenzte Anzahl von Benutzern Zugang haben und die deshalb nur eine begrenzte Anzahl von Komponenten beinhalten. Dies ist beispielsweise bei Verteilten Systemen für die Fahrzeugtelematik durch die räumliche Begrenzung des Fahrzeugs gegeben. Hier kann ein solches Verfahren Teil einer zentral arbeitenden Konfigurationsverwaltung sein, die auf einem leistungsstarken Rechner abläuft.

- Für die Komplexität der Auswahl von lastverursachenden und -aufnehmenden Instanzen beim Verfahren, das mit exakten MVA-Berechnungen arbeitet, gilt:

  Nach Auswahl einer Anwendung werden mit Hilfe von MVA jeweils für N Konfigurationen Berechnungen durchgeführt, um hieraus die beste Konfiguration zu ermitteln. Die Zeit für die Ausführung der Mittelwertanalyse für eine Konfiguration wächst nach Gleichung (4.36) exponentiell mit der Anzahl der Anwendungen R im Verteilten System. Somit gilt insgesamt für die Anzahl der Rechenschritte

  $$\Re \sim 2^R N\,. \tag{5.18}$$

  Dieses exponentielle Anwachsen der Anzahl der Rechenschritte bereitet große Schwierigkeiten für die Skalierbarkeit. Der Einsatz dieses Verfahrens während der Systemlaufzeit wird deshalb nicht als sinnvoll erachtet.

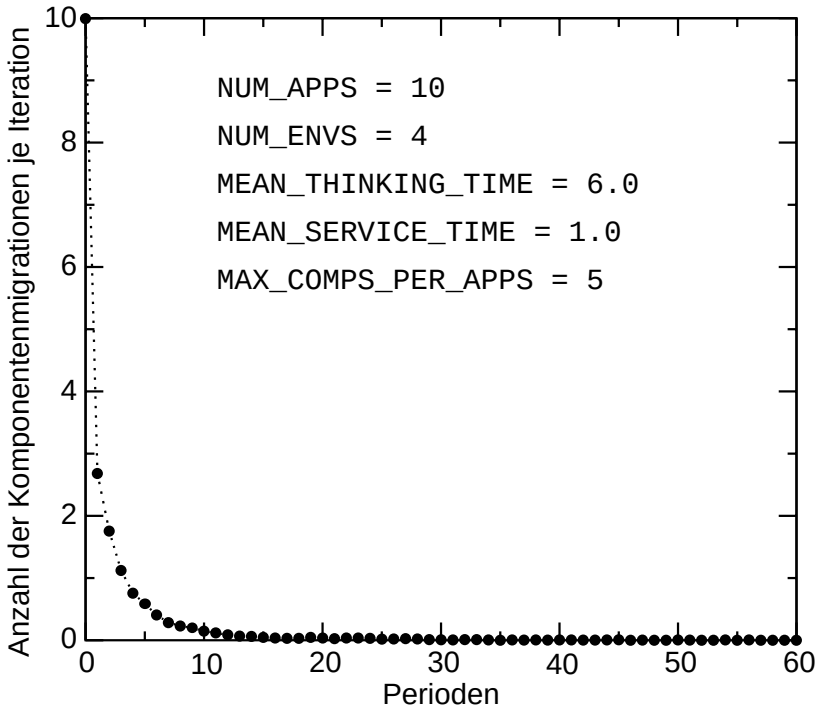


Bild 5.8: Konvergenz des Lastverteilungsverfahrens

5.3.3.4 Konvergenz des Core-basierten Verfahrens

Die Frage, ob das Core-basierte Verfahren konvergiert, wird durch Simulation von Szenarien untersucht, deren Ergebnisse in Bild 5.8 dargestellt sind. Es gelten die gleichen Parameter wie bei den Untersuchungen der Effektivität des Verfahrens bei homogenen Verteilten Systemen in Abschnitt 5.3.2.1. Zudem gilt, dass zehn Anwendungen (*NUM_APPS* = 10) ins Verteilte System aufgenommen werden, so dass im Mittel ein Verteiltes System je Szenario 30 Komponenten beinhaltet. Es werden alle Komponenten zuerst auf einer Ablaufumgebung angeordnet und danach wird das Verfahren zur Rekonfigurierung periodisch immer neu gestartet. Man sieht, dass das Verfahren konvergiert, da die Anzahl der Komponentenmigrationen auf null abnimmt. Es stellt sich folglich ein Gleichgewichtszustand ein.

5.3.4 Bewertung der Einsatzmöglichkeiten der Verfahren

Die Untersuchungen zeigen, dass das Round Robin-Verfahren, das auslastungsbasierte Verfahren und das Core-basierte Verfahren geeignete Kandidaten für den Einsatz bei lastbezogenen Rekonfigurierungen sind. Welches Verfahren am meisten geeignet ist, hängt sowohl von der grundsätzlichen Leistungsfähigkeit des Verteilten Systems als auch von den darin vorkommenden Inhomogenitäten ab.

Das Core-basierte Verfahren benötigt viel Rechenleistung und Speicherplatz. Je leistungsschwächer das Verteilte System ist, desto ineffektiver arbeitet das Verfahren. Sein Vorteil ist,

dass es vorhandene Inhomogenitäten am besten berücksichtigt. Je inhomogener das Verteilte System ist, desto effektiver arbeitet das Verfahren im Vergleich zu den anderen.

Das Round Robin-Verfahren verhält sich gegensätzlich zum Core-basierten Verfahren. Es benötigt wenig Rechenleistung und Speicherplatz und ist deshalb auch für den Einsatz in leistungsschwachen Verteilten Systemen geeignet. Sein Nachteil ist, dass es vorkommende Inhomogenitäten kaum berücksichtigen kann. In diesem Zusammenhang sei darauf hingewiesen, dass die in dieser Arbeit vorgestellten Ergebnisse bezüglich des Round Robin-Verfahrens auch deshalb so gut sind, weil keine Inhomogenitäten betrachtet wurden, die durch das willkürliche Entfernen von hinzugenommenen Anwendungen zustande kommen.

Zwischen den MVA-basierten Verfahren und dem Round Robin-Verfahren ist das auslastungsbasierte Verfahren angesiedelt. Es steht sowohl bei der Aufwandsbetrachtung als auch bei der Betrachtung der Reaktionsfähigkeit auf Inhomogenitäten zwischen den anderen Verfahren.

Kapitel 6

Rekonfigurierung als Teil der Ressourcen-Verwaltung

In diesem Kapitel wird eine umfassende Ressourcen-Verwaltung für komponentenbasierte Verteilte Systeme betrachtet, die eine Konfigurationsverwaltung beinhaltet. Es wird dargestellt, dass die Reservierung von Ressourcen für konstantbitratige A/V-Ströme und ihre Freigabe die Prozessorkapazität der betroffenen Komponentenablaufumgebungen schwanken lassen kann. In solchen Fällen ist der Einsatz von Verfahren zur lastbezogenen Rekonfigurierung lohnenswert, da durch das Umordnen von Komponenten die Systemleistung signifikant verbessert werden kann.

Zunächst werden allgemein Mechanismen zur Reservierung von Ressourcen vorgestellt. Anschließend wird auf die im 3. Kapitel eingeführte Software-Architektur DANA nochmals näher eingegangen. DANA-Systeme beinhalten sowohl Ressourcen-Reservierungsmechansimen als auch Verfahren zur lastbezogenen Rekonfigurierung.

6.1 Mechanismen zur Reservierung von Ressourcen

Es werden Mechanismen, mit denen sowohl bei einem Bussystem als auch bei einem Endgerät Ressourcen reserviert werden können, vorgestellt.

6.1.1 Reservierung von Übertragungskapazität

Bei Bussystemen existieren einige Ausprägungen, die die Reservierung von Übertragungskapazität vorsehen. So kann ein SBM für ein Ethernet-Bussystem oder ein IRM (*Isochronous Resource Manager,* [56]) für ein IEEE 1394-Bussystem eingesetzt werden. Unter Verwendung von synchronen Übertragungsdiensten ist bei dieser Art der Ressourcen-Reservierung eine Garantie gegeben, dass eine Belegungsanforderung erfolgreich zur Belegung der Ressource innerhalb einer gegebenen Zeitspanne führt, falls keine Überreservierung vorkommt. Dies wird durch spezielle Mechanismen für die Arbitrierung des Übertragungsmediums realisiert, bei

denen eine priorisierte Behandlung von synchronen gegenüber asynchronen Übertragungsdiensten stattfindet. So darf beim Bussystem IEEE 1394 für eine Übertragung von Daten mittels eines asynchronen Übertragungsdienstes erst dann eine Arbitrierung erfolgen, nachdem für eine gewisse Zeit keine Übertragung stattgefunden hat. Eine Arbitrierung zur Übertragung von Daten mit Hilfe eines synchronen Übertragungsdienstes kann jedoch innerhalb einer kürzeren Zeitspanne vorgenommen werden.

6.1.2 Reservierung von Prozessorkapazität

Bei Endgeräten können ebenfalls durch spezielle Verwaltungsinstanzen Ressourcen in Form von Prozessorkapazität reserviert werden. Eine Möglichkeit ist, bei einer EDF-Prozessverwaltung „synchrone“ Dienste durch zyklische Zuteilung von Prozessen mit endlicher Periodendauer bereitzustellen. Die Periodendauern bestimmen hierbei die Fertigstellungstermine. „Asynchrone“ Dienste ermöglichen die einmalige Zuteilung von Prozessen mit keinem Fertigstellungstermin. Auch hier erhält durch die EDF-Strategie eine Anforderung, die mit Hilfe eines synchronen Dienstes bearbeitet wird, immer den Vorzug gegenüber einer mittels eines asynchronen Dienstes bearbeiteten Anforderung. Wenn derartige synchrone Dienste von einer Prozessverwaltung angeboten werden und von einem periodisch bereitzustellenden Prozess die mittleren Prozessorbelegungszeiten bei den Bereitstellungen bekannt sind, kann durch den Einsatz einer zusätzlichen Verwaltungsinstanz Prozessorkapazität reserviert und wieder freigegeben werden.

6.2 Verwaltung von Ressourcen durch DANA

In diesem Abschnitt werden zunächst Mechansimen vorgestellt, die bei DANA das Reservieren von Ressourcen ermöglichen. Anschließend wird auf ein Gesamtszenario eingegangen, bei dem aufgrund der Reservierung von Ressourcen und deren Freigabe ein Umordnen von in Java realisierten Software-Komponenten lohnenswert ist. Die Fahrzeugtelematik, die das potenzielle Einsatzgebiet von DANA bildet, dient wiederum als Beispiel.

6.2.1 Verwaltung von Ressourcen-Reservierungen durch DANA

Anhand einer genaueren Beschreibung der Verwaltung von Kanälen durch DANA wird im Folgenden das Zusammenwirken einzelner Verwaltungsinstanzen einer Konfigurationsverwaltung und Verwaltungsanteilen von Komponenten bei der Reservierung von Ressourcen vorgestellt.

Bei DANA ist eine Bindung durch einen sogenannten Kanal (eigentlich ist der Begriff Verbindung angemessener) repräsentiert, über den Daten ausgetauscht werden. Der A/V-Kanal, der in Bild 6.1 dargestellt ist, beginnt an einer Video-Kamera und endet in der Grafikverarbeitungs-

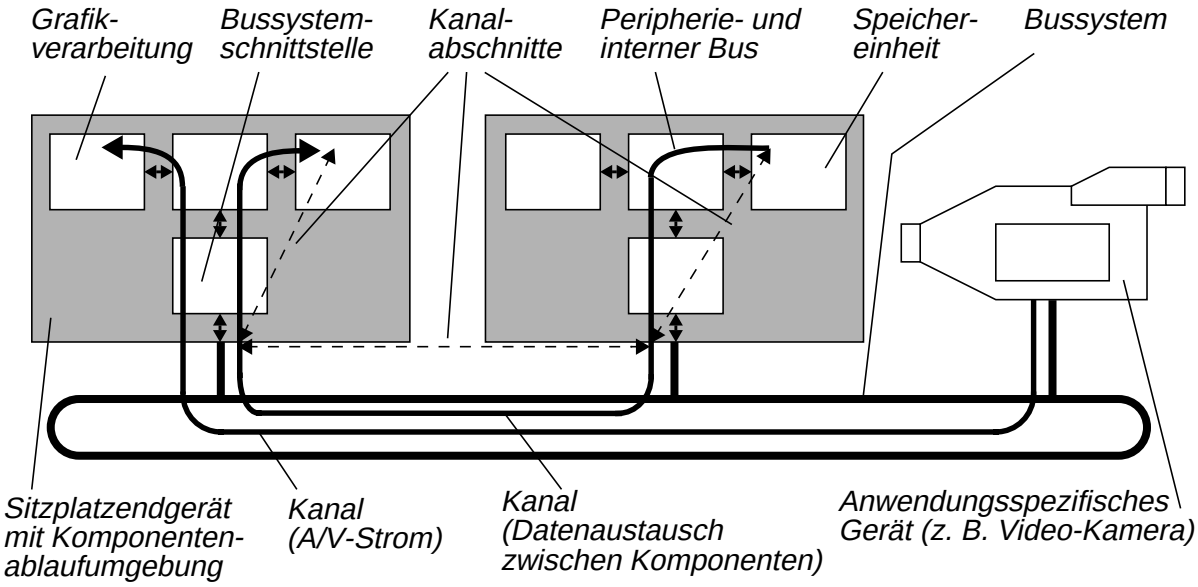


Bild 6.1: Kanalabschnitte bei DANA

einheit eines Sitzplatzendgerätes. Ein Kanal besteht aus den in Bild 6.1 eingeführten Kanalabschnitten, die entweder einem Endgerät oder einem Bussystem zugeordnet werden können.

Kanäle können zur Systemlaufzeit eingerichtet, zu Punkt-zu-Mehrpunkt-Kanälen erweitert, reduziert und wieder abgebaut werden. Die hierfür benötigte Kanalverwaltung wird durch mehrere Verwaltungsinstanzen realisiert. Wenn ein Übertragungsmedium vorhanden ist, müssen sogenannte LCMs (*Local Channel Manager*), die für die Kanalabschnitte in einem Sitzplatzendgerät verantwortlich sind, und ein sogenannter RCM (*Range Channel Manager*) für die Kanalabschnitte beim Bussystem miteinbezogen werden.

Durch Protokolle handeln zwei LCMs die Kanal-Modifikationen aus. In Bild 6.2 initiiert der LCM in Gerät A einen Vorgang zum Einrichten eines Kanals, indem `Establish`-Nachrichten mit dem LCM in Gerät B ausgetauscht werden. Bei geräteübergreifenden Kanälen wird zudem der für das Bussystem zuständige RCM durch den Austausch von `MediaAlloc`-Nachrichten miteingebunden. Er sorgt dafür, dass die für den Kanal geforderte Dienstgüte auf dem Übertragungsmedium gewährleistet wird, falls dies möglich ist.

Durch die Vorstellung einer prototypischen Implementierung [29] soll im Folgenden die Funktionsweise der DANA-Verwaltung erläutert werden. Bei dieser Implementierung wird das IEEE 1394-Bussystem verwendet. Ein RCM sorgt für die Verwaltung isochroner Ressourcen des Bussystems, da über ihn Übertragungskapazität reserviert werden kann. Wenn keine Überreservierung vorliegt, wird eine erfolgreiche Busarbitrierung zur Übertragung von Daten innerhalb eines Zyklus (ca. 125 µs) gewährleistet. Die Bussystemschnittstelleneinheit (siehe Bild 2.10) entspricht beim Prototypen einer handelsüblichen PCI-Host-Adapter-Karte.

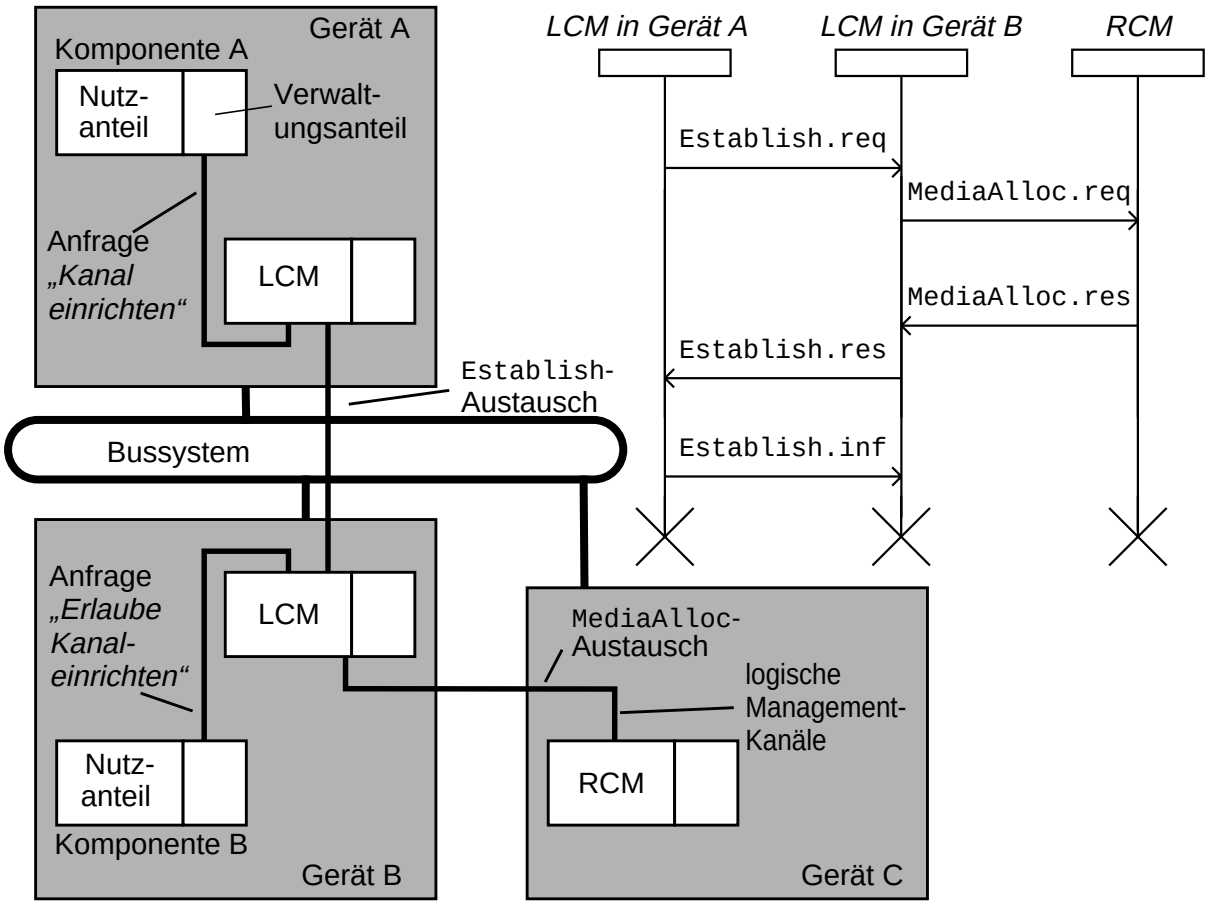


Bild 6.2: Kooperation der DANA-Verwaltungsinstanzen

- Ressourcen-Reservierung im Initiatorendgerät

  Nach der Initiierung einer Kanal-Modifikation durch eine Komponente (z. B. Anfrage „Kanal einrichten“ in Bild 6.2), versucht der zuständige LCM, die lokal im Gerät für diesen Kanal benötigten Ressourcen zu reservieren (bzw. freizugeben). Falls beispielsweise ein Kanal zum Senden von Daten mit einer garantierten Datenrate von mindestens 8 Mbit/s eingerichtet werden soll, muss der LCM u. a. ein spezielles Register in der Bussystemschnittstelleneinheit auf einen entsprechenden Wert setzen. Dort muss eingetragen werden, dass ein 128 Byte großer Rahmen im isochronen Modus je Zyklus versendet wird (128 Byte/ 125 µs > 8 Mbit/s). Beim Prototypen wird eine Unix-Prozessverwaltung in den Endgeräten eingesetzt. Es stehen deshalb dort keine synchronen Dienste zur Reservierung von Prozessorkapazität zur Verfügung.

  Eine exemplarische Realisierung eines synchronen Dienstes zur Reservierung von Prozessorkapazität ist jedoch im Umfeld dieser Arbeit durch den Aufbau eines weiteren Prototypen [71] entstanden. Er realisiert die Übertragung von CD-Audio-Strömen zwischen einem CD-Abspielgerät und einem Endgerät, auf dem eine sogenannte Jbed-JVM [104] läuft und das einen Lautsprecher in seiner Peripherie besitzt. Eine Jbed-JVM beinhaltet eine EDF-

Prozessverwaltung, in die die Java-Standard-Prozessverwaltung (siehe auch Abschnitt 3.1.3.3) so integriert ist, dass sie innerhalb eines von der EDF-Prozessverwaltung verwalteten Prozesses ohne Fertigstellungstermin abläuft. Ein CD-Audio-Strom kann mit einer konstanten Bitrate vom CD-Abspielgerät unter Nutzung eines synchronen Übertragungsdienstes z. B. über ein IEEE 1394-Bussystem übertragen und innerhalb des Endgerätes ebenfalls unter Nutzung eines synchronen Dienstes vom Bussystempuffer zur Peripherieeinheit des Lautsprechers kopiert werden. Eine Hintergrundbelastung, die durch Anforderungen für asynchrone Dienste sowohl innerhalb des Bussystems als auch innerhalb des Endgerätes zustandekommt, beeinflusst nicht die konstante Bitrate.

- Ressourcen-Reservierung für den Bus

  Nach der Reservierung dieser Ressourcen gibt der Initiator-LCM durch das Versenden einer `Establish.req`-Nachricht die Initiative an den LCM im empfangenden Gerät weiter. Der beauftragte LCM erkennt, dass die `Establish.req`-Nachricht aus einem anderen Gerät stammt und dass deshalb ein geräteübergreifender Kanal eingerichtet werden muss. Er leitet deshalb die Initiative unmittelbar an den für das Übertragungsmedium zuständigen RCM durch eine `MediaAlloc.req`-Nachricht weiter. Bei der prototypischen Implementierung stehen RCMs für Ethernet und IEEE 1394 zur Verfügung. Während der RCM für Ethernet jeden Versuch zur Übertragungskapazitätsreservierung durch eine Fehlernachricht zurückweist, sorgt der RCM für IEEE 1394 durch die Verwaltung aller Übertragungskapazitätsreservierungen dafür, dass keine Überreservierung stattfindet. Danach wird die Initiative wieder an den LCM im beauftragten Gerät zurückgegeben.

- Ressourcen-Reservierung im beauftragten Gerät

  Da der LCM im beauftragten Gerät für das Einrichten eines Kanalabschnitts mit einem Empfangsendpunkt zuständig ist, muss er der Bussystemschnittstelleneinheit in seinem Endgerät mitteilen, welcher IEEE 1394-Kanal abgehört werden soll.

Nach dem Einrichten eines Kanals kann innerhalb jeder Komponente über einen eindeutigen Bezeichner für das Kanalende (CEI, *Channel Endpoint Identifier*) auf den Kanal zugegriffen werden.

In einer im Umfeld dieser Arbeit vorgenommenen Untersuchung [30] wird dargestellt, dass eine Auflösungsstrategie für bei der Ressourcen-Reservierung entstehende Konflikte durch ein mehrdimensionales Rucksackproblem quantitativ beschrieben werden kann. Dieses klassische Optimierungsproblem kann mit Hilfe von Heuristiken zur Systemlaufzeit näherungsweise gelöst werden, womit dann eine dem Optimum nahe Konfliktauflösung vorgenommen werden kann.

6.2.2 Ressourcen-Reservierung und Rekonfigurierung

DANA beschreibt eine Architektur für komponentenbasierte Verteilte Systeme, wobei unter dem Begriff Komponente sowohl eine Software-Komponente, wie sie in Bild 6.2 beschrieben ist, als auch eine Hardware-Komponente verstanden werden kann. So kann z. B. ein Kanal auch an einem CD-Abspielgerät oder, wie in Bild 6.1 beschrieben, an einer Video-Kamera enden.

Software-Komponenten können durch bekannte Java- und IP-basierte Middleware- und Komponenten-Software-Technologien realisiert sein. DANA ermöglicht dann die integrative Verwaltung von Ressourcen-Reservierungen für den Austausch von konstantbitratigen A/V-Strömen und von Ressourcen-Anforderungen der Java-Komponenten, die mit Hilfe von IP Daten austauschen. Während für den Austausch von A/V-Strömen Ressourcen-Reservierungsinstanzen – bei DANA sind dies LCMs und RCMs – und synchrone Dienste bereitstehen, sind für die Bereitstellung von Standard-Prozessen von Java-Komponenten und den Austausch ihrer Daten, z. B. über IP-basierte Methodenfernaufrufe, lediglich asynchrone Dienste zuständig. In einem solchen Szenario ist es lohnenswert, zusätzliche Verwaltungsinstanzen einzuführen, die eine lastbezogene Rekonfigurierung durch Umordnen von ortsunabhängigen, asynchrone Dienste nutzende Software-Komponenten vornehmen, um die Systemleistung zu steigern.

Kapitel 7

Zusammenfassung und Ausblick

Es wird erwartet, dass in Zukunft komponentenbasierte Verteilte Systeme im Unternehmensbereich, im Heimbereich und in der Fahrzeugtelematikwelt eine weite Verbreitung finden werden. Wie im 2. und 3. Kapitel dargestellt ist, sind die hierfür benötigten Middleware- und Komponenten-Software-Technologien und Konzepte zur Konfigurationsverwaltung bereits heute schon vorhanden. In der vorliegenden Arbeit werden anhand der Fahrzeugtelematik die Vorteile eines Einsatzes von Komponenten-Software-Technologien benannt. So existieren in diesem Bereich verstärkte Bemühungen, durch das Verwenden von Komponenten-Software die Wiederverwendbarkeit von Software zu verbessern und den Austausch von Software während der Fahrzeuglebenszeit, die um ein Vielfaches länger als ein Software-Innovationszyklus ist, zu vereinfachen.

Ein wichtiges Ergebnis der vorliegenden Arbeit ist, dass die Systemleistung durch lastbezogene Rekonfigurierung, d. h. durch Umordnen von Komponenten zur Systemlaufzeit, verbessert werden kann. So wird im 3. Kapitel dargestellt, dass trotz des durch den Einsatz von Middleware entstehenden Mehraufwands es lohnenswert sein kann, eine Anwendung in Software-Komponenten mit Middleware-Schnittstellen zu zerlegen, um diese dann im Verteilten System günstig zu verteilen.

Um über die Systemleistung Aussagen treffen zu können, wird im 4. Kapitel eine Methode zur Modellierung von benutzergesteuerten Anwendungen, die in derartige verteilbare Software-Komponenten zerlegt sind, entwickelt. Nach dieser Methode ist eine Anwendung durch eine Kette modelliert, die durch ein Warteschlangennetz gelegt ist und in der eine Anforderung zirkuliert. Eine Komponente dieser Anwendung wird durch eine Teilkette repräsentiert. Die Komponentenablaufumgebungen werden durch Bediensysteme des Warteschlangennetzes repräsentiert. Dieses einfache Modellierungskonzept eignet sich für die Bewertung der Leistung eines komponentenbasierten Verteilten Systems mit benutzergesteuerten Anwendungen.

Aussagen über Leistungsgrößen können durch ereignisgesteuerte Simulationen oder mathematisch-analytisch durch auf Produktformansätzen beruhenden Berechnungen getroffen werden.

Die Anzahl der Rechenschritte steigt bei derartigen Berechnungen exponentiell mit der Zunahme der Anzahl von Komponenten. Die Anwendung der Core-Heuristik mildert dieses Komplexitätsproblem, da die Anzahl der Rechenschritte nur noch quadratisch mit der Zunahme von Komponenten steigt. Sie liefert für die mit der vorgestellten Modellierungsmethode hergeleiteten Warteschlangennetze zufriedenstellende Ergebnisse.

In der vorliegenden Arbeit wird gezeigt, dass die Vorgänge beim Umordnen von Komponenten und insbesondere die hierbei zu berücksichtigenden Synchronisationsaspekte ebenfalls modelliert werden können. Es wird vorgestellt, wie eine Modellierung mit Hilfe Stochastischer Petri-Netze vorgenommen werden kann. Eine mathematisch-analytische Berechnung von Leistungsgrößen ist jedoch in diesem Umfeld zumeist nur über das Lösen eines komplexen LGS möglich, da die Voraussetzungen für die Gültigkeit von Produktformlösungen nur selten erfüllt sind.

Im 5. Kapitel werden mit Hilfe der warteschlangennetzbasierten Modellierungsmethode dynamische Lastverteilungsverfahren auf ihre Eignung für den Einsatz bei lastbezogenen Rekonfigurierungen untersucht. Ein Ergebnis ist, dass abhängig von der Leistungsfähigkeit des Verteilten Systems und der Art und Größe der darin vorkommenden Inhomogenitäten ein einfaches Round Robin-Verfahren, ein auslastungsbasiertes Verfahren oder auch ein komplexeres Verfahren, bei dem die Core-Heuristik selbst zur Laufzeit angewendet wird, für den Einsatz in Frage kommen.

Es ist zu erwähnen, dass die in dieser Arbeit gewonnenen Ergebnisse nicht für allgemeine komponentenbasierte Verteilte Systeme gelten. Es werden vielmehr nur Verteilte Systeme mit Client- und Server-Komponenten, die sich zu benutzergesteuerten Anwendungen zusammensetzen lassen, untersucht.

Zudem wird der Begriff Benutzersteuerung in dem Sinne eingeschränkt verwendet, dass jede Anwendung nur einen vom Benutzer gelenkten Kontrollfluss beinhaltet. Es wird dargestellt, dass menübasierte Anwendungen, zu denen auch klassische WWW-Browser gehören, solchen benutzergesteuerten Anwendungen entsprechen können. Bei anderen Anwendungen mit Benutzerinteraktionen, wie z. B. bei interaktiven Spielen, trifft diese Annahme jedoch nicht mehr im vollen Umfang zu. Mit der Frage, inwieweit sich die gefundenen Ergebnisse verallgemeinern lassen, bzw. wo die Grenzen ihrer Anwendbarkeit liegen, ist sicherlich das Potenzial für weitere Forschungsarbeiten gegeben.

Es bleibt zudem noch die Frage zu klären, ob bzw. wo eine Verwaltung zur dynamischen Rekonfigurierung zukünftig lohnenswert eingesetzt werden kann. Wie im 6. Kapitel vorgestellt wird, ist ihr Einsatz vor allem bei sich dynamisch stark ändernden Systemen von Vorteil. Bei

der dort dargestellten Ausprägung kann es zu Schwankungen der Prozessorkapazität, die für asynchrone Dienste bereitgestellt wird, durch Ressourcen-Reservierungen für A/V-Ströme kommen.

Die bisher für die Fahrzeugtelematik spezifizierten Architekturen sind bezüglich des Umfangs der vorgesehenen Konfigurationsverwaltungen noch unvollständig und nur ein erster Schritt, dem weitere Schritte folgen müssen. Es gibt wie in dieser Arbeit an vielen Stellen erwähnt wird, einige Forschungsbeiträge, die beispielsweise Betrachtungen zur Konsistenzsicherung, zur Verwaltung von Ressourcen für Mehrplatzsysteme und über Sicherheitsaspekte beinhalten. Eine Integration der dort erarbeiteten Ergebnisse in bestehende Architekturen wird erwartet. Lastbezogene Rekonfigurierung zur Verbesserung der Systemleistung kann dann durchaus einen Platz innerhalb einer zukünftigen Konfigurationsverwaltung einnehmen.

Literaturverzeichnis

[1] G. BALBO, S.C. BRUELL, M. SERENO: “Product form solution for generalized stochastic petri nets“, *IEEE Transaction on Software Engineering*, vol. 25, no. 10, October 2002, pp. 915-931.

[2] F. BASKET, K.M. CHANDY, R.R. MUNTZ, F. PALACIOS: “Open, closed and mixed networks for queues with different classes for customers“, *Journal of the ACM*, no. 22, 1975, pp. 248-260.

[3] F. BAUSE, P. BUCHHOLZ: “Queueing petri nets with product form solution“, *Performance Evaluation*, vol. 32, 1998, pp. 268-299.

[4] T. BERGER, P. BORT, D. JOHN: „Verteilte Systeme im Kraftfahrzeug“, *Informationstechnik und Technische Informatik (it+ti)*, Bd. 41, Nr. 5, Oktober 1999, S. 7-11.

[5] C. BIDAN, V. ISSARNY, T. SARIDAKIS, A. ZARRAS: “A dynamic reconfiguration service for CORBA“, *Proceedings of the 4th International Conference on Configurable Distributed Systems*, Maryland, USA, 1998, pp. 35-42.

[6] J. BISCHOFF, E. COCHLOVIOUS. M. TIEN TRAN: „Flexible Software-Architekturen für Embedded Multimediasysteme“, *Elektronik Automotive*, Juni 2002, S. 63-69.

[7] L. BURGSTAHLER, M. NEUBAUER: „New modifications of the exponential moving average algorithm for bandwidth estimation“, *Proceedings of the 15th ITC Specialist Seminar on Internet Traffic Engineering and Traffic Management*, Würzburg, 2002, pp. 210-219.

[8] BLUETOOTH: *Specification of the bluetooth system*, Version 1.1, Bluetooth, February 2001.

[9] BMW, DAIMLERCHRYSLER: „FlexRay für verteilte Anwendungen im Fahrzeug“, *Elektronik Automotive*, Mai 2001.

[10] G. BOLCH, I.F. AKYILDIZ: *Analyse von Rechensystemen*, B.G. Teubner, Stuttgart, Deutschland, 1982, ISBN 3-519-0359-8.

[11] J. BRADLEY CHEN, Y. ENDO, K. CHAN, D. MAZIERES, A. DIAS, M. SELTZER, M.D. SMITH: “The measured performance of personal computer operating systems“, *ACM Transactions on Computer Systems*, Vol. 14, No. 1, February 1996, pp. 3-40.

[12] K. BUBENDORFER, J.H. HINE: *A compositional classification for load-balancing algorithms*, Technical Report CS-TR-99-9, Victoria University of Wellington, New Zealand, July 1998.

[13] T.L. CASAVANT, J.G. KUHL: “Effects of response and stability on scheduling in distributed computing systems“, *IEEE Transactions on Software Engineering*, vol. 14, no. 11, November 1988, pp. 1578-1588.

[14] T.L. CASAVANT, J.G. KUHL: “A taxonomy of scheduling in general-purpose distributed computing systems“, *IEEE Transactions on Software Engineering*, vol. 14, no. 2, February 1998, pp. 141-154.

[15] K.M. CHANDY, D. NEUSE: “Linearizer: A heuristic algorithm for queuing network models of computing systems“, *Communications of the ACM*, vol. 25, no. 2, February 1982, pp. 126-134.

[16] X. CHEN: “Extending RMI to support dynamic reconfiguration of distributed systems“, *Proceedings of the 22nd International Conference on Distributed Computing Systems*, ICDCS 2002, Vienna, Austria, July 2002.

[17] X. CHEN, M. SIMONS: “A component framework for dynamic reconfiguration of distributed systems“, *Proceedings of the International IFIP/ACM Working Conference on Component Deployment*, Berlin, Germany, June 2002.

[18] C. CIOCAN: “The domestic digital bus system (D2B)“, *IEEE Transactions on Consumer Electronics*, vol. 36, no. 3, 1990, pp. 619-622.

[19] P. CLIP: “Java’s object-oriented communications“, *Byte*, McGraw-Hill, February 1998, pp.53-54

[20] P. CLIP: “DCOM: Microsoft enhances DCE“, *Byte*, McGraw-Hill, March 1998, pp. 47-48.

[21] J.L. COLEMAN, W. HENDERSON, P.G. TAYLOR: “Product form equilibrium distributions and a convolution algorithm for stochastic petri nets“, *Performance Evaluation*, vol. 26, 1996, pp. 159-180.

[22] D.E. COMER: *Internetworking with TCP/IP, Vol I: Principles, protocols, and architecture, Chapter 20: The socket interface*, Third Edition, Prentice-Hall, New Jersey, USA, 1995, ISBN 0-13-227836-7.

[23] D.L. EAGER, E.D. LAZOWSKA, J. ZAHORJAN: “Adaptive load sharing in homogeneous distributed systems“, *IEEE Transactions on Software Engineering*, vol. 12, no. 5, May 1986, pp. 662-675.

[24] U.W. EISENECKER: „Das generative Paradigma oder was kommt nach der Objektorientierung?“, *OBJEKT-Spektrum*, Nr. 6, November 1996, S. 30-34.

[25] H. ENGESSER, V. CLAUS, A. SCHWILL: *Duden Informatik*, 2. Auflage, Dudenverlag, Mannheim, Deutschland, 1993, ISBN 3-411-05232-5.

[26] P. ERHARD, H. SEILER: „Ganzheitliches Diagnosekonzept - Diagnosesoftware-Erstellung von der Steuergeräte-Entwicklung bis hin zur Werkstatt“, *Beiträge zur Tagung Elektronik im Kraftfahrzeug*, VDI Berichte 1152, Baden-Baden, Deutschland, September 1994, S. 341-353.

[27] L.M. FEENEY, B. AHLGREN, A. WESTERLUND: “Spontaneous networking: An application oriented approach to ad hoc networking“, *IEEE Communications Magazine*, vol. 39, June 2001, pp. 176 -181.

[28] V. FEIL, U. GEMKOW, M. STÜMPFLE: “A vehicular software architecture enabling dynamic alterability of services sets“, *Proceedings of the 3th International IFIP/GI Working Conference towords a universal services market (USM 2000)*, LNCS 1890, Munich, Germany, 2000, pp. 68-80.

[29] V. FEIL, M. STÜMPFLE: „Management-Aufgaben bei komponentenbasierten verteilten Systemen im Fahrzeug-Telematikbereich“, *Beiträge zur 12. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS 2001)*, Hamburg, Deutschland, 2001, S. 403-414.

[30] V. FEIL: “Resource management for constant bit rate streams in component-based distributed systems“, *Proceedings of the 20th IASTED International Conference on Applied Informatics*, Symposium on Parallel and Distributed Computing and Networks, Innsbruck, Austria, 2002, pp. 149-154.

[31] H. FENNEL, S. STÖLZL: „Software-Funktionen – Auf dem Weg zu eigenständigen Produkten im Automobil”, *Automotive Electronics*, Nr. 1, 2003, S. 16-26.

[32] V. FINEBERG: “*A practical architecture for implementing end-to-end QoS in an IP network*“, IEEE Communications Magazine, January 2002, pp. 122-129.

[33] T. FLIK, H. LIEBIG: *16-Bit-Mikroprozessorsysteme: Aufbau, Arbeitsweise, Programmierung*, Springer-Verlag, Berlin, Deutschland, 1982, ISBN 3-540-11469-6.

[34] H. FOSSA: *Interactive configuration management for distributed systems*, PHD Thesis, University of London, Great Britain, 1997.

[35] A. FUGETTA G.P. PICCO, G. VIGNA: “Understanding code mobility“, *IEEE Transactions on Software Engineering*, vol. 24, no. 5, May 1998, pp. 342-361.

[36] S. FÜNFROCKEN: “Transparent migration of java-based mobile agents“, *Proceedings of the 2nd International Workshop on Mobile Agents (MA '98)*, Stuttgart, Germany, September 1998.

[37] E. GAMMA, R. HELM, R. JOHNSON, J. VLISSIDES: *Design patterns: Elements of reusable object-oriented software*, Addison-Wesley, New York, USA, 1995, ISBN 0-201-63361-2.

[38] D. GERLACH: *Konzeption und Implementierung einer Messumgebung für CORBA-basierte, verteilte Objektsysteme*, Diplomarbeit Nr. 1588, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, Februar 1999.

[39] P. GÖHNER: „Komponentenbasierte Entwicklung von Automatisierungssystemen“, *VDI/VDE GMA-Kongress*, Ludwigsburg, Deutschland, 1998.

[40] Y.Y. GOLAND, T. CAI, P. LEACH, Y. GU: *Simple service discovery protocol 1.0, operating without an arbiter*, Internet Draft <draft-cai-ssdp-v1-03.txt>, IETF, October 1999.

[41] H. GOMAA, D.A. MENASCE: “Design and performance modeling of component interconnection patterns for distributed software architectures“, *Proceedings of the 2nd International Workshop on Software and Performance*, Ottawa, Canada, 2000.

[42] W.J. GORDON, G.F. NEWELL: “Closed queueing systems with exponential servers“, *Operations Research*, no. 15, 1967, pp. 254-265.

[43] J. GOSLING, B. JOY, G. STEELE, G. BRACHA: *The java language specification*, Second Edition, Sun Microsystems, Addison-Wesley, New York, USA, 2000, ISBN 0-201-31008-2.

[44] E. GUTTMAN, C. PERKINS, J. VEIZADES, M. DAY: *Service Location Protocol*, Version 2, IETF, RFC 2608, June 1999.

[45] M. HAUB, R. TILGNER, G. HAHLGANß: „Fahrerinformation und Verkerstelematik - Ein ganzheitliches Zukunftskonzept des Mannesmann-Konzerns“, *Beiträge zur Tagung Elektronik im Kraftfahrzeug*, VDI Berichte 1152, Baden-Baden, Deutschland, September 1994, S. 565-575.

[46] HAVI: *The HAVi specification*, Version 1.1, Home Audio Video Interoperability-Gremium, May 2001.

[47] W.D. HELLER, H. LINDENBERG, M. NUSKE, K.-H. SCHRIEVER: *Stochastische Systeme, Markoffketten, Stochastische Prozesse, Warteschlangen*, de Gruyter, Berlin, Deutschland, 1978, ISBN 3-11-007624-1.

[48] W. HENDERSON, P.G. TAYLOR: “Product form in networks of queues with batch arrivals and batch services“, *Queueing Systems: Theory and Applications*, vol. 6, November 1990, pp. 71-88.

[49] W. HENDERSON, P.G. TAYLOR: “Embedded processes in stochastic petri nets“, *IEEE Transactions on Software Engineering*, vol. 17, no. 2, February 1991, pp. 108-116.

[50] R.G. HERRTWICH, G. HOMMEL: *Kooperation und Konkurrenz: Nebenläufige, verteilte und echtzeitabhängige Programmsysteme*, Springer-Verlag, Berlin, Deutschland, 1989, ISBN 3-540-51701-4.

[51] R. HINZ: „Verteilte Kommunikation in der Praxis: Ein neuer Verkehrstelematik-Dienst Parken & Leiten“, *Beiträge zur 12. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS 2001)*, Hamburg, Deutschland, 2001, S. 259-264.

[52] C.A.R. HOARE : “Monitors: An operating system structuring concept“, *Communications of the ACM*, vol. 17, no. 10, 1974, pp. 549-557.

[53] C.R. HOFMEISTER: *Dynamic reconfiguration of distributed applications*, PHD Thesis, University of Maryland, Department of Computer Science, USA, 1993.

[54] J. HOPKINS: “Component Primer“, *Communications of the ACM*, vol. 43, no. 10, 2000, pp. 27-30.

[55] R. HUDI M. BARTL, M. TAPPE: „Infotainment-Systeme im Fahrzeug“, *Elektronik Automotive*, September 2001, S. 42-48.

[56] IEEE: *IEEE standard for a high performance serial bus*, IEEE Std 1394-1995, IEEE, New York, USA, 1996, ISBN 1-55937-583-3.

[57] ITG: *Begiffswerk des ITG-Fachausschusses 5.2*, Informationstechnische Gesellschaft im VDE, 1997.

[58] ISO: *Information technology – Open systems interconnection – Basic reference model: The basic model*, ISO/IEC 7948-1, International Organization of Standardization, 1994.

[59] S. JAGANNATHAN, R. KESLEY: “On the interaction between mobile processes and objects“, *Proceedings of the 7th Heterogenous Computing Workshop*, Orlando, USA, March 1998.

[60] A. JAMEEL, M. STÜMPFLE, D. JIANG, A. FUCHS: “Web on wheels: Toward internet-enabled cars“, *Computer*, IEEE, vol. 31, 1998, pp. 69-71.

[61] P. KÄHKIPURO: *Performance modeling framework for CORBA based distributed systems*, PHD Thesis, University of Helsinki, Department of Computer Science, Finland, 2000.

[62] F.P. KELLY: *Reversibility and stochastic networks*, Wiley series in probability and mathematical statistics, John Wiley & Sons, USA, 1979, ISBN 0-471-27601-4.

[63] N. KLUßMAUL: *Lexikon der Kommunikations- und Informationstechnik*, 3. Auflage, Hüthig Verlag, Heidelberg, Deutschland, 2000, ISBN 3-7785-3913-2.

[64] J.A. KÖGEL: *Erweiterung einer Java-Ausführungsumgebung (JVM) um eine Software zum Aufzeichnen des zeitlichen Verhaltens von Java-Threads*, Studienarbeit, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, Juli 2002.

[65] P. KOGUT, P. CLEMENTS: “Features of architecture description languages“, *Proceedings of the 7th Annual Software Technology Conference*, Salt Lake City, USA, 1995.

[66] F. KON: *Automatic configuration of component-based distributed systems*, PHD Thesis, University of Illinois at Urbana-Champaign, USA, 2000.

[67] F. KON, R.H. CAMPBELL: “Dependence management in component-based distributed systems“, *IEEE Concurrency*, vol. 8, no. 1, January 2000, pp. 26-36.

[68] P. KÖNIG, C. THIEL: „Media Oriented Systems Transport“, *Informationstechnik und Technische Informatik (it+ti)*, Nr. 5, 1999, S. 36-42.

[69] H. KOPETZ, G. GRÜNSTEIDL: “TTP - A protocol for fault-tolerant real-time systems“, *Computer*, IEEE, vol. 27, no. 1, January 1994, pp. 14-23.

[70] H. KOPETZ: “A comparison of CAN and TTP“, *Proceedings of the IFAC Distributed Computer Systems Workshop*, Como, Italy, 1998.

[71] F. KRÄMER: *Untersuchung einer echtzeitfähigen, Java-basierten Hardware-Plattform*, Studienarbeit Nr. 1708, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, Juli 2001.

[72] J. KRAMER, J. MAGEE, A. FINKELSTEIN: “A constructive approach to the design of distributed systems“, *Proceedings of the 10th International Conference on Distributed Systems (ICDCS)*, Paris, France, 1990.

[73] J. KRAMER, J. MAGEE: “The evolving philophers problem: Dynamic change management“, *IEEE Transactions on Software Engineering*, vol. 16, no. 11, November 1990, pp. 1293-1306.

[74] G.E. KRASNER, S.T. POPE: “A cookbook for using the Model-View-Controller user interface paradigm in smalltalk 80“, *Journal of Object-Oriented Programming*, August 1988.

[75] P.J. KÜHN: *Analyse zufallsabhängiger Prozesse in Systemen zur Nachrichtenvermitlung und Nachrichtenverarbeitung*, 30. Bericht über verkehrstheoretische Arbeiten, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1981.

[76] P.J. KÜHN: *Technische Informatik II*, Manuskript zur Vorlesung, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 2001.

[77] P.J. KÜHN: *Informatikpraktikum*, Manuskript zum Informatikpraktikum, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 2001/2002.

[78] P.J. KÜHN: *Teletraffic theorie and engineering*, Manuskript zur Vorlesung, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 2002.

[79] B. KURZ: *Modellierung und Implementierung von Security-Diensten für eine Fahrzeug-Telematikplattform*, Diplomarbeit Nr. 1660, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, Januar 2000.

[80] S.S. LAM, Y.L. LIEN: “A tree convolution algorithm for the solution of queuing networks“, *Communications of the ACM*, vol. 26, no. 3, March 1983, pp. 204-217.

[81] S.S. LAVENBURG: *Computer performance modeling handbook*, Academic Press, New York, USA, 1983, ISBN 0-12-438720-9.

[82] M. LITOIU, H. KHAFAGY, B. QIN, A.R. WAN, J. ROLIA: “A performance engineering tool and method for distributed applications“, *Proceedings of the Center for Advanced Studies Conference (CASCON 97)*, Toronto, Cananda, November 1997.

[83] V.M. LO: “Heuristic algorithms for task assignment in distributed systems“, *IEEE Transaction on Computers*, vol. 37, no. 11, November 1988, pp. 1384-1397.

[84] K. LOUREIRO, M. BLECHAR: *Componentware: Categorisation and cataloging*, Applications development and management strategies research note, Gartner Group, December 1997.

[85] C. MÄRZ: „Der Lotse der Zukunft: Auf dem Weg von der Navigation zum vernetzten Informationssystem“, *Elektronik Automotive*, Juni 2002, S. 58-62.

[86] B. MEYER: “On to components“, *Computer*, IEEE, vol. 32, no. 1, January 1999, pp. 139-140.

[87] B. MEYER: “A really good idea“, *Computer*, IEEE, vol. 32, no. 12, December 2000, pp. 144-147.

[88] M.K. MOLLOY: “Performance analysis using stochastic petri nets“, *IEEE Transactions on Computers*, vol. c-31, no. 9, September 1982, pp. 913-917.

[89] P. NAUR, B. RANDELL: “Software engineering“, *Report of a conference sponsored by the NATO Science Committee*, Garmisch , Germany, October 1968.

[90] K.J. NEUMANN, U. KIENCKE, A. MAISCH: „Ein aufkommender Standard für verteilte Systeme im Kfz“, *Automatisierungstechnische Praxis*, Nr. 4, 1998, S. 22-31.

[91] P. NIEMEYER, J. PECK: *Exploring Java*, O’Reilly, USA, 1997, ISBN 3-930673-52-5.

[92] OMG: *Common object request broker: architecture and specification*, Version 2.6.1, Object Management Group, May 2002.

[93] OMG: *CORBA 3.0 new components chapters*, Object Management Group, November 2001.

[94] OMG: *Event service specification*, Version 1.1, Object Management Group, March 2001.

[95] OMG: *Load balancing and monitoring of CORBA based applications*, Request for proposal, Object Management Group, August 2001.

[96] OMG: *Naming service specification, Version 1.1*, Object Management Group, February 2001.

[97] OMG: *Notification service specification*, Version 1.0, Object Management Group, June 2000.

[98] OMG: *Security service specification*, Version 1.8, Object Management Group, March 2002.

[99] OMG: *Unified Modeling Language Specification*, Version 1.4, Object Management Group, September 2001.

[100] P. OREIZY, R.N. TAYLOR: “On the role of software architectures in runtime system reconfiguration“, *Proceedings of the International Conference of Configurable Distributed Systems (ICCDS)*, Annapolis, USA, 1998.

[101] OSGi: *OSGi service gateway specification*, Release 1.0, Open Services Gateway Initiative, May 2000.

[102] K. Panzer, G. Müller, H. Hurt, C. Thiel: „Von D2B zu MOST – Übertragung mit Licht: Kfz-Multimedia-Infrastruktur“, *Components*, Siemens, Nr. 1, Januar 1999, S. 23-25.

[103] C. Pfister, C. Szyperski: “Why objects are not enough“, *Proceedings of the 1st International Component Users Conference (CUC '96)*, Munich, Germany, 1996.

[104] M. Pilz: “Earliest deadline first scheduling for real-time java“, *Proceedings of the Embedded Systems Conference Europe*, Stuttgart, Germany, 2000.

[105] C. Popien, G. Schürmann, K.-H. Weiß: *Verteilte Verarbeitung in Offenen Systemen*, B.G. Teubner, Stuttgart, Deutschland, 1996, ISBN 3-519-02142-0.

[106] K. Ramamritham, J.A. Stankovic, W. Zhao: “Distributed scheduling of tasks with deadlines and resource requirements“, *IEEE Transactions on Computers*, vol. 38, no. 5, August 1989, pp. 1110-1123.

[107] M. Reiser, S.S. Lavenberg: “Mean value analysis of closed multichain queueing networks“, *Journal of the ACM*, vol. 27, no. 2, April 1980, pp. 313-322.

[108] Robert Bosch: *CAN Specification*, Version 2.0, Robert Bosch GmbH, 1991.

[109] K. Rothermel, M. Schwehm, A. Kent (Eds), J. Williams (Eds): “Mobile agents“, *Encyclopedia of computer science and technology*, Marcel Dekker, New York, USA, 1998, ISBN 0-8247-2298-1.

[110] Salutation: *Salutation architecture specification*, Version 2.0c, The Salutation Consortium, June 2001.

[111] J. Schoof: „Der OSEK/VDX-Standard“, *Technologie Forum Embedded Software*, In-Car Computing, 3Soft GmbH, Erlangen, Deutschland, 1999, S. 7-19.

[112] U. Schrey: „Software als Produkt”, *Automotive Electronics*, Nr. 1, 2003, S. 8-15.

[113] P. Schweitzer: “Approximate analysis of multichain closed networks of queues“, *Proceedings of the International Conference of Stochastic Control and Optimization*, Amsterdam, Nederlands, 1979.

[114] H. Schwichtenberg: „Mit MTS zu COM+: Komponentendienste in NT 4 und Windows 2000“, *iX*, Nr. 2, 2000, S. 124-129.

[115] U. Seimet: „CORBA übernehmen Sie – Services: Basisdienste für den ORB“, *iX*, Nr. 1, Januar 1999.

[116] M. Sereno, G. Balbo: “Mean value analysis of stochastic petri nets“, *Performance Evaluation*, vol. 29, no. 1, 1997, pp. 35-62.

[117] R. Serfozo: “Markovian network processes: congestion-dependent routing and processing“, *Queueing Systems: Theory and Applications*, vol. 5, November 1989, pp.5-36.

[118] F. SHEIKH, J. ROLIA, P. GARG, S. FROLUND, A. SHEPPARD: “Layered performance modelling of a CORBA-based distributed application design“, *Proceedings of the 4th International Conference on Analytical and Numerical Modelling Techniques with Application to QoS Modelling*, Singapore, 1997.

[119] N.G. SHIVARATRI, P. KRUEGER, M. SINGHAL: “Load distributing for locally distributed systems“, *Computer*, IEEE, vol. 25, no. 12, December 1992, pp. 33-44.

[120] SIEMENS VDO: *Siemens VDO Automotive führt dynamisch erweiterbare Mulimedia-Software-Plattform ein*, Pressemitteilung, Informationsnummer SV200209.018d, `www.siemensvdo.com`, 2002.

[121] G. SIGLE: „Multimedia-Verbindung in bewegte Kraftfahrzeuge“, *Beiträge zur 7. Internationalen Fachtagung Elektronik im Kraftfahrzeug*, VDI Verlag, Baden-Baden, Deutschland, September 1996, S. 373-380.

[122] B. SOSTAWA, T. DANNEMANN, J. SPEIDEL: “DSB-Based transcoding of digital video signals with MPEG-2 format“, *IEEE Transactions on Consumer Electronics*, vol. 46, no. 2, 2000, pp. 358-362.

[123] S. STANCHINA: *Access Control innerhalb dynamisch konfigurierbarer Software-Komponentensysteme im Fahrzeug*, Diplomarbeit Nr. 1727, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, Juni 2001.

[124] J.A. STANOVIC: “Stability and distributed scheduling algorithms“, *IEEE Transactions on Software Engineering*, vol. 11, no. 10, October 1985, pp. 1141-1152.

[125] H.S. STONE: “Multiprocessor scheduling with the aid of network flow algorithms“, I*EEE Transactions on Software Engineering*, vol. 3, January 1977, pp. 85-93.

[126] K.D. STRAUß: „Die Bedeutung von Datennetzwerken für das Kraftfahrzeug“, *Beiträge zur Tagung Elektronik im Kraftfahrzeug*, Bd. Elektronik im Kraftfahrzeug, VDI Berichte 1009, Baden-Baden, Deutschland, September 1992, S. 229-243.

[127] M. STÜMPFLE, M. EBERSPÄCHER, H. KOCHER: „Design und Realisierung eines offenen Fahrzeug-Kommunikationssystems“, *Informationstechnik und Technische Informatik (it+ti)*, Nr. 1, 1998, S. 26-33.

[128] M. STÜMPFLE, F. HERMES, V. FRIESEN, X. CHEN, S. BACHHOFER, D. JIANG, K. LY, C. GAUGER, P. STIESS: “COSIMA – A component system information and management architecture“, *Proceedings of the IEEE Intelligent Vehicles Symposium*, Dearborn, USA, 2000.

[129] SUN: *Java servlet specification*, Version 2.3, Sun Microsystems, August 2001.

[130] SUN: *Enterprise java beans specification*, Version 2.1, Sun Microsystems, June 2002.

[131] C. SZYPERSKI: *Component Software: Beyond object-oriented programming*, Addison-Wesley, New York, USA, 1997, ISBN 0-201-17888-5.

[132] D. TEICHNER: „Netzwerk-Konzepte für Video- und Audiofunktionen im Auto“, *Fernseh- und Kinotechnik*, Hüthig-Verlag, Bd. 54, Nr. 3, 2000.

[133] J. TRYGGVESSON, T. MATTSON, H. HEEB: “JBED: Java for real-time systems“, *Dr. Dobb’s Journal*, Miller Freeman, USA, 1999.

[134] S. VIEHWEG: „Neuer Standard für die Verkehrstelematik“, *Funkschau*, September 1998, S. 44-46.

[135] S. WALDENMEIER: „Multimedia im Auto der Zukunft“, *Funkschau*, August 2001, S. 36-38.

[136] J. WALDO: *The jini specifications*, Second Edition, Sun Microsystems, Addison-Wesley, New York, USA, 2000, ISBN 0-201-72617-3.

[137] G. WILLMANN: *Lösungsalgorithmen für Warteschlangennetze mit Produktlösungsform*, Diplomarbeit D II, Nr. 38, Elektrotechnik I, F.G. Nachrichtentechnik, Universität Gesamthochschule Siegen, Juni 1983.

[138] D. WRIGHT: *Broadband: Business services, technologies and stratetic impact*, Artech House, Boston, USA, 1993, ISBN 0-89006-589-6.

[139] W3C: *Simple object access protocol (SOAP)*, Version1.1, World Wide Web Consortium, 2000.

[140] W3C: *Web services description language (WSDL)*, Version 1.1, World Wide Web Consortium, 2001.

WWW-Adressen

[141] `www.netcluesoft.com/`

[142] `ninja.cs.berkeley.edu/`

[143] `www.kaffe.org/`

Anhang A

Veranschaulichung der Berechnungen von Leistungsgrößen in GPWNs

Im Folgenden werden die in der Arbeit angewendeten Algorithmen für Produktformlösungen zur Berechnung von Leistungsgrößen in Geschlossenen Warteschlangennetzen exemplarisch veranschaulicht. Zunächst werden die im 4. Kapitel vorgestellten Algorithmen zur Berechnung von exakten Lösungen betrachtet. Hierzu zählen BCMP [2], die Algorithmen nach Kelly [62] und MVA [107]. Schließlich wird die in dieser Arbeit in Abschnitt 5.2.2.2 vorgestellte MCH (*modifizierte Core-Heuristik*) anhand eines Beispiels veranschaulicht.

A.1 Algorithmen zur Berechnung von exakten Lösungen

Das in Bild A.1 dargestellte Geschlossene Warteschlangennetz bildet die Grundlage für die folgenden Berechnungen. Zwei Ketten mit je einer zirkulierenden Anforderung repräsentieren jeweils eine Anwendung. Die Produktform ist anwendbar, da in den Knoten 3 und 4 die Verar-

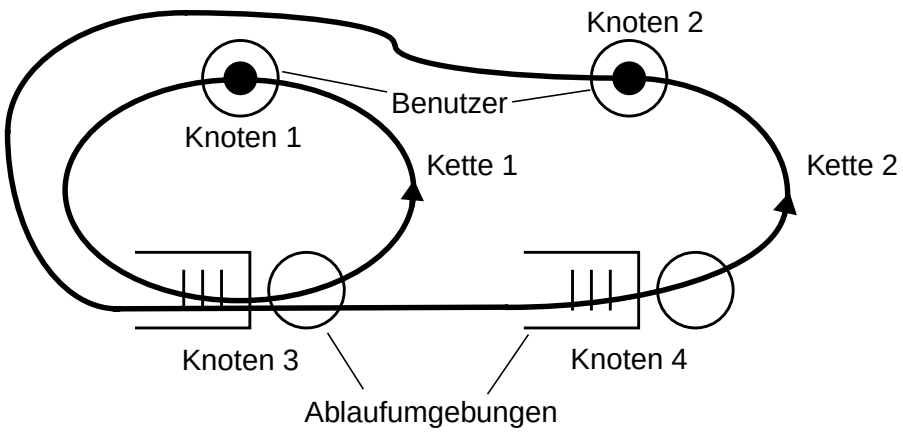


Bild A.1: Ein exemplarisches Geschlossenes Warteschlangennetz

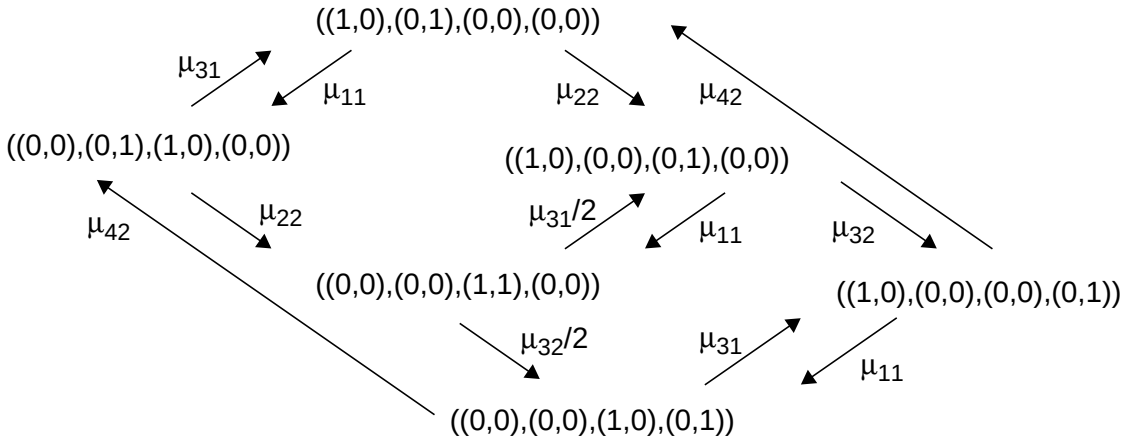


Bild A.2: Zustandsraum für das exemplarische Geschlossene Warteschlangennetz

beitungsdisziplin Processor Sharing angewendet wird und die Knoten 1 und 2 jede Anforderung ohne Verzögerung bedienen.

A.1.1 Berechnungen mit Hilfe der Zustandswahrscheinlichkeiten

In Bild A.2 ist der Zustandsraum für das exemplarische Geschlossene Warteschlangennetz dargestellt. Man kann nun ein LGS für das globale Gleichgewicht nach Gleichung (4.7) aufstellen und die Zustandswahrscheinlichkeiten berechnen. Man erhält folgende Zustandswahrscheinlichkeiten:

$$p((1,0),(0,1),(0,0),(0,0)) = \frac{1}{G(1,1)}\frac{v_1}{\mu_{11}}\frac{v_2}{\mu_{22}} = p_a$$

$$p((0,0),(0,1),(1,0),(0,0)) = \frac{1}{G(1,1)}\frac{v_1}{\mu_{31}}\frac{v_2}{\mu_{22}} = p_b$$

$$p((1,0),(0,0),(0,1),(0,0)) = \frac{1}{G(1,1)}\frac{v_1}{\mu_{11}}\frac{v_2}{\mu_{32}} = p_c$$

$$p((0,0),(0,0),(1,1),(0,0)) = \frac{2}{G(1,1)}\frac{v_1}{\mu_{31}}\frac{v_2}{\mu_{32}} = p_d$$

$$p((1,0),(0,0),(0,0),(0,1)) = \frac{1}{G(1,1)}\frac{v_1}{\mu_{11}}\frac{v_2}{\mu_{42}} = p_e$$

$$p((0,0),(0,0),(1,0),(0,1)) = \frac{1}{G(1,1)}\frac{v_1}{\mu_{31}}\frac{v_2}{\mu_{42}} = p_f$$

$$G(1,1) = \left(\frac{\nu_1\nu_2}{\mu_{11}\mu_{22}} + \frac{\nu_1\nu_2}{\mu_{31}\mu_{22}} + \frac{\nu_1\nu_2}{\mu_{11}\mu_{32}} + \frac{2\nu_1\nu_2}{\mu_{31}\mu_{32}} + \frac{\nu_1\nu_2}{\mu_{11}\mu_{42}} + \frac{\nu_1\nu_2}{\mu_{31}\mu_{42}}\right)$$

Man kann erkennen, dass diese Zustandswahrscheinlichkeiten auch über Produktformlösungen nach BCMP [2] oder Kelly [62] berechenbar sind, da die lokalen Gleichgewichtsgleichungen gelten. Es sei zur Veranschaulichung der Zustand $((1,0),(0,0),(0,1),(0,0))$ näher betrachtet. Er wird u. a. vom Zustand $((1,0),(0,1),(0,0),(0,0))$ erreicht, wenn eine Anforderung der Kette 2 in Knoten 3 eintrifft. Ein lokales Gleichgewicht stellt sich ein, wenn die Flussrate der in Knoten 3 ankommenden Anforderung(en) der Kette 2 gleich der Flussrate der von Knoten 3 abgehenden Anforderung(en) der Kette 2 ist. Tatsächlich gilt diese Beziehung mit:

$$\mu_{22}p((1,0),(0,1),(0,0),(0,0)) = \mu_{32}p((1,0),(0,0),(0,1),(0,0)) = \frac{1}{G(1,1)}\frac{\nu_1\nu_2}{\mu_{11}}$$

In Tabelle A.1 sind die Wahrscheinlichkeiten $p_i(\underline{k})$, dass ein Knoten i eine Anforderungsmenge $\underline{k}$ enthält, dargestellt.

$\underline{k}$ \ i	1	2	3	4
(0,0)	$p_b + p_d + p_f$	$p_c + p_d + p_e + p_f$	$p_a + p_e$	$p_a + p_b + p_c + p_d$
(0,1)	0	$p_a + p_b$	p_c	$p_e + p_f$
(1,0)	$p_a + p_c + p_e$	0	$p_b + p_f$	0
(1,1)	0	0	p_d	0

Tabelle A.1: $p_i(\underline{k})$

Hierdurch lassen sich nun durch die Gleichung (4.1), d. h. nach dem Gesetz von Little, die Aufenthaltsdauern von Anforderungen in den Knoten berechnen. Es gilt

$$\bar{t}_{11} = \frac{p_1(1,0)}{\mu_{11}p_1(1,0)} = \frac{1}{\mu_{11}},$$

$$\bar{t}_{31} = \frac{1}{\mu_{31}}\frac{p_3(1,0) + 2p_3(1,1)}{p_3(1,0) + p_3(1,1)} = \frac{1}{\mu_{31}}\frac{p_b + p_f + 2p_d}{p_b + p_f + p_d} = \frac{1}{\mu_{31}}\frac{\frac{1}{\mu_{22}} + \frac{2}{\mu_{32}} + \frac{1}{\mu_{42}}}{\frac{1}{\mu_{22}} + \frac{1}{\mu_{32}} + \frac{1}{\mu_{42}}},$$

$$\bar{t}_{22} = \frac{p_2(0,1)}{\mu_{22}p_2(0,1)} = \frac{1}{\mu_{22}},$$

$$\bar{t}_{32} = \frac{1}{\mu_{32}} \frac{p_3(0,1) + 2p_3(1,1)}{p_3(0,1) + p_3(1,1)} = \frac{1}{\mu_{32}} \frac{p_c + 2p_d}{p_c + p_d} = \frac{1}{\mu_{32}} \frac{\frac{1}{\mu_{11}} + \frac{2}{\mu_{31}}}{\frac{1}{\mu_{11}} + \frac{1}{\mu_{31}}} \text{ und}$$

$$\bar{t}_{42} = \frac{p_4(0,1)}{\mu_{42} p_4(0,1)} = \frac{1}{\mu_{42}}.$$

A.1.2 Anwendung der Mittelwertanalyse (MVA)

Auch über MVA lassen sich die Aufenthaltsdauern von Anforderungen in den Knoten berechnen. Es müssen die Gleichungen (4.28) bis (4.34) angewendet werden. Man erhält auf diesem Wege

$$\bar{t}_{11}(1,1) = \frac{1}{\mu_{11}}(1 + \bar{k}_{11}(0,1) + \bar{k}_{12}(0,1)) = \frac{1}{\mu_{11}}(1 + 0 + 0) = \frac{1}{\mu_{11}},$$

$$\bar{t}_{31}(1,1) = \frac{1}{\mu_{31}}(1 + \bar{k}_{31}(0,1) + \bar{k}_{32}(0,1)) = \frac{1}{\mu_{31}}\left(1 + 0 + \frac{\frac{1}{\mu_{32}}}{\frac{1}{\mu_{22}} + \frac{1}{\mu_{32}} + \frac{1}{\mu_{42}}}\right)$$

$$= \frac{1}{\mu_{31}} \frac{\frac{1}{\mu_{22}} + \frac{2}{\mu_{32}} + \frac{1}{\mu_{42}}}{\frac{1}{\mu_{22}} + \frac{1}{\mu_{32}} + \frac{1}{\mu_{42}}},$$

$$\bar{t}_{22}(1,1) = \frac{1}{\mu_{22}}(1 + \bar{k}_{21}(1,0) + \bar{k}_{22}(1,0)) = \frac{1}{\mu_{22}}(1 + 0 + 0) = \frac{1}{\mu_{22}},$$

$$\bar{t}_{32}(1,1) = \frac{1}{\mu_{32}}(1 + \bar{k}_{31}(1,0) + \bar{k}_{32}(1,0)) = \frac{1}{\mu_{32}}\left(1 + \frac{\frac{1}{\mu_{31}}}{\frac{1}{\mu_{11}} + \frac{1}{\mu_{31}}} + 0\right) = \frac{1}{\mu_{32}} \frac{\frac{1}{\mu_{11}} + \frac{2}{\mu_{31}}}{\frac{1}{\mu_{11}} + \frac{1}{\mu_{31}}}$$

und

$$\bar{t}_{42}(1,1) = \frac{1}{\mu_{42}}(1 + \bar{k}_{41}(1,0) + \bar{k}_{42}(1,0)) = \frac{1}{\mu_{42}}(1 + 0 + 0) = \frac{1}{\mu_{42}}.$$

A.2 Die modifizierte Core-Heuristik (MCH)

In diesem Abschnitt wird zunächst die Anwendung der MCH durch ein Beispiel veranschaulicht. Weiter wird die Qualität ihrer Ergebnisse für dieses Beispiel abgeschätzt, indem sie mit exakten Berechnungen verglichen werden.

In Bild A.3 ist das Geschlossene Warteschlangennetz dargestellt, das die Grundlage für die folgenden Überlegungen bildet. Die Bediensysteme, in denen sich mehr als eine Anforderung befinden können, arbeiten in diesem Beispiel wiederum mit Processor Sharing. Es sind Produktformlösungen anwendbar. Eine zu einer Anwendung r gehörende Kette ist immer exklusiv durch einen Knoten, der einen zu dieser Anwendung gehörenden Benutzer mit einer mittleren Dauer seiner Denkpause τ_r repräsentiert, und durch zwei weitere Knoten, die die Umgebungen r und $r + 1$ repräsentieren, gelegt. Es wird angenommen, dass die mittlere Bedienrate für eine Anforderung der Kette r in einem die Umgebung i repräsentierenden Knoten μ_{ir} ist.

A.2.1 Anwendung der modifizierten Core-Heuristik

Wendet man die MCH mit einer Iteration ($I = 1$) an, um z. B. die Aufenthaltsdauer der zur Kette 1 gehörenden Anforderung in Umgebung 2 zu berechnen, so erhält man nach Gleichung (5.5) die Beziehung

$$\tilde{t}_{21}(1, 1, 1) = \frac{1}{\mu_{21}}\left(1 + \frac{\frac{1}{\mu_{22}}}{\tau_2 + \frac{1}{\mu_{22}} + \frac{1}{\mu_{32}}}\right).$$

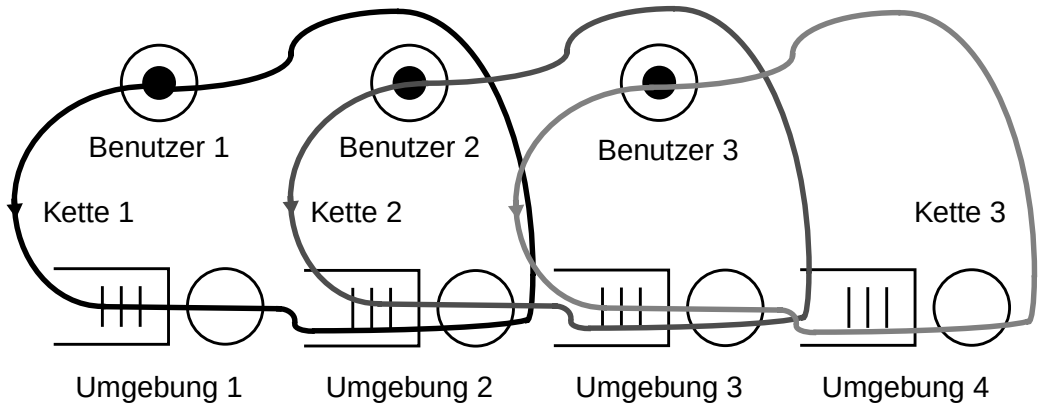


Bild A.3: Ein weiteres exemplarisches Geschlossenes Warteschlangennetz

A.2.2 Anwendung der Mittelwertanalyse (MVA)

Bei einer exakten Berechnung dieser Aufenthaltsdauer mittels MVA gelten folgende Überlegungen:

$$\bar{t}_{21}(1,1,1) = \frac{1}{\mu_{21}}(1 + \bar{k}_{21}(0,1,1) + \bar{k}_{22}(0,1,1) + \bar{k}_{23}(0,1,1))$$

$$= \frac{1}{\mu_{21}}(1 + \bar{k}_{22}(0,1,1)) = \frac{1}{\mu_{21}}\left(1 + \frac{\bar{t}_{22}(0,1,1)}{\tau_2 + \bar{t}_{22}(0,1,1) + \bar{t}_{32}(0,1,1)}\right)$$

$$= \frac{1}{\mu_{21}}\left(1 + \frac{\frac{1}{\mu_{22}}}{\tau_2 + \frac{1}{\mu_{22}} + \bar{t}_{32}(0,1,1)}\right) = \frac{1}{\mu_{21}}\left(1 + \frac{\frac{1}{\mu_{22}}}{\tau_2 + \frac{1}{\mu_{22}} + \frac{1}{\mu_{32}}(1 + \bar{k}_{32}(0,0,1))}\right)$$

$$= \frac{1}{\mu_{21}}\left(1 + \frac{\frac{1}{\mu_{22}}}{\tau_2 + \frac{1}{\mu_{22}} + \frac{1}{\mu_{32}}\left(1 + \frac{\frac{1}{\mu_{33}}}{\tau_3 + \frac{1}{\mu_{33}} + \frac{1}{\mu_{43}}}\right)}\right)$$

Man sieht, dass die Berechnungen über MCH und MVA nur dann gleich sind, d. h. dass MCH nur dann exakte Ergebnisse liefert, wenn

$$\frac{1}{\mu_{33}} = 0$$

ist.

Anhang B

Veranschaulichung der Produktform-lösung bei Stochastischen Petri-Netzen

Die Verwendung der Gleichungen für Produktformlösungen bei Stochastischen Petri-Netzen wird nun an einem in Bild B.1 gegebenen Beispiel, das aus [21] entnommen ist, veranschaulicht. Es kann ein Produktformlösung angewendet werden, da zum einen

$$\underline{M}_e(1) = \underline{M}_a(3) = (1, 0, 0, 0),$$

$$\underline{M}_e(2) = \underline{M}_a(4) = (1, 1, 0, 0),$$

$$\underline{M}_e(3) = \underline{M}_a(1) = (0, 0, 1, 0) \text{ und}$$

$$\underline{M}_e(4) = \underline{M}_a(2) = (0, 0, 0, 1)$$

gilt und zum anderen

$$\Psi(\underline{S}) = 1 \text{ und}$$

$$\Phi(\underline{S}) = 1 \text{ für alle möglichen Zustände } \underline{S} \text{ ist.}$$

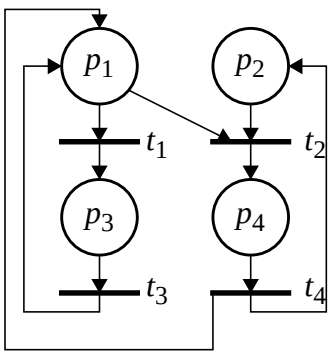


Bild B.1: Exemplarisches SPN

Daraus folgt, dass

$$\varepsilon(1) = 3, \varepsilon(2) = 4,$$

$$\varepsilon(3) = 1 \text{ und } \varepsilon(4) = 2$$

und

$$\underline{I} = \begin{bmatrix} -1 & 0 & 1 & 0 \\ -1 & -1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$$

ist. Unter Verwendung von (4.47) erhält man die beiden unabhängigen Gleichungen

$$\log(y_1) - \log(y_3) = \log\left(\frac{\mu_3}{\mu_1}\right)$$

und

$$\log(y_1) + \log(y_2) - \log(y_4) = \log\left(\frac{\mu_4}{\mu_2}\right).$$

Da nur zwei unabhängige Gleichungen für vier Unbekannte existieren, kann man willkürlich

$$y_1 = 1 \text{ und } y_2 = 1$$

setzen. Man erhält weiter

$$y_3 = \frac{\mu_1}{\mu_3} \text{ und } y_4 = \frac{\mu_2}{\mu_4}.$$

Mit den Zuständen $\underline{S} = (1, 1, 0, 0)$, $\underline{S} = (0, 1, 1, 0)$ und $\underline{S} = (0, 0, 0, 1)$ wird der Zustandsraum vollständig beschrieben. Die Zustandswahrscheinlichkeiten werden mit (4.48) berechnet und lauten

$$p(1, 1, 0, 0) = \frac{1}{G},$$

$$p(0, 1, 1, 0) = \frac{1}{G}\frac{\mu_1}{\mu_3} \text{ und}$$

$$p(0, 0, 0, 1) = \frac{1}{G}\frac{\mu_2}{\mu_4}.$$