Thomas Stöckel

Kommunikationstechnische Integration der Prozeßebene in Produktionssysteme durch Middleware-Frameworks

Thomas Stöckel

Kommunikationstechnische Integration der Prozeßebene in Produktionssysteme durch Middleware-Frameworks

Herausgegeben von
Professor Dr.-Ing. Klaus Feldmann,
Lehrstuhl für
Fertigungsautomatisierung und Produktionssystematik

FAPS

Meisenbach Verlag Bamberg

Als Dissertation genehmigt von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der Einreichung:	6. September 2000
Tag der Promotion:	22. Dezember 2000
Dekan:	Prof. Dr.-Ing. Harald Meerkamm
Berichterstatter:	Prof. Dr.-Ing. Klaus Feldmann
	Prof. Dr. rer. nat. Fridolin Hofmann

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Stöckel, Thomas:
Kommunikationstechnische Integration der Prozeßebene in Produktionssysteme durch Middleware-Frameworks / Thomas Stöckel. - Bamberg : Meisenbach, 2001
(Fertigungstechnik - Erlangen ; 107)
Zugl.: Erlangen, Nürnberg, Univ., Diss., 2000
ISBN 3-87525-143-1 ISSN 1431-6226

Herstellung: Gruner Druck GmbH, Erlangen-Eltersdorf
Printed in Germany

Vorwort und Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter von Prof. Dr.-Ing. K. Feldmann am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik an der Friedrich-Alexander-Universität Erlangen-Nürnberg. Ihm gilt mein besonderer Dank für die Ermöglichung der zugrundeliegenden Arbeiten, seine großzügige Förderung und seine Anregungen. Herrn Prof. Dr. rer. nat. F. Hofmann, dem Leiter des Lehrstuhls für Betriebssysteme, danke ich für die sehr gute Zusammenarbeit und die Übernahme des Zweitgutachtens.

Ein herzlicher Dank gilt meinen Kollegen am Lehrstuhl FAPS für die gute Zusammenarbeit in den Jahren, insbesondere meiner Kollegin Frau Dr.-Ing. Elke Rauh und meinem Kollegen Herrn Dipl.-Ing. Andreas Licha. Für die vielen hilfreichen Diskussionen und die Möglichkeit, die von mir bearbeitete Thematik auch aus einem anderen Blickwinkel betrachten zu können, danke ich den ehemaligen Kollegen aus den Teilprojekten B2, B3, B4 und D5 des Sonderforschungsbereichs SFB 182.

Ferner gilt mein Dank den Studenten und wissenschaftlichen Hilfskräften, die mich bei der Anfertigung der Arbeit unterstützt haben. Vor allem möchte ich an dieser Stelle namentlich die Herren Dipl.-Ing. Olivier Dufils, Dipl.-Inf. Peter Datsichin, cand.-inf. Bernd Haberstumpf, Dipl.-Ing. Florian Kießwetter sowie Frau Andrea Bohland erwähnen. Für die Durchsicht der Arbeit und die wertvollen Anregungen möchte ich meinen Dank den Herren Dipl.-Inf. Thomas Collisi, Dipl.-Inf. Christoph Felbinger und Dipl.-Ing. Andreas Licha aussprechen.

In privater Hinsicht gilt mein herzlichster Dank meinen Eltern, die durch ihre fortwährende und umfassende Unterstützung über alle Jahre diese Arbeit erst möglich gemacht haben.

Kommunikationstechnische Integration der Prozeßebene in Produktionssysteme durch Middleware-Frameworks

Inhaltsverzeichnis

Glossar

API	Application Programmers Interface *oder* Application Process Instance
ASI	Aktuator-Sensor-Interface
ASP	Active Server Pages
BOA	Basic Object Adapter
CAD	Comupter Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CAN	Controller Area Network
CAP	Computer Aided Planning
CAQ	Computer Aided Quality
CAx	Gesamtheit der "Computer-Aided" Technologien CAD, CAE, CAP, CAM, CAQ
CGI	Common Gateway Interface
CLQ	Champions League Quake
CLSID	Class Identifier
COM	Component Object Model
CR	Communication Reference
CSS	Cascading Style Sheet
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DD	Device Description
DDL	Device Description Language
DII	Dynamic Invocation Interface
DOM	Document Object Model
DP	Dezentrale Peripherie
DSI	Dynamic Skeleton Interface
DTD	Document Type Definition
DTM	Device Type Manager
EDD	Electronic Device Description
EJB	Enterprise JavaBean
ES	Extended Server
FDL	Fieldbus Data Link
FDT	Field Device Tool
FIMO	Fieldbus Messaging Object Request Broker
FMA	Fieldbus Management

FMS	Fieldbus Message Specification
FIP	Factory Instrumentation Protocol *(frz.: Flux Information Processus)*
FNS	FIMO Naming Service
FTP	File Transfer Protocol
GIOP	General IOP
GUID	Globally Unique Identifier
HART	Highway Addressable Remote Transducer
HMI	Human Machine Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IID	Interface Identifier
IIOP	Internet Inter-ORB Protocol
ISO	International Organization for Standardization
ISP	InterOperable Systems Project
JDBC	Java Database Connectivity
Jini	Java Intelligent Network Infrastructure
JVM	Java Virtual Machine
LAN	Local Area Network
LLC	Logical Link Control
LLI	Lower Layer Interface
LPC	Local Procedure Call
MAC	Medium Access Control
MAP	Manufacturing Automation Protocol
MMS	Manufacturing Message Specification
NC	Numeric Control (Numerische Steuerung)
OA	Object Adapter
OD	Object Dictionary
ODL	Object Description Language
OLE	Object Linking and Embedding
OPC	OLE for Process Control
ORB	Object Request Brooker
OSF	Open Software Foundation
OSI	Open Systems Interconnection
P3P	Platform for Privacy Preferences
PA	Prozeßautomatisierung

PC	Personal Computer
PDU	Protocol Data Unit
PICS	Platform for Internet Content Selection
PIDL	Pseudo Interface Definition Language
POA	Portable Object Adapter
PROFIBUS	Process Fieldbus
PPS	Produktionsplanung und -steuerung
RC	Robot Control
RDF	Resource Description Framework
RFP	Request for Proposal
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SDS	Smart Distributed System
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SPS	Speicherprogrammierbare Steuerungen
SSL	Secure Socket Layer
TOP	Technical and Office Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VFD	Virtual Field Device
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XML-Transformations

1 Einleitung

Die Bewältigung komplexer Aufgaben im globalen Innovationswettbewerb produzierender Unternehmen erfordert die möglichst optimale Kombination weltweit verteilter Produktionsfaktoren. Durch die arbeitsteilige Organisation und zunehmende Modularisierung heutiger Unternehmen sowie durch die Ausprägung neuer Formen von Unternehmensnetzwerken hängt der wirtschaftliche Erfolg in entscheidendem Maße vom Austausch von Informationen ab. Dazu müssen sich die Unternehmen moderner Informations- und Kommunikationssysteme bedienen. Die richtige Information zum richtigen Zeitpunkt am richtigen Ort wird daher zu einem integrierenden und zugleich einem der wichtigsten Produktionsfaktoren. [1,2]

In diesem Zusammenhang müssen zukünftige Informations- und Kommunikationstechnologien zwei wesentliche Anforderungen erfüllen. Zum einen müssen sie innerhalb eines Unternehmens die Integration von Anwendungen und Daten über die verschiedenen funktionalen Bereiche, von der Planung über Konstruktion zu Produktion und Montage, sicherstellen. Diese Aufgabe wäre für sich genommen eher statischer Natur, da eine einmal aufgebaute Struktur nicht ständig geändert werden muß. Zum anderen erzwingen jedoch neue Formen von Unternehmensnetzwerken die dynamische und kurzfristige Rekonfigurierbarkeit der Anwendungssysteme und bedingen damit eine höhere Flexibilität der zugrundeliegenden Informations- und Kommunikationstechnologien auch hinsichtlich der Integration externer Anwendungssysteme. [3,4,5]

Ein zentrales Problem innerhalb eines Unternehmens mit verteilten Produktionsumgebungen stellt die Verfügbarkeit und der Zugriff auf aktuelle Prozeßdaten dar. Diese bilden die Grundlage für fundierte Entscheidungen, indem sie die Entscheidungs- bzw. Handlungsträger mit allen notwendigen und relevanten Informationen über den Verlauf des Produktionsprozesses versorgen [6]. Dabei reicht die Spanne der Anwendungen von der kurz- bis mittelfristigen Produktionssteuerung über die betriebsbegleitende Simulation bis hin zur Überwachung des Produktionsprozesses in der Leitebene. Im Zuge der verstärkten Nutzung des Internets sind zudem die Ferndiagnose und Fernwartung produktionstechnischer Anlagen als wichtige Anwendungsfelder hinzugekommen [7].

Zur Übermittlung von Prozeßdaten werden in der Prozeßebene in zunehmendem Maße Feldbussysteme eingesetzt. Diese digitalen Kommunikationssysteme lösen schrittweise die parallele Verdrahtung mit analoger Signalübertragung ab. Neben den rein technischen Vorteilen, die in der digitalen Datenübertragung begründet liegen, können hinsichtlich der Engineering-, Installations- und Hardwarekosten bis zu 43% der Gesamtkosten eingespart werden [8]. Darüber hinaus ist der Einsatz offener, standardisierter Kommunikationsprotokolle wie sie die Feldbussysteme bieten, die wichtigste Voraussetzung für eine nahtlose Integration der Prozeßebene, um aktuelle Pro-

zeßdaten in den höheren Schichten der Automatisierungshierarchie weiterverarbeiten zu können [9].

Dennoch existieren für eine effiziente Einbindung der Prozeßebene in eine unternehmensweite, umfassende Informationsverarbeitung noch keine befriedigenden Lösungen. So ist es nicht möglich, daß Anwendungssysteme in der Zellen- oder Leitebene auf die von einem Feldgerät angebotenen Daten und Funktionen direkt zugreifen können. Unter einem "direkten" Zugriff wird dabei die über eine virtuelle Punkt-zu-Punkt Verbindung realisierte Kommunikation mit dem Feldgerät verstanden.

Die Gründe für diese Einschränkung sind einerseits in technischen Schwierigkeiten zu sehen, da Feldbussysteme ein anderes Übertragungsprotokoll als die Kommunikationssysteme der darüberliegenden Zellenebene aufweisen. Andererseits liegen sie im Umfeld der Feldbussysteme selbst. Dieses ist geprägt von einer großen Zahl unterschiedlicher und zueinander inkompatibler Systeme, deren Hersteller eine konsequente Strategie der Marktabschottung verfolgen. Eng mit der Festlegung auf ein bestimmtes Bussystem verbunden ist die Auswahl der geeigneten Feldgerätehersteller, der Steuerungssoftware und der einsetzbaren Engineeringwerkzeuge in der Leitebene. Durch die europäische bzw. internationale Standardisierung konkurrierender Feldbussysteme wurde zudem eine Chance verspielt, sich im Bereich der prozeßnahen Kommunikationssysteme auf einen einheitlichen Standard zu verständigen (vgl. Bild 1).

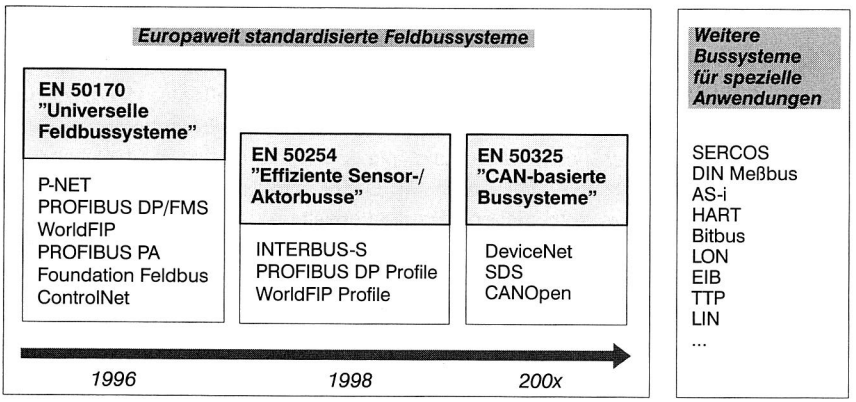


Bild 1: *Überblick über den derzeitigen Stand der europäischen Standardisierung von Feldbussystemen und über weitere Bussysteme*

Für die Hersteller von Steuerungs- und Leitstandsystemen wurde es in der Folge immer kostenintensiver, jedes einzelne Bussystem mit einem speziellen Treiber zu unterstützen. In einer gemeinsamen Initiative von Systemherstellern, Feldbusherstellern

und der Firma Microsoft wurde deshalb 1996 mit OPC (OLE for Process Control) eine feldbusneutrale Zugriffsschnittstelle am Übergang von der Prozeßebene zur Zellenebene definiert. Während diese Lösung einerseits einen wesentlichen Fortschritt bedeutet, da nun Anwendungen der Zellen- und der Leitebene unabhängig vom unterlagerten Feldbussystem entwickelt werden können, sind hiermit jedoch Konsequenzen verbunden, die von allen Beteiligten erst langsam erkannt werden:

- Da OPC eine feldbusneutrale Schnittstelle definiert, ist es hinsichtlich der Funktionalität auf die Schnittmenge der von den unterstützten Feldbussystemen angebotenen Dienste beschränkt.
- Bei den in OPC verfügbaren Diensten handelt es sich nur um Produktivdaten- und Ereignisdienste. Bietet ein Feldbussystem daher erweiterte Funktionalitäten, die der Anwender nutzen möchte, so müssen auch weiterhin spezifische Feldbustreiber eingesetzt werden, beispielsweise zur Bus- oder Gerätediagnose, zur Übertragung von Datenblöcken oder zum Starten und Stoppen von Programmen auf einem Feldgerät.
- Mittels OPC können zudem keine strukturierten Daten versendet oder die von einem Feldgerät angebotenen Funktionen aufgerufen werden. Dies kann als zusätzliche Ursache für den immer noch ausbleibenden Durchbruch der sogenannten intelligenten Feldgeräte angesehen werden. Diese sind gekennzeichnet durch dezentrale Verarbeitungsleistung, so daß sie einfache Regelungs- und Vorverarbeitungsaufgaben direkt "vor Ort" durchführen können. Damit vollziehen sie den Schritt von der Regelung über den Bus hin zum Client/Server-Prinzip, bei dem sie von der überlagerten Steuerung lediglich neue "Aufgaben" erhalten, die sie weitgehend autonom bearbeiten.

Um das Problem der Einbindung der Prozeßebene in eine unternehmensweite, umfassende Informationsverarbeitung zu lösen, bedarf es daher eines gänzlich anderen Ansatzes. Wie sich zeigen wird, geht die Lösung dieses Integrationsproblems mit der Beseitigung der es verursachenden Heterogenität Hand in Hand. Generell lassen sich nach *Wedekind* [10] zwei Ebenen der Heterogenität unterscheiden. Auf Ebene der *Implementierung* muß die Inkompatibilität von Betriebssystemen und Hardwareplattformen, Kommunikationsprotokollen und Programmiersprachen überwunden werden. Dazu ist ein einheitliches Systemkonzept erforderlich, welches erlaubt, aus Anwendungssicht von den genannten Unterschieden zu abstrahieren und das sich sowohl an die Anforderungen der Prozeßebene anpassen läßt, als auch in höhere Ebenen integrierbar ist.

Auf Ebene der *Anwendungslogik* muß die Heterogenität hinsichtlich der verwendeten Daten- und Schnittstellenmodelle überwunden werden. Dazu ist die Definition geeigneter Modelle notwendig, die eine anwendungsbezogene Integration der Feldgeräte in die Anwendungssysteme der darüberliegenden Ebenen ermöglichen. Ein offenes Daten- und Schnittstellenmodell stellt in diesem Zusammenhang sicher, daß auch in

einer verteilten Produktionsumgebung keine informationstechnisch begründeten Insellösungen entstehen [11,12].

Gegenstand der vorliegenden Arbeit ist die Entwicklung neuer Konzepte zur Erweiterung der Funktionalität vorhandener Feldbussysteme[1] mit dem Ziel der Überwindung der Heterogenität auf den beiden genannten Ebenen. Die weiteren Ausführungen sind dazu folgendermaßen gegliedert:

Das nachfolgende Kapitel stellt zunächst die Grundlagen bezüglich der Kommunikations- und Steuerungsstrukturen in der rechnerintegrierten Fertigung vor. Dabei erfolgt auch eine ausführlichere Analyse der Integrationsproblematik. Anschließend werden die für die weiteren Kapitel wichtigsten Basistechnologien aus den Bereichen der Middleware-basierten Frameworks und des Internets eingeführt. Zudem wird die Problematik der Sicherheit in verteilten Systemen aufgegriffen.

Im dritten Kapitel werden die qualitativen Anforderungen formuliert, die eine umfassende Lösung zur Integration der Prozeßebene in die Informationsverarbeitung eines global agierenden Unternehmens erfüllen muß. Insbesondere werden die Anforderungen hinsichtlich der beiden Ebenen der Heterogenität detailliert betrachtet. Dort erfolgt auch die Vorstellung und Bewertung der Feldbussysteme sowie der in diesem Zusammenhang verfügbaren Modellierungskonzepte.

Gegenstand der Kapitel vier bis sechs ist die Erweiterung bestehender Feldbussysteme, mit dem Ziel den herausgestellten Anforderungen zu genügen. Hierzu werden verschiedene Konzepte vorgeschlagen und die bei der Umsetzung erzielten Ergebnisse vorgestellt und diskutiert.

Wie sich zeigen wird, ist eine vollständige Erfüllung der Anforderungen nur durch einen umfassenden Ansatz im Rahmen der Entwicklung eines Middleware-Frameworks erreichbar. Kapitel sieben stellt ein solches Middleware-Framework vor, das gemäß der aufgestellten Anforderungen entwickelt wurde, und gibt eine detailliertere Darstellung einzelner Dienste und Eigenschaften. Den Abschluß bildet eine Bewertung des Konzepts.

1. Die im Verlauf vorgestellten Konzepte sind in weiten Teilen auch auf TCP/IP-basierte, prozeßnahe Kommunikationssysteme übertragbar.

2 Grundlagen und Stand der Technik

Dieses Kapitel stellt die wichtigsten Grundlagen, die für das Verständnis der Ausführungen der nachfolgenden Kapitel von Bedeutung sind, vor. Ausgehend von einem Überblick über die Kommunikationsstrukturen in der rechnerintegrierten Fertigung und im besonderen der Steuerungssysteme, gibt es zunächst eine Einführung in die Architektur und die Protokollstruktur von Feldbussystemen. Im Anschluß werden die allgemeinen Konzepte verteilter, objektorientierter Client/Server-Architekturen besprochen und am Beispiel von CORBA und DCOM konkretisiert. Den Abschluß bilden eine Betrachtung der wichtigsten Internet-Technologien sowie des Themas "Sicherheit" in verteilten Systemen. Hinsichtlich der Basisbegriffe der Kommunikation zwischen informationsverarbeitenden Systemen, wie etwa dem ISO-OSI-Basisreferenzmodell, soll vorab auf die einschlägige Literatur verwiesen werden ([13], [14]).

2.1 Kommunikation in der rechnerintegrierten Fertigung

2.1.1 Hierarchische Kommunikationsstrukturen in der rechnerintegrierten Fertigung

Das koordinierte und effiziente Zusammenwirken der einzelnen Funktionsbereiche innerhalb eines produzierenden Unternehmens verlangt nach einer Automatisierungslösung. Diese hat die Integration der Informationsflüsse zwischen den Funktionsbereichen sowie die Synchronisation mit dem Materialfluß zur Aufgabe, mit dem Ziel der rechtzeitigen Verfügbarkeit aller relevanten Daten an verschiedenen Orten [15]. Dazu ist der Aufbau eines Informationsverbunds Voraussetzung, der die verschiedenen hierarchischen Kommunikationsebenen innerhalb eines Unternehmens verbindet. Erschwert wird diese Aufgabe durch die Verschiedenartigkeit der zu vernetzenden Rechnersysteme, die sich hinsichtlich ihrer Hardware, Funktionalität, örtlichen Verteilung und Leistungsfähigkeit unterscheiden: CAD-Systeme (Computer-Aided Design) sind mit PPS-Systemen (Produktionsplanung und -steuerung) zu koppeln, Zellenrechner mit Leitrechnern und speicherprogrammierbaren Steuerungen. Darüber hinaus müssen eine Vielzahl einfacher und komplexer Automatisierungsgeräte verschiedenster Hersteller kommunikationstechnisch untereinander und mit CAx- bzw. PPS-Systemen gekoppelt werden, z.B. Sensoren, Aktoren, Motoren, Roboter, Werkzeugmaschinen, Transport- und Visualisierungssysteme. Zur Koordination dieser Systeme und Anwendungen bedarf es offener bzw. standardisierter Anwendungsschnittstellen und Kommunikationsprotokolle.

Die innerhalb eines Unternehmens vorherrschenden Kommunikationsstrukturen spiegeln in der Regel die verschiedenen hierarchischen Organisationsebenen wider. Damit wird gezielt eine Entkopplung der Informationsverarbeitung der einzelnen Ebenen angestrebt, die sich in ihren Anforderungen stark unterscheiden und damit auch aus

wirtschaftlichen Gründen angepaßte Systemlösungen benötigen. Bezüglich der genauen Einteilung der Ebenen und deren Bezeichnungen existieren in der Literatur unterschiedliche Begriffe. Einen Überblick über die Kommunikationsstrukturen der einzelnen Bereiche und der unterschiedlichen qualitativen Anforderungen, die sie erfüllen müssen, gibt Bild 2.

Zur Vernetzung der Rechnersysteme werden zunehmend standardisierte Kommunikationssysteme eingesetzt. In den eher dispositiven Bereichen basieren diese häufig auf dem Industriestandard TCP/IP, wobei die Anwendungsschicht entsprechend den Kommunikationsbedürfnissen ausgeprägt ist. In den prozeßnahen Bereichen findet derzeit eine Verdrängung der analogen Verdrahtungstechnik durch digitale Kommunikationssysteme, den Feldbussystemen, statt.

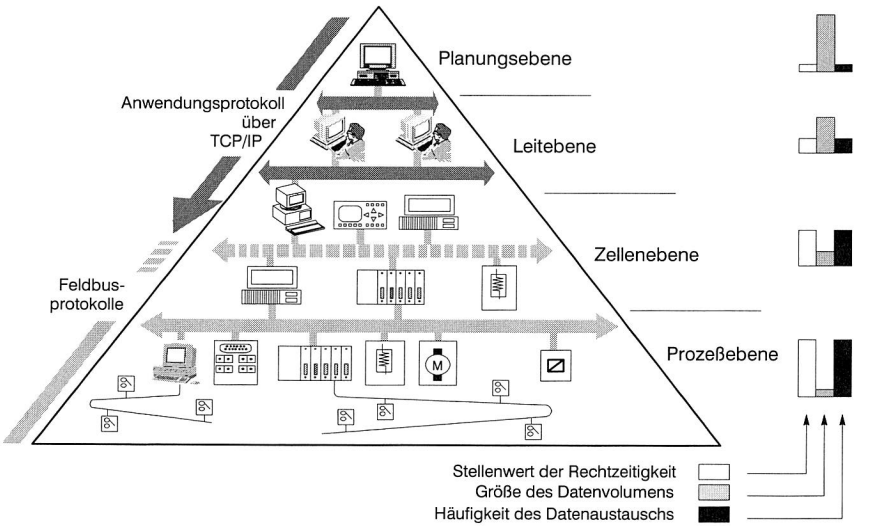


Bild 2: *Hierarchische Kommunikationsstrukturen eines Unternehmens und die unterschiedlichen qualitativen Anforderungen an die Datenübertragung*

Aufgrund der hierarchisch aufgebauten Kommunikationsstrukturen gilt es zwei Kommunikationsflüsse innerhalb eines Unternehmens zu unterscheiden. Der Begriff der *horizontalen Kommunikation* bezeichnet die Kommunikation von Anwendungen, die in der selben Hierarchie angesiedelt sind. Bedingt durch die Notwendigkeit der Kooperationen von Anwendungen in unterschiedlichen Ebenen der Hierarchie entsteht die *vertikale Kommunikation.* Die Steuerung und Integration der beiden Informationsflüsse ist die entscheidende Aufgabe der Informationsverarbeitung innerhalb eines Unternehmens.

2.1.2 Problemstellung bei der kommunikationstechnischen Integration der Prozeßebene

Während davon auszugehen ist, daß bei der horizontalen Kommunikation hinsichtlich der Hard- und Software und der Kommunikationssysteme eine weitestgehend einheitliche Systemumgebung vorherrscht, erweist sich die Realisierung der vertikalen Kommunikation unter Einbeziehung der prozeßnahen Bereiche als besonders schwierig. Dies liegt zum einen an den dort vorherrschenden Automatisierungssystemen selbst, die sich u.a. in den Betriebssystemen, Programmiersprachen und der lokalen Verfügbarkeit von Ressourcen sehr stark voneinander und von den Systemen der eher dispositiven Bereiche unterscheiden. Damit scheiden Standardlösungen für die Datenaufbereitung und -übertragung in der Regel aus.

Ein weiterer Punkt ist in der Verschiedenartigkeit der Kommunikationssysteme und in diesem Zusammenhang insbesondere der Anwendungsprotokolle der Prozeßebene im Vergleich zu den überlagerten Ebenen zu sehen. Im Gegensatz zu den Büronetzen sind Feldbussysteme für die zeitkritische Übertragung kleiner Datenmengen in Echtzeit optimiert und bieten daher keine besonders flexiblen und komfortablen Kommunikationsdienste an. Deshalb kommt am Übergang von der Zellenebene zur Prozeßebene bereits aus konzeptioneller Sicht keine Gateway-Lösung für eine Protokollumsetzung in Betracht, da eine eins-zu-eins Abbildung der existierenden Protokolle nicht sinnvoll realisierbar ist. In der Praxis wird das Problem auf zweifache Weise gelöst. Anwendungen, die auf Prozeßdaten zugreifen müssen, realisieren dies über feldbusspezifische Treiber, oder – und das ist der gegenwärtige Trend – über eine feldbusneutrale Zugriffsschnittstelle.

Große Probleme bereiten auch die fast unüberschaubare Vielzahl an Feldbussystemen. Die Realisierung einer feldbusneutralen Zugriffsschnittstelle stellt hier bereits einen wesentlichen Fortschritt dar, da eine Entkopplung von einem konkreten Feldbussystem erreicht wird. Jedoch bleibt ein derartiger Ansatz hinsichtlich der Funktionalität auf die Schnittmenge der von den unterlagerten Feldbussystemen angebotenen Dienste beschränkt – und das sind lediglich die Produktivdatendienste. Funktionen wie die Gerätediagnose, die Busdiagnose oder das Geräteengineering sind feldbusspezifisch und mit diesem Ansatz nicht in die höheren Ebenen integrierbar. Doch gerade diese Funktionen sind für den wirtschaftlichen Einsatz von Feldbussystemen extrem wichtig. Beispielsweise macht das Geräteengineering des PROFIBUS ein Fünftel der beim Einsatz anfallenden Gesamtkosten aus [8].

Die angeführten Schwierigkeiten bei der Integration der Prozeßebene in eine unternehmensweite Informationsverarbeitung lassen sich offensichtlich nur durch die Beseitigung der ihnen zugrundeliegenden Ursache lösen, der *Heterogenität.* Dabei können nach *Wedekind* [10] zwei Ebenen der Heterogenität unterschieden werden. Auf Ebene der *Implementierung* muß die Verschiedenheit von Betriebssystemen und Hardwareplattformen, Kommunikationsprotokollen und Programmiersprachen über-

wunden werden. Hier ist ein einheitliches Systemkonzept für alle hierarchischen Ebenen Voraussetzung, um aus Anwendungssicht von den genannten Unterschieden zu abstrahieren. Auf Ebene der *Anwendungslogik* muß die Heterogenität hinsichtlich der verwendeten Daten- und Schnittstellenmodelle überwunden werden. Dazu ist die Definition geeigneter Daten- und Schnittstellenmodelle notwendig, die eine anwendungsbezogene Integration der Feldgeräte in die Anwendungssysteme der darüberliegenden Schichten erlauben. Das Datenmodell beschreibt dabei, in welcher Weise Prozeßdaten, Konfigurationsdaten oder Betriebsdaten im Rechner abgebildet werden. Bei Verwendung einer relationalen Datenbank liegt das Datenmodell beispielsweise in Form des relationalen Schemas vor, das Wertebereiche, Relationen und Attribute umfaßt. Aufbauend auf dem Datenmodell ist eine Programmierschnittstelle und damit das Schnittstellenmodell festzulegen. Letzteres wird häufig in einer separaten Datenbank hinterlegt, dem sogenannten *Interface Repository*, um so zur Laufzeit die von einer Anwendung unterstützten Schnittstellen abfragen zu können.

An dieser Stelle soll kurz ein konzeptioneller Lösungsansatz zur Integration von Anwendungen und Daten innerhalb eines Unternehmens kritisch betrachtet werden, der auf einem unternehmensweit konsolidierten Datenmodell basiert, das sich in der Modellvorstellung einer globalen Unternehmensdatenbank widerspiegelt (Bild 3).

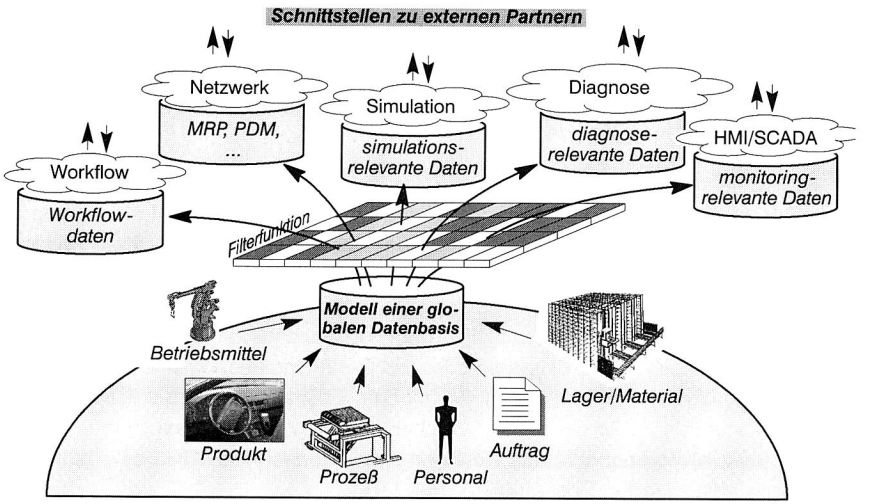


Bild 3: *Integration von Anwendungen und Daten innerhalb eines Unternehmens bei Einsatz einer globalen Datenbasis*

Diese umfaßt die wichtigsten Datenquellen, die sich nach *REFA* [16] hinsichtlich ihres Datenbezugs strukturieren lassen in Betriebsmitteldaten, Produktdaten, Prozeßdaten, Personaldaten, Auftragsdaten und Lager/Materialdaten. Die einzelnen Anwendungen

filtern die für sie relevanten Daten aus und bereiten sie entsprechen auf. Die Kommunikation der Anwendungen untereinander wird über die zugrundeliegende gemeinsame Datenbank abgewickelt.

In der betrieblichen Praxis sind globale Unternehmensdatenbanken jedoch nicht realisierbar. Es ist in diesem Fall die Aufgabe der eingesetzten Datenbanksysteme die notwendige Transparenz in Bezug auf die örtlich verteilten Datenquellen herzustellen. Versuche, darauf aufbauend unternehmensweit gültige, konsolidierte Datenmodelle zu implementieren sind vielfach gescheitert, jedoch immer mit hohen Kosten verbunden. Es ist sogar zu bezweifeln, daß in einem Szenario, in welchem ein Unternehmen verschiedene Komponenten mit ihren jeweiligen proprietären Datenbanken erwirbt und diese zu einer Gesamtlösung verbindet, ein konsolidiertes Datenmodell flexibel genug ist. Statt dessen bringen Schnittstellen-zentrierte Modelle, die beispielsweise beschreiben, welche Nachrichten die Anwendungen verstehen und wie redundante Daten konsistent gehalten werden können, in einer solchen Situation Vorteile [17]. Dies gilt um so mehr im Fall der Kooperation eines Unternehmens in verteilten Produktionsstrukturen, wenn der Aufbau eines firmenübergreifenden Datenmodells erst gar nicht möglich ist.

2.1.3 Entwicklung der Kommunikationsstrukturen von Steuerungssystemen

Die fortschreitende Modularisierung von Produktionssystemen wird prozeßnah in dezentralen Steuerungsstrukturen reflektiert. Mit der rasanten Entwicklung der Mikroelektronik und dem Preisverfall für elektronische Bauelemente wurde dabei der Grundstein für die Dezentralisierung geschaffen. Bereits einfache Feldelemente wie Sensoren und Aktoren können kostengünstig mit Mikroprozessoren ausgestattet werden und so Vorverarbeitungsaufgaben bis hin zu einfachen Steuerungsaufgaben vor Ort wahrnehmen.

Grundsätzlich können drei Aspekte der Dezentralisierung unterschieden werden, die in Bild 4 gezeigt sind:

- die *Funktion*, d.h. die Strukturierung der Steuerungsaufgabe in einzelne Teilfunktionen, z.B. die Teilfunktionen zur Steuerung des Materialflusses und der Technologiemodule einer flexiblen Fertigungszelle,
- die *Hardware*, d.h. die örtliche Verteilung von Hardwarekomponenten wie Steuerungen, Sensoren oder Aktoren und damit implizit die örtliche Verteilung von Prozeßdaten,
- die *Software*, d.h. die softwaretechnische Realisierung der Funktionen unter den Hardwaregegebenheiten.

Für den Anwender ergeben sich aus der Realisierung einer dezentralen Steuerungsstruktur eine Reihe von Vorteilen [18–22]. Insbesondere die Erhöhung der Flexibilität

beim Auf- und Umbau von Anlagen sowie die Verkürzung der Inbetriebnahmezeiten, durch die Möglichkeit die Teilsysteme vorher einzeln testen, sind wichtige Aspekte, die für ein derartiges Systemdesign sprechen.

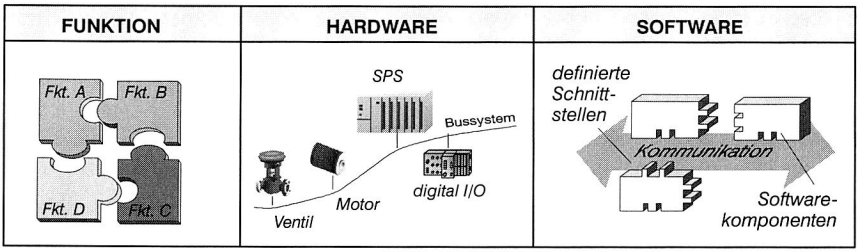


Bild 4: *Die verschiedenen Aspekte der Dezentralisierung – unterschieden nach Funktion, Hardware und Software*

Mit dem verstärkten Einsatz dezentraler Steuerungsstrukturen nimmt die Bedeutung des Informationsaustauschs zwischen den Teilsystemen zu (Bild 5). Bei einer früher üblichen, zentralen Steuerungslösung wurden mittels signalorientierter Verdrahtung Soll- und Istwerte analog zwischen der Peripherie und der Steuerung ausgetauscht.

Eine signifikante Reduzierung des Verkabelungsaufwands sowie eine schrittweise Verlagerung der Rechenleistung wird durch den Einsatz eines digitalen Kommunikationssystems erreicht, über das die Steuerung mit den dezentralen Peripheriegeräten kommuniziert. An den Peripheriegeräten selbst sind Gruppen von Sensoren und Aktoren durch parallele Verdrahtung angeschlossen. Die Regelung erfolgt dabei über das Kommunikationssystem, das hinsichtlich Übertragungszeit und Fehlertoleranz optimiert ist. Ein derartiges Systemdesign wird auch als "Dezentrale Peripherie" bezeichnet.

Mit der fortschreitenden Miniaturisierung elektronischer Bauelemente wird nun selbst die Ausstattung einfacher Sensoren und Aktoren mit Mikroprozessoren wirtschaftlich. Diese ermöglichen die Ausführung einfacher Operationen "vor Ort" sowie die Ersetzung der analogen Verdrahtung zum übergeordneten Feldgerät durch ein Sensor-/Aktor-Kommunikationssystem. Die Regelung erfolgt dabei in einem kleinen Regelkreis vor Ort bzw. über das Sensor-/Aktor-Kommunikationssystem. Im Idealfall ist die übergeordnete Steuerung durch einen PC ersetzt, der lediglich ereignisgesteuert, nach dem Client/Server-Modell, mit den Feldgeräten kommuniziert und auf diese Weise neue Sollwerte vorgibt. Gerade die höheren Anforderungen der Prozeßindustrie haben zu diesbezüglichen Weiterentwicklungen der klassischen Feldbussysteme geführt, so daß diese Stufe den derzeitigen Stand der Technik darstellt. Die Bezeichnung "Verteilte Intelligenz" trägt der geschilderten Charakteristik Rechnung.

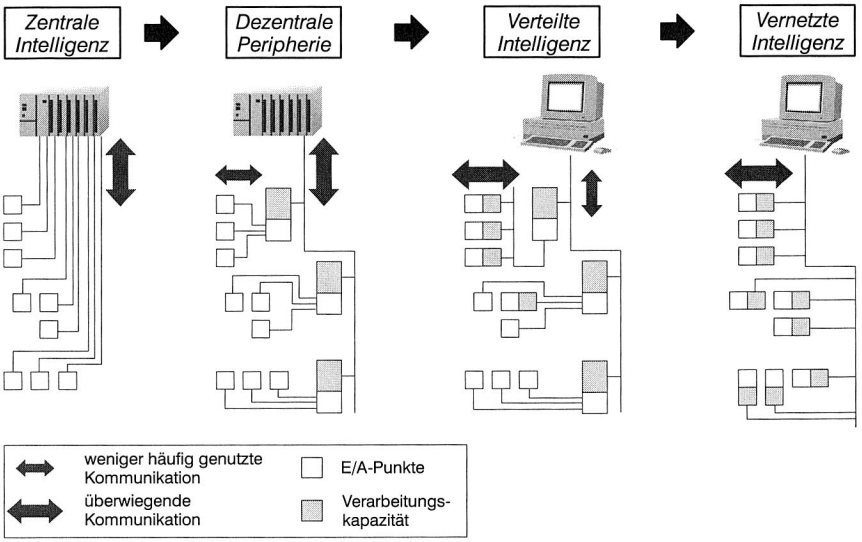


Bild 5: *Die vier Stufen der Dezentralisierung beim Einsatz intelligenter Feldgeräte (nach [23])*

Für die Zukunft ist abzusehen, daß die Dezentralisierung weiter fortschreiten wird, so daß selbst Sensoren und Aktoren programmierbar werden und so die Regelungsaufgabe praktisch vollständig vor Ort erfüllen können. Der Einsatz intelligenter Feldgeräte als Mittler entfällt damit weitgehend. Dieser noch eher theoretisch anmutende Grenzfall wird dabei in *Bender* [23] als "Vernetzte Intelligenz" bezeichnet.

2.1.4 Charakteristik von Feldbussystemen

In Übereinstimmung mit *Solvie* [24] wird im Kontext dieser Arbeit unter einem Feldbussystem jedes digitale Kommunikationssystem verstanden, das "in verteilten Automatisierungssystemen zur Erfüllung der Kommunikationsanforderungen derjenigen Automatisierungsgeräte dient, die in der untersten Ebene der Hierarchie der rechnerintegrierten Fertigung angesiedelt sind." Im Bereich der prozeßnahen Kommunikation werden Feldbussysteme zur Vernetzung von Feldgeräten, Sensoren und Aktoren untereinander und zur Anbindung an die überlagerten Steuerungssysteme eingesetzt. Die Vernetzung von Geräten der Zellenebene ist ebenfalls möglich, sie erfolgt aber zunehmend über Ethernet.

Der Einsatz von Feldbussystemen löst schrittweise die Prozeßkommunikation über analoge Schnittstellen und die als Übergangslösung anzusehende Smart-Technik ab. Bei der analogen Übertragung wird lediglich die Signalgröße unidirektional übermittelt, während bei der Smart-Technik (vgl. Bild 6) die zusätzliche bidirektionale Übertra-

gung von Informationen über entsprechend aufmodulierte Signale möglich ist. Ein Feldbussystem bietet demgegenüber die bidirektionale, digitale Übertragung komplexer Datentypen, wobei neben den reinen Prozeßwerten bereits standardmäßig Status- und Diagnoseinformationen enthalten sind.

Im Rahmen dieser Arbeit sind nur die universell einsetzbaren Feldbussysteme von Bedeutung. Diese sind zu unterscheiden von solchen Systemen, die für dedizierte Anwendungsfelder entwickelt wurden, und von den Sensor-/Aktor-Bussen, wie beispielsweise Interbus-S, CAN oder AS-i, die primär zur Anbindung einfachster binärer Sensoren und Aktoren konzipiert wurden. Mit den im vorherigen Kapitel eingeführten Bezeichnungen werden somit Systeme betrachtet, die man im engeren Sinne zur "Verteilten Intelligenz" zählen kann.

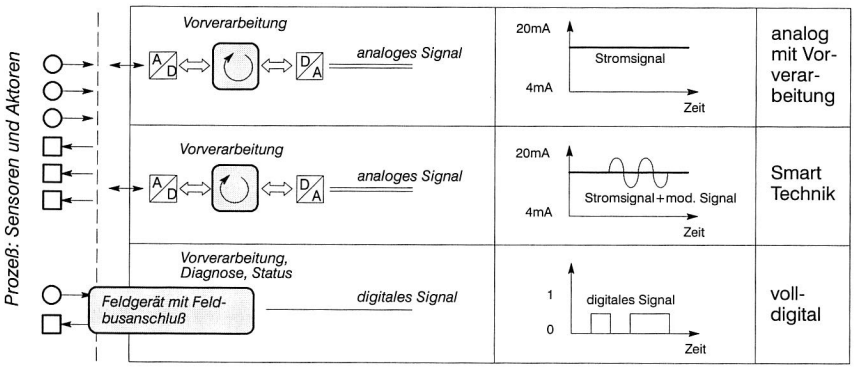


Bild 6: *Gegenüberstellung analoge Verdrahtung, Smart- und Feldbustechnik (nach [25])*

Feldbussysteme spielen vor allem in Europa eine maßgebliche Rolle bei der Vernetzung von Geräten der Feldebene. Der Vielzahl herstellerspezifischer und untereinander inkompatibler Feldbusse wird dabei in zunehmendem Maß durch europäische und internationale Standardisierung der relevanten Systeme begegnet. Die europäische Norm EN 50170 [26] umfaßt mit P-NET, PROFIBUS-DP/FMS, WorldFIP, PROFIBUS-PA, Foundation Fieldbus und ControlNet die sechs wichtigsten universell einsetzbaren Feldbussysteme, wobei allein der PROFIBUS einen Marktanteil von 48% in Deutschland bzw. 22% weltweit aufweist (Stand 1998, [27]). Eine zweite, für den Feldbusbereich wichtige Norm stellt die EN 50254 [28] dar, die u.a. mit dem INTERBUS-S einfachere Sensor-/Aktorbusse beinhaltet. Darüber hinaus existieren weitere Standardisierungsbemühungen im Bereich der CAN-basierten Bussysteme [29].

Vergleicht man Feldbussysteme mit den weit verbreiteten Kommunikationssystemen der Bürowelt, so ergeben sich im Aufbau des Protokolls bereits entscheidende Unter-

schiede. Verglichen mit dem ISO-OSI-Basisreferenzmodell implementieren sie nur eine Teilmenge der dort vorhandenen sieben Schichten. Dabei handelt es sich, wie in Bild 7 dargestellt, um die Bitübertragungsschicht, die Sicherungsschicht und die Anwendungsschicht. Dies geschieht aus Gründen der Effizienz, um den Anforderungen an eine zeitkritische und deterministische Datenübertragung Rechnung zu tragen.

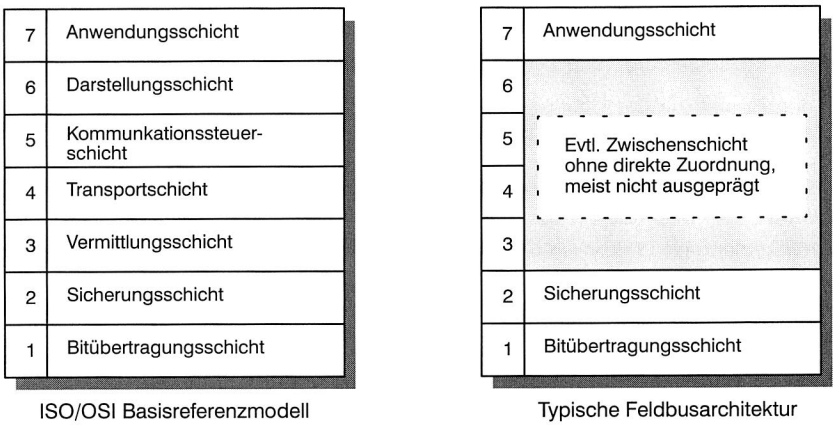


Bild 7: *Reduzierte Schichtenarchitektur in Feldbussystemen*

2.2 Verteilte Client/Server-Architekturen

Mit der Programmiersprache Smalltalk-80 wurde 1980 die Ära der objektorientierten Programmierung eingeleitet, die wesentlichen Einfluß auf die heutige Art und Weise der Softwareerstellung hat. Die Grundkonzepte der Objektorientierung werden durch Objekte, Klassen, Attribute, Operationen, Botschaften, Vererbung und Polymorphismus gebildet [30]. Mit der Abbildung dieser Konzepte auf eine konkrete Programmiersprache wurde dabei je nach zugrundegelegter "Objektphilosophie" eine eigene Programmiersprache geschaffen – Beispiele sind die Sprachen C++ oder Java. Die Idee der Wiederverwendbarkeit von Objekten bzw. Softwarekomponenten konnte damit nur innerhalb einer Programmiersprache verwirklicht werden.

Durch die Definition verteilter, objektorientierter Architekturen wurden zwei wesentliche Defizite der klassischen Objektorientierung behoben: Zum einen wurde die Wiederverwendbarkeit binärer, d.h. kompilierter Objekte ermöglicht, und so die Einschränkungen auf eine bestimmte Programmiersprache aufgehoben. Zum anderen können binär vorliegende Objekte, eine entsprechende Infrastruktur vorausgesetzt, nicht nur innerhalb eines gemeinsamen Prozeßraumes, sondern auch über Prozeß- und Rechnergrenzen hinweg verwendet werden [31].

2.2.1 Client/Server-Systeme

Ein zentraler Bestandteil verteilter, objektorientierter Architekturen ist das zugrundeliegende Client/Server-Modell. Nach einer einführenden Begriffsbestimmung werden diejenigen Architekturen vorgestellt, die aus dem Blickwinkel der Informatik einen wesentlichen Einfluß auf die zukünftige Art und Weise der Softwareerstellung ausüben werden. Die Bedeutung dieser Architekturen für den Bereich der rechnerintegrierten Fertigung ist eminent, da hier die Aufgabe der Koordination und Kooperation hochgradig verteilter, heterogener Systeme im Mittelpunkt steht.

Für den Begriff der *Client/Server-Systeme* existiert in der Literatur keine einheitliche Definition, weshalb in Übereinstimmung mit den inhaltlichen Gemeinsamkeiten im Kontext dieser Arbeit folgende Festlegung getroffen wird:

Definition 1: *Client/Server-Systeme*

In Anlehnung an [32] ist ein Client/Server-System typischerweise charakterisiert durch

a) die funktionale Trennung zweier Prozesse in einen Dienstanforderer (Client) und einen Diensterbringer (Server),

b) die Nutzung gemeinsamer Ressourcen, d.h. ein Server kann mehrere Clients gleichzeitig bedienen und koordiniert deren Zugriff auf gemeinsame Ressourcen,

c) ein asymetrisches Protokoll, d.h. Clients initiieren die Kommunikation, indem sie einen Service aufrufen, der Server wartet passiv auf Anfragen,

d) einen nachrichtenbasierten Datenaustausch zwischen Client und Server (Dienstanforderung/request und Rückmeldung/response),

e) die Kapselung der Dienste, d.h. der Client kennt nur die Schnittstellen des Servers, die Implementierung ist ihm verborgen.

Darüber hinaus weisen Client/Server-Systeme oft folgende Eigenschaften auf:

f) Garantierte Integrität der Daten des Servers bei konkurrierendem Zugriff mehrerer Clients

g) Skalierbarkeit, d.h. die Anzahl der Clients hat nur geringe Auswirkungen auf die Performance des Gesamtsystems (horizontale Skalierbarkeit) bzw. die Server-Software wird durch Wahl einer leistungsfähigeren Rechnerplattform entsprechend performanter (vertikale Skalierbarkeit)

h) Ortstransparenz, d.h. Client und Server können auf dem selben Rechner oder auf verschiedenen Rechnern ausgeführt werden. Die Client-Software übernimmt – idealerweise transparent für den Programmierer – die Aufgabe, den Server innerhalb eines Netzwerks zu lokalisieren.

i) Plattformunabhängigkeit, d.h. die Client- und die Server-Software sollten über verschiedene Hardware- und Betriebssystemplattformen austauschbar sein.

Typische Beispiele für Client/Server-Systeme sind Druck-Server, File-Server oder Programm-Server. Das Internet stellt weltweit das größte Client/Server System dar und

wird in Kapitel 2.5 detaillierter besprochen. Im Bereich der Fertigungsautomatisierung existiert mit dem MMS-Protokoll (Manufacturing Message Specification, [33]) ein ISO-Standard zur Kommunikation zwischen Rechnern und Automatisierungsgeräten, der ebenfalls dem Client/Server-Modell folgt.

2.2.2 Begriffsbestimmungen "Middleware" und "Frameworks"

Nach *Bernstein* [34] werden verteilte Systemdienste, die standardisierte Schnittstellen (API – Application Programming Interface) und ein standardisiertes Protokoll besitzen, als Middleware bezeichnet. Beispiele sind Netzwerktreiber von Datenbankherstellern oder das *Remote Procedure Call* Protokoll (RPC-Protokoll), spezifiziert im Distributed Computing Environment (DCE) der Open Software Foundation (OSF). Im Rahmen dieser Arbeit wird der Begriff "Middleware" sinngemäß verwendet:

Definition 2: *Middleware (nach [34])*
Unter Middleware werden verteilte Systemdienste verstanden, die auf unterschiedlichen Rechner- und Betriebssystemplattformen verfügbar sind, eine standardisierte Programmierschnittstelle besitzen und auf standardisierten Übertragungsprotokollen basieren. In einem Schichtenmodell sind diese Dienste oberhalb der Betriebssystemdienste und unterhalb der eigentlichen Anwendung angesiedelt.

Gemäß der Festlegung umfaßt Middleware demnach die APIs, die der Client nutzen kann, um einen Dienst des Servers anzufordern, die Übermittlung der Dienstanforderung über das Netzwerk sowie die Rückmeldung des Servers. Aufgrund der Ansiedlung der Dienste oberhalb des Betriebssystems entkoppelt die Middleware die Anwendungen von den spezifischen Eigenschaften der Hardware und des Betriebssystems.

Bernstein [34] unterscheidet Middleware bezüglich anwendungsunabhängiger Middleware und den anwendungsspezifischen Middleware-Frameworks, die bestimmte Anwendungen gezielt unterstützen. Beispiele für anwendungsunabhängige Middleware sind die Architekturen CORBA (vgl. Kapitel 2.3) oder DCOM (vgl. Kapitel 2.4) oder das Internet-spezifische HTTP-Protokoll (vgl. Kapitel 2.5). Transaktions- und Groupwaresysteme oder verteilte Datenbanken stellen hingegen Middleware-Frameworks dar.

Definition 3: *Middleware-Framework (nach [34])*
Ein Middleware-Framework stellt eine anwendungsspezifische Softwareumgebung dar, die ein Framework-spezifisches API, eine Benutzerschnittstelle und gegebenenfalls Werkzeuge umfaßt. Zur Realisierung eines Middleware-Frameworks wird auf die Middleware-Dienste zurückgegriffen.

Da die im Rahmen dieser Arbeit betrachteten Frameworks ausschließlich auf Middleware basieren, werden die Begriffe Middleware-Framework und Framework synonym

gebraucht. In Bild 8 ist die Architektur von Frameworks dargestellt. Anwendungen, die in ein Middleware-Framework eingebettet sind, steht es dabei frei, auf die Middleware-Dienste direkt zuzugreifen oder die speziellen Dienste des Middleware-Frameworks zu nutzen. Ein wesentlicher Aspekt von Frameworks ist ihre Anwendungsbezogenheit. Der durch ein Framework typischerweise vorgegebene Rahmen, innerhalb dessen die Anwendungen ausgeführt werden, kann nicht nur die Verwendung von Schnittstellen und Datenformaten regeln, sondern darüber hinaus auch das Zusammenwirken der Anwendungen selbst.

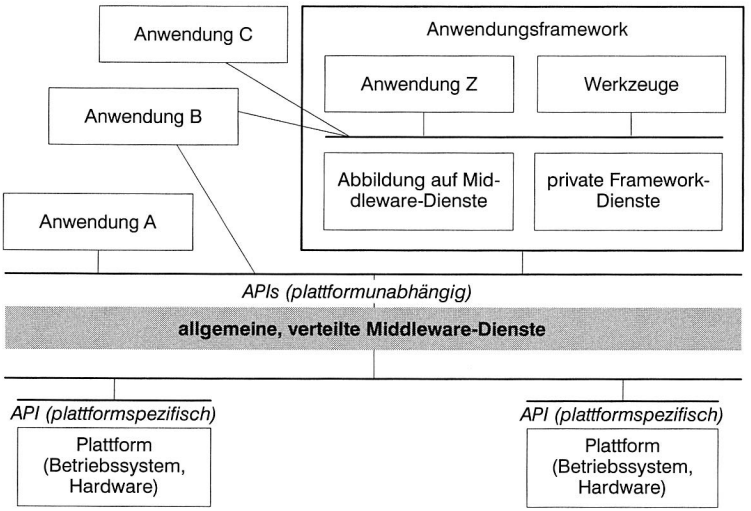


Bild 8: *Architektur von Middleware-Frameworks (nach [34])*

Aus Sicht des Anwendungsentwicklers ist die Programmierung verteilter Anwendungen von Interesse und damit das auf der jeweiligen Middleware aufbauende Programmiermodell. Allgemein handelt es sich um *verteilte Client/Server-Modelle*, in einer objektorientierten Programmierumgebung spricht man von dem *verteilten Objektmodell*.

2.2.3 Grundlagen eines verteilten Objektmodells

Die Definition eines verteilten Objektmodells gewährleistet, daß in einem heterogenen, objektorientierten Client/Server-System die einzelnen Objekte über Prozeß- und Rechnergrenzen hinweg miteinander zusammenarbeiten können. Um die Programmiersprachenunabhängigkeit zu gewährleisten, trennt ein Objektmodell die Schnittstellendefinitionen von der tatsächlichen Implementierung. Zum Aufbau der Kommunikation stellt ein Objektmodell Dienste zur Verfügung, die es einem Client erlauben, ein Objekt zu lokalisieren und die für den Methodenaufruf benötigte Objektreferenz zu erhalten.

Ein entfernter Methodenaufruf besteht dann aus der Zusammenstellung und Verpackung des Methodenaufrufes (marshaling), der Versendung an das Ziel, dem Entpacken des Methodenaufrufes (unmarshaling), der Ausführung der Methode durch den Server und der Zurücksendung der Ergebnisse an den Client. In einer heterogenen Umgebung muß dazu ein einheitliches Datenaustauschformat definiert sein, um das Marshaling bzw. Unmarshaling für die verschiedenen Datentypen korrekt durchführen zu können.

Eine weitere Charakteristik eines verteilten Objektmodells ist, daß sich die kompilierten Objekte zur Laufzeit zu einem Gesamtsystem zusammenfügen lassen. Außerdem besteht in der Regel die Möglichkeit, Methodenaufrufe zur Laufzeit aus der Schnittstellendefinition zu generieren und zu versenden. Dies hat einerseits den Vorteil, daß Server-Objekte austauschbar ohne Änderungen des Client austauschbar sind. Andererseits kann es sein, daß zur Implementierungszeit nicht unbedingt feststeht, welche Schnittstellen des Servers der Client tatsächlich verwendet wird. Dies trifft insbesondere für Skript-Sprachen zu. Über die genannten Mechanismen hinaus, bietet ein verteiltes Objektmodell noch Dienste für die Instantiierung, Verwaltung und Löschung entfernter Objekte.

2.3 Die Objektarchitektur der OMG

Die *Object Management Architecture* (OMA, [35]) der *Object Management Group* (OMG) beschreibt eine abstrakte Architektur verteilter Objektsysteme, in der als Komponenten der *Object Request Broker* (ORB, [36]), *CORBA Services* (COSS, [37]), *CORBA Facilities* [38] und Anwendungen unterschieden werden (Bild 9).

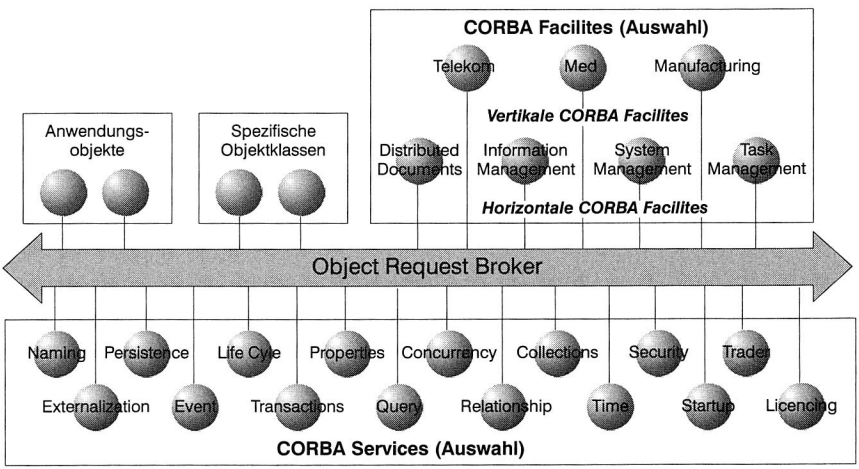


Bild 9: *Die "Object Management Architecture" der OMG (nach [35])*

Der ORB stellt dabei ein allgemeines Kommunikationsmedium zur Verfügung, das von den anderen Komponenten verwendet wird. Mit den CORBA Services werden anwendungsunabhängige, systemnahe aber dennoch plattformunabhängige Dienste definiert. Die CORBA Facilities stellen Dienste bereit, die zur Unterstützung von bestimmten Anwendungen gedacht sind. Während die horizontalen Facilities ein breiteres Einsatzspektrum abdecken, sind die vertikalen Facilities stark anwendungsspezifisch (Frameworks).

2.3.1 CORBA-Objekte

Zur Beschreibung der von Server-Objekten angebotenen Dienste dient als Schnittstellenbeschreibungssprache die *Interface Definition Language* (IDL). Sie ist rein deklarativ und der Syntax von C++ angelehnt. CORBA basiert auf einem klassischen Objektmodell und erlaubt u.a. Mehrfachvererbung und Polymorphismus auf Schnittstellenebene. Die Referenzierung eines Objekts geschieht über eine systemweit eindeutige Objektreferenz, die bei der Instantiierung des Objekts vergeben wird. Ein Client nimmt den Dienst eines Objekts in Anspruch, indem er einen Methodenaufruf ausführt, der aus dem Methodennamen, der Objektreferenz, Parametern und einem wahlfreien Kontext besteht. Dieser Kontext enthält Informationen über die Position des Aufrufers, außerdem werden hier eventuelle Fehlerrückmeldungen übergeben. Die Erzeugung und Löschung von Objekten erfolgt ebenfalls über Methodenaufrufe. CORBA spezifiziert eine Anzahl von Typen, die für Parameterübergaben verwendet werden können. Es stehen einfache Typen wie *short*, *long* und *string*, Strukturtypen wie *array* und *sequence* sowie Objektreferenzen zur Verfügung. Übergabeparameter sind in Eingabe-, Ausgabe- und Ein-/Ausgabeparameter unterteilt, je nachdem, ob Werte nur übergeben oder auch zurückgeliefert werden.

2.3.2 Die Architektur von CORBA

Innerhalb der CORBA Spezifikation stellt der Object Request Broker die Kommunikationsdienste zur Verfügung, die zum Absetzen von Methodenaufrufen und Empfangen der Rückmeldungen dienen. Für die Entwicklung eines Clients spielt es dabei keine Rolle, ob sich der Server auf dem selben Rechner oder innerhalb eines Netzwerks auf einem entfernten Rechner befindet.

Ausgehend von der IDL-Beschreibung der Server-Dienste werden mittels eines IDL-Compilers *Stubs* generiert. Diese sind spezifisch für die Programmiersprache in der der Client erstellt wird und agieren als *Proxy* (Stellvertreterobjekt) für den eigentlichen Server. Mit Hilfe des *Dynamic Invocation Interface* (DII) besteht darüber hinaus die Möglichkeit, die Dienste eines Servers zur Laufzeit abzufragen und aufzurufen. Eine verteilte Datenbank, das *Interface Repository*, dient zur Registrierung von IDL-Beschreibungen und ermöglicht damit das Auffinden von Diensten und die Abfrage der Dienstparameter. Weitere Funktionen, auf die hier nicht näher eingegangen wird, werden über die ORB Schnittstelle zur Verfügung gestellt (vgl. Bild 10).

Ausgangspunkt für die Implementierung des Servers stellt die IDL-Beschreibung dar, aus der, mit Hilfe eines IDL-Compilers, *Skeletons* generiert werden. Diese sind analog zu den Stubs programmiersprachenspezifisch und bilden auf der Server-Seite das Gegenstück zum Client-Proxy. Entsprechend ist das *Dynamic Skeleton Interface* (DSI) das Pendant zum DII des Clients und in der Lage, die Dienste eines Servers dynamisch aufzurufen. Aufgabe des *Object Adapter* ist die Entgegennahme der Dienstanfragen und das Weiterleiten der Anfragen an den zuständigen Server, der, falls notwendig, zuvor automatisch gestartet wird. Die Informationen über die verfügbaren Server-Objekte sowie zusätzliche ORB spezifische Informationen werden im *Implementation Repository* hinterlegt.

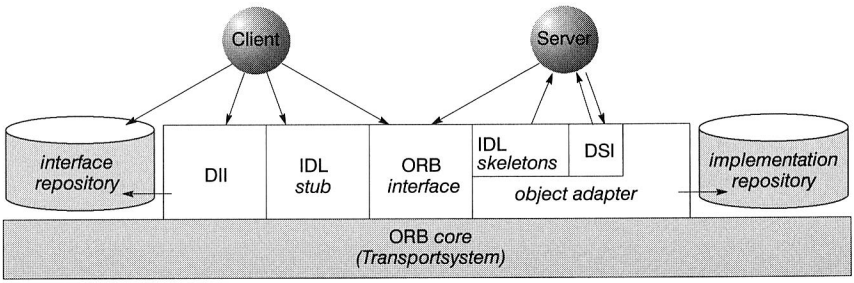


Bild 10: *Aufbau eines zur CORBA 2.2 Spezifikation konformen Object Request Broker (nach [36])*

2.4 Das Distributed Component Object Model

Microsoft's *(Distributed) Component Object Model* ((D)COM) ist im wesentlichen ein Programmiermodell mit eine Menge von Systemdiensten, das schrittweise aus der ursprünglichen Spezifikation von *OLE* (Object Linking and Embedding) hervorgegangen ist. Mit dem Betriebssystem Windows NT 4.0 liegt erstmals ein verteiltes Programmiermodell vor, das mit eingeschränkter Funktionalität mittlerweile auch auf anderen Plattformen verfügbar ist [39].

2.4.1 COM-Objekte

COM definiert einen *Binärstandard*, der festlegt, wie Objekte im Binärcode vorliegen müssen. Zur Beschreibung der Schnittstellen (Interfaces) von COM-Objekten dient die *Object Description Language* (ODL). Im Gegensatz zu CORBA basiert COM nicht auf einem klassischen Objektmodell und unterstützt insbesondere keine Vererbung. Die Wiederverwendung von COM-Objekten wird über die Mechanismen *Containment* und *Aggregation* erreicht [40].

COM-Schnittstellen stellen eine Gruppe von verwandten Funktionen dar. Sie besitzen keinen internen Zustand, so daß, im Gegensatz zu CORBA, auf Basis der Objektschnittstellen keine Objektinstanzen erzeugt werden können. Demnach greifen COM-Clients, die sich mehrmals mit einem COM-Objekt verbinden, in der Regel nicht auf ein und dasselbe Objekt zu, sondern lediglich auf eine (beliebige) COM-Implementierung der entsprechenden COM-Schnittstelle.

COM-Objekte werden über ihren *Class Identifier* (CLSID) identifiziert, die einzelnen Schnittstellen über den *Interface Identifier* (IID) wie in Bild 11 dargestellt. Zur Interaktion benötigt der Client einen Zeiger auf ein bestimmtes Interface, um dann über das *IUnknown-Interface*[2], das jedes COM-Objekt anbieten muß, die gewünschten Methoden in Form von Funktionszeigern zu erfragen.

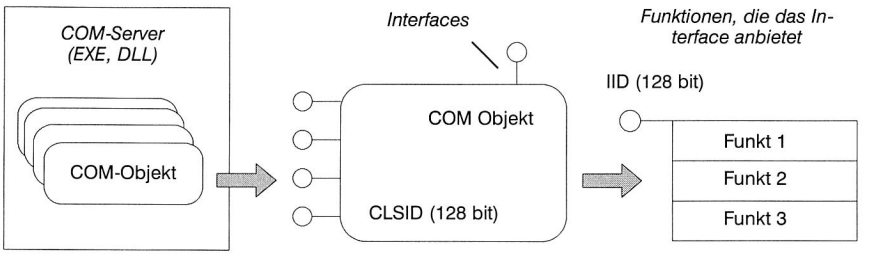


Bild 11: *Das Programmiermodell von COM*

2.4.2 Die Architektur von COM

Den Kern der COM-Architektur bildet ein Softwarebus, ähnlich dem Object Request Broker (vgl. Bild 10). Die Systemdienste von COM *(COM System Services)* werden in Form einer COM-Bibliothek zur Verfügung gestellt. Diese beinhaltet derzeit APIs für die folgenden COM-Services: *Locator Service*, um COM-Server zu lokalisieren und zu starten, *Transparent Local Procedure Calls (LPCs)* und *Remote Procedure Calls (RPCs)* für lokale oder entfernte Server. Weitere Services sind in Vorbereitung.

Die Registrierungsdatenbank *(System Registry)* fungiert als systemweite Datenbank und bildet eine Art Tabelle für die CLSIDs, die den zugehörigen Filenamen eines COM-Servers zurückliefert. Die Registrierungsdatenbank ist mittels *hierarchischer Schlüssel* organisiert, die Wurzel des Schlüsselbaums wird durch den *root key (HKEY_CLASSES_ROOT)* gebildet.

Das COM-Programmiermodell bietet dem Client Ortstransparenz bezüglich des Servers, indem Methodenaufrufe an ein entferntes COM-Objekt über ein Proxy-Objekt

2. Die Namenskonvention sieht vor, daß COM-Interfaces mit einem Prefix "I" versehen werden.

versendet werden. Dieses nimmt anstelle des Servers alle Aufrufe entgegen und leitet sie über einen *Local* bzw. *Remote Procedure Call* (LPC bzw. RPC, lokaler bzw. entfernter Funktionsaufruf) an ein Stub-Objekt weiter, das mit dem Server in Verbindung steht. Der LPC wird im Rahmen eines lokalen Methodenaufrufs eingesetzt, wenn sich die COM-Objekte zwar auf einem Rechner, jedoch in unterschiedlichen Adreßräumen befinden. Der RPC dient zum Aufruf von Objekten, die physikalisch auf einem anderen Rechner lokalisiert sind, zu dem eine Netzwerkverbindung besteht.

In Bild 12 ist das Prinzip des Methodenaufrufs zusammenfassend dargestellt. Die Methoden des Servers werden in einem *Virtual Function Table* (VTBL) verwaltet, der Zeiger auf die einzelnen Methoden speichert. Der Client besitzt lediglich einen Zeiger auf das Interface *(pISomeInterface)* und muß daher über eine weitere Indirektion, einen Zeiger auf die VTBL *(pVTBL)*, zunächst auf den VTBL zugreifen.

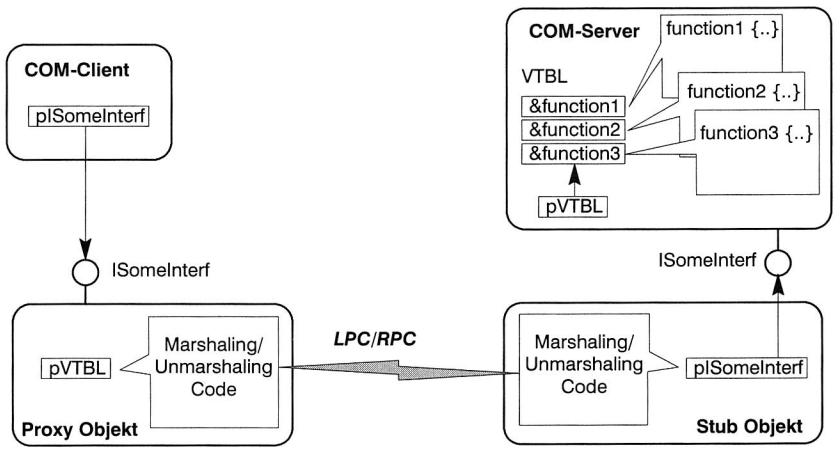


Bild 12: *Ortransparente Kommunikation zwischen Client und Server über einen lokalen bzw. entfernten Funktionsaufruf (nach [39])*

2.5 Vorstellung relevanter Internet-Technologien

Das Internet stellt mit mehr als 80 Millionen miteinander verbundenen Rechnern [41] das weltweit größte und am schnellsten wachsende Client/Server-System dar. Seit der Einführung des ersten graphischen Browsers 1993, ist das *World Wide Web* (WWW oder Web) der bekannteste Teil des Internet. Dem allgemeinen Sprachgebrauch folgend, werden im folgenden die Begriffe Internet und WWW weitgehend synonym verwendet. Durch die schrittweise Einführung neuer Technologien, verändert sich das,

ursprünglich auf die rein statische Publizierung von Hypertext-Dokumenten beschränkte, WWW zu einer Basis für interaktive Anwendungen. Die Bereiche des E- und M-Commerce sind beste Beispiele hierfür.

Bild 13 gibt eine Übersicht über die bedeutendsten technologischen Entwicklungen der letzten Jahre mit ihrem (geschätzten) Verbreitungszeitpunkt. Oft wird in diesem Zusammenhang von den "Internet-Technologien" gesprochen, ohne daß eine klare Begriffsbestimmung hierzu existiert, weshalb folgende Definition hier maßgeblich sein soll:

Definition 4: *Internet-Technologie*
Der Begriff der "Internet-Technologie" umfaßt die verfügbaren Technologien auf Protokoll- und Anwendungsebene, die für die Interaktion von Rechnersystemen über das Internet zur Verfügung stehen.

Über die in Bild 13 dargestellten Technologien hinaus, umfaßt die Definition damit u.a. das dem Internet zugrundeliegende TCP/IP-Protokoll, die Dienste der TCP/IP-Protokollfamilie, wie FTP (File Transfer Protocol) oder Telnet, und die für die Bereitstellung der Hypertext-Dokumente eingesetzten Web-Server.

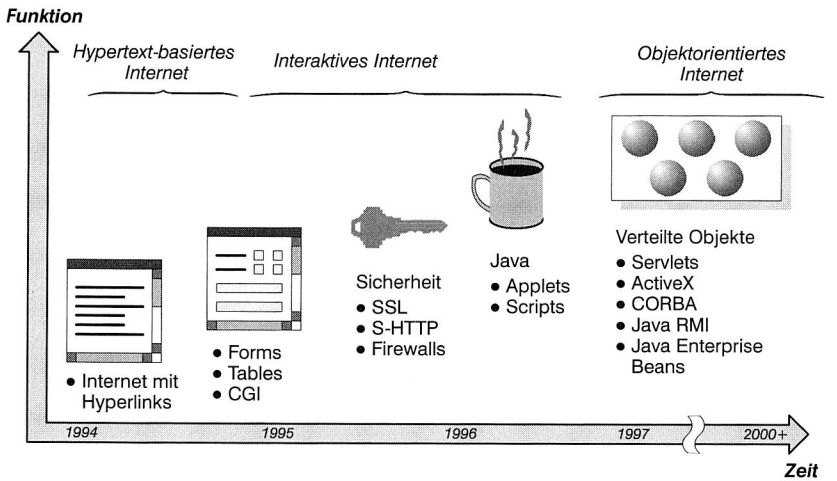


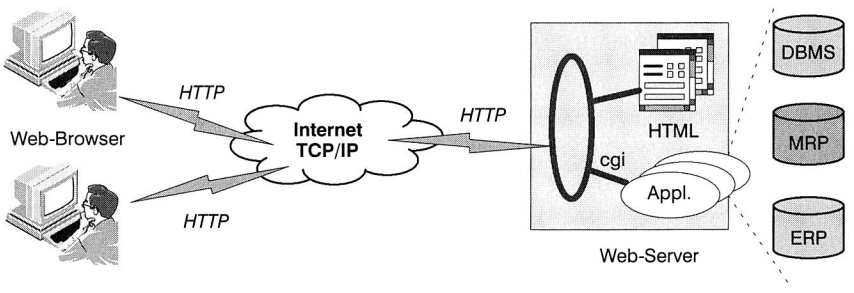
Bild 13: *Entwicklung der WWW-Technologien (nach [32])*

2.5.1 Grundlagen der Internet-Technologie

Die *Hypertext Markup Language* (HTML, [42,43]) ist die Sprache für die Veröffentlichung von multimedialen Dokumenten im Internet. Sie wurde ursprünglich am CERN zum internen Dokumentenaustausch entwickelt. Primär ist HTML eine Sprache, mit deren Hilfe in Dokumenten enthaltene Information strukturiert werden kann. Jedoch

wird auch die Darstellung von Dokumenten mit beeinflußt. Standardstrukturelemente, wie Überschriften, Absätze oder Listen, werden durch in den Text eingefügte Marken *(Tags)* gekennzeichnet. Die Konzentration auf die Dokumentenstruktur und das Weglassen von Layoutmöglichkeiten macht es erheblich leichter, Dokumente von einem System auf ein anderes zu transportieren, was HTML im heterogen aufgebauten Internet letztlich zum Durchbruch verholfen hat. Ein zentrales Element stellt das in HTML verwirklichte Konzept des Hypertextes dar. Damit können Dokumentabschnitte Verweise auf andere Dokumente darstellen. Die Idee dazu wurde bereits 1945 von *Bush* [44] entwickelt und 1989 von *Berners-Lee* in seinem Vorschlag eines "Information Management System" [45] aufgegriffen – dem Vorläufer des modernen WWW. Zur Übertragung der HTML-Dokumente wird das zustandslose *Hypertext Transfer Protocol* (HTTP, [46,47]) eingesetzt.

Client/Server-Interaktionen im WWW finden typischerweise zwischen Web-Browsern und Web-Servern statt. Der Web-Browser spricht einen Server über die Angabe des *Uniform Resource Locator* (URL, [48]) an. Die URL dient der eindeutigen Lokalisierung von Ressourcen im WWW, beispielsweise HTML-Dokumenten, und beschreibt zudem das für die Übertragung zu verwendende Protokoll. Die typische 3-Schichten-Architektur von WWW-basierten Client/Server-Systemen ist in Bild 14 dargestellt. Hierbei bildet das Internet, basierend auf dem Übertragungsprotokoll TCP/IP, die Middleware-Schicht.



MRP: Manufacturing Resource Planning
ERP: Enterprise Resource Planning
DBMS: Datenbanksystem

Bild 14: *3-Schichten-Architektur von WWW-basierten Client/Server-Systemen (nach [49])*

Neben der Bereitstellung von HTML-Dokumenten durch den Web-Server, können interaktive Anwendungen über spezielle Schnittstellen und Server-Erweiterungen reali-

siert werden. Auf diese Weise können Clients auf ursprünglich nicht-internetfähige Anwendungssysteme, z.B. Datenbanksysteme, zugreifen. Eine weit verbreitete und sehr einfache Schnittstelle ist das *Common Gateway Interface* (CGI-Schnittstelle), über die der Client Anwendungsprogramme, typischerweise auf dem Server selbst, starten kann. Durch die Nutzung verschiedener neuerer Technologien wie Java-Applets, Servlets, Java-RMI (Kapitel 2.5.4) oder *Active Server Pages* (ASP) können über das Internet auch komplexere interaktive Sitzungen realisiert werden.

2.5.2 Die "Extensible Markup Language"

Die *eXtensible Markup Language* (XML, [50]) ist wie HTML eine Sprache zur Beschreibung von Inhalten im WWW. Sie stellt im wesentlichen ein Anwendungsprofil der noch allgemeineren *Standard Generalized Markup Language* (SGML, ISO 8879) dar. Ebenso wie bei HTML besitzen XML-Dokumente gezwungenermaßen eine innere Struktur, die durch die Tags mit ihren optionalen Attributen bestimmt ist und die das Grundgerüst des Dokuments aufbauen. Während jedoch ein HTML-Programmierer nur den durch die HTML-Spezifikation vorgegebenen Wortschatz verwenden kann, in der die genaue Bedeutung der einzelnen Tags und Attribute festgelegt ist, kennt XML dagegen überhaupt keine vordefinierten Tags oder Attribute. Der Autor einer XML-Seite hinterlegt vielmehr in einer separaten *Document Type Definition* (DTD) seine Tags und Attribute. Ein XML-Parser vergleicht den XML-Quelltext mit der DTD und kann dabei feststellen, ob es sich um ein gültiges Dokument im Sinne der DTD handelt. Für viele Zwecke existieren bereits passende DTDs, etwa MathML für mathematische Formeln oder SMIL (Synchronized Multimedia Integration Language) für Multimedia-Daten.

Bei HTML beeinflussen die Tags gleichzeitig die Struktur und die Darstellung im Web-Browser. In einem XML-Dokument wird lediglich der Inhalt strukturiert. Die vom Autor beabsichtigte Darstellungsform eines Dokuments wird hingegen ebenfalls separat hinterlegt. Entweder in einem *Cascading Style Sheet* (CSS), das bereits in HTML zur dynamischen Formatierung der Darstellung eingesetzt werden konnte, oder in einem in der *Extensible Stylesheet Language* (XSL) definierten *XSL-Style-Sheet*. Letzteres hat den Vorteil, daß nicht nur die Anzeige eines Dokuments formatiert, sondern dessen Inhalte auch transformiert werden können mittels *XML-Transformations* (XSLT).

Da die Bedeutung der Tags und Attribute in XML prinzipiell nicht festgeschrieben ist, erlaubt es XML auf sehr einfache und individuelle Weise, beliebig strukturierte Informationen in einem entsprechenden textbasierten Format abzubilden. Andererseits kann der das Dokument auswertende oder darstellende Client die Tags und Attribute auf vielfältige Weise verarbeiten. Man spricht in diesem Zusammenhang auch von XML als eine *Metasprache*, da der Autor eines Dokuments seine eigene formale Sprache im Rahmen der DTD spezifizieren kann. Ein Beispiel für ein strukturiertes Datenobjekt, das sich mit XML modellieren läßt, ist der Auszug einer Feldgerätebeschreibungsdatei:

```
<?xml version="1.0"? standalone='yes'>
<!DOCTYPE gsd SYSTEM "gsd.dtd">
<gsd-definition>
  <Unit-Definition-List>
    <GSD_Revision>   1          </GSD_Revision
    <Vendor_Name>    SIEMENS    </Vendor_Name>
    <Model_Name>       DP/PA-Link  </Model_Name>
    <Revision>         V1.2     </Revision>
    <Ident_Number>     0x8052   </Ident_Number>
    <Protocol_Ident>   0        </Protocol_Ident>
    <Station_Type>     0        </Station_Type>
    <Hardware_Release>V1.1      </Hardware_Release>
    <Software_Release>V1.3      </Software_Release>
    ...
  </Unit-Definition-List>
</gsd-definiton>
```

Als Beispiel für die Einsatzmöglichkeiten, die XML bietet, soll kurz das von der Firma Microsoft definierte *BizTalk-Framework* [51] vorgestellt werden. Dabei handelt es sich um ein Messaging-Framework, bei dem *BizTalk-Server* spezielle, mittels XML-formatierte Nachrichten, die *BizTalk-Messages*, austauschen. Eine Nachricht besteht aus den folgenden Komponenten (vgl. Bild 15):

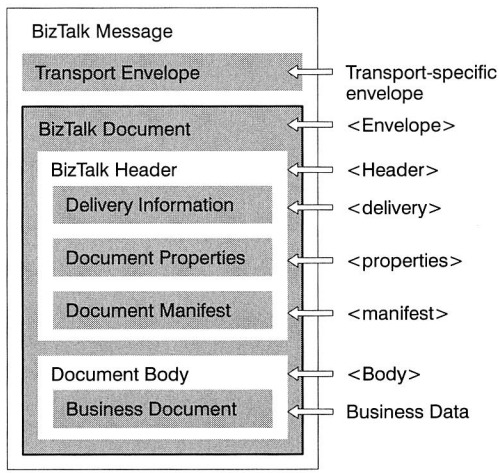


Bild 15: *Die Struktur einer BizTalk-Nachricht (nach [51])*

- *Transport Envelope:* Dieser "Rahmen", der die eigentliche Nachricht kapselt, ist spezifisch für das verwendete Transportprotokoll. Bei der Verwendung von HTTP wäre es ein HTTP-konformer Rahmen. Der Rest der Nachricht stellt dann den Inhalt der HTTP-Nachricht dar. Durch den Austausch des Rahmens kann auf einfachste Weise ein neues Transportprotokoll für das BizTalk-Framework spezifiziert werden.

- *BizTalk-Document:* BizTalk-Dokumente sind SOAP-konforme (Simple Object Access Protocol, [52]) Nachrichten, bestehend aus dem Nachrichtenkopf *(BizTalk-Header)* und dem Nachrichtenkörper *(Document Body).* Zur Strukturierung der einzelnen Elemente des Dokuments wird XML eingesetzt. In einer eigens definierten DTD ist festgelegt, welche Tags im Rahmen von BizTalk gültig sind und in welcher Reihenfolge sie im Dokument vorkommen müssen. Die Bedeutung (die "Semantik") der mittels bestimmter Tags geklammerten Informationen ist Gegenstand des BizTalk-Frameworks.
- *BizTalk-Header:* Er enthält im XML-Tag "<delivery>" die Adresse des Senders und des Empfängers der Nachricht, in "<property>" Dokumenteninformationen wie den Sendezeitpunkt sowie in "<manifest>" eine Beschreibung der mit dem Dokument gelieferten Informationen.
- *Document-Body:* Beinhaltet die eigentliche Nachricht.

2.5.3 Die Programmiersprache Java

Java ist eine Programmiersprache, deren Ursprünge auf das Jahr 1991 zurückgehen. Ein Team von Softwareingenieuren bei Sun Microsystems hatte die Entwicklung einer neuen Programmiersprache für die Steuerung einer in Planung befindlichen neuen Generation von programmierbaren Geräten für den privaten Haushalt und die Unterhaltungsindustrie (TV-Steuerungen, Videorekorder etc.) begonnen. Die Sprache mußte vor allem leicht portierbar sein, weil eine Vielzahl unterschiedlichster Steuerungssysteme und Anwendungen denkbar war. Mit dem Erfolg des WWW wurde aus der ursprünglich als "Oak" bezeichneten Sprache die objektorientierte Programmiersprache Java, und das Einsatzgebiet verlagerte sich hin zur Erstellung von Anwendungen für das Internet. Auf einige der relevanten Besonderheiten von Java wird nachfolgend genauer eingegangen, ansonsten sei an dieser Stelle auf die Standardliteratur verwiesen (z.B. [53], [54]).

Einmal erstellte Java-Programme sind prinzipiell auf beliebigen Plattformen, d.h. Hardware und Betriebssystemen, ablauffähig. Dies wird durch die Ausführung eines kompilierten Zwischencodes (*Bytecode*) in einem speziellen Interpreter, der *Java Virtual Machine* (JVM), erreicht. Einzige Voraussetzung ist damit die Verfügbarkeit einer JVM für die jeweilige Plattform. In den gängigen Web-Browsern ist eine JVM bereits integriert, so daß Java-Programme dort ebenfalls ausführbar sind (*Java-Applets*).

In den vergangenen Jahren entwickelte sich die Programmiersprache Java zu einem kompletten Middleware-Framework weiter. Seitens der Middleware wird dabei *Java-RMI* (Remote Method Invocation), eine auf die Kommunikation zwischen Java-Servern und Java-Clients beschränkte Middleware-Implementierung, angeboten sowie *Java-IDL*, das einer Java Sprachbindung für CORBA entspricht (siehe nächstes Kapitel). Für die Programmierung anwendungsspezifischer Frameworks definiert Java bereits zahlreiche Erweiterungen zu den Standardklassen des *Java Core API* (z.B. E-Com-

merce, Security). Als Framework für die Ausführung von Java-Komponenten wurde *JavaBeans* entwickelt und darauf aufbauend *Enterprise JavaBeans* (EJB). Die EJB-Spezifikation ist CORBA-konform, so daß EJB-Server und Clients auch mit nicht in der Programmiersprache Java erstellten Anwendungen auf Komponentenebene verteilt kooperieren können, sofern diese gemäß dem EJB-Komponentenmodell erstellt wurden. Bild 16 stellt die Java-Architektur im Überblick dar.

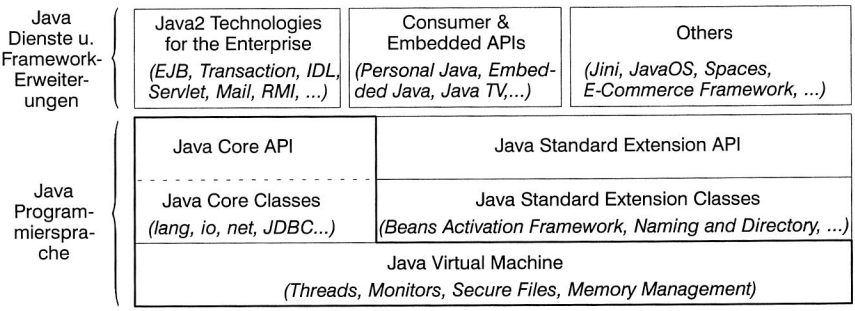


Bild 16: *Die Java-Architektur im Überblick*

2.5.4 Java-RMI

Das Objektmodell, das von Sun aufbauend auf der Programmiersprache Java spezifiziert wurde, ist hinsichtlich der Grundfunktionalität vergleichbar mit den bereits vorgestellten Architekturen CORBA und DCOM. Insbesondere der wechselseitige Einfluß von Java und CORBA hat in der Vergangenheit zu einem Abgleich der Schnittstellen für die in beiden Systemen gleichermaßen angebotenen Dienste geführt. In diesem Sinn bietet das Objektmodell keine wesentlichen funktionellen Erweiterungen, seine wesentliche Bedeutung liegt vielmehr in der Möglichkeit, daß Java-Anwendungen über das Internet verbreitet werden können, ohne daß eine Compilierung oder eine Installation notwendig ist.

Java-RMI ist eine Middleware-Implementierung zur Kommunikation zwischen örtlich verteilten Java-Anwendungen. Sie beinhaltet u.a. die automatische Generierung von client- und serverseitigen Java-Proxys sowie die Registrierung von Servern in einer speziellen Registrierungsdatenbank zur Lokalisierung durch potentielle Clients. Ein einfacher Namensdienst wird ebenfalls bereitgestellt. Im Vergleich zu CORBA oder DCOM können die in der Programmiersprache Java vorliegenden Proxys jedoch dynamisch über das Internet geladen und ohne Übersetzung direkt verwendet werden. Bei der Erstellung von CORBA- und DCOM-Clients müssen hingegen die typischerweise bei der Server-Implementierung erzeugten Proxys für jede mögliche Client-Plattform separat generiert und übersetzt werden.

Ein wesentlicher Nachteil von Java-RMI stellt die Beschränkung auf Java-Clients und Java-Server dar. Die angestrebte Konvergenz der Java- und CORBA-Technologien wird zum einen durch die Verwendung des CORBA *Internet Inter-ORB Protocol* (IIOP) als Transportprotokoll für Java-RMI erreicht. Zum anderen existiert mit Java-IDL die Möglichkeit, aus Java-RMI Servern über einen speziellen Compiler automatisch CORBA-IDL Schnittstellen zu erzeugen und diese damit in die CORBA-Umgebung einzubinden.

2.6 Sicherheit in verteilten Systemen

Der Funktions- und Informationsverbund innerhalb eines Unternehmens, der die koordinierte Abwicklung des Fertigungsprozesses überwacht, stellt ein hochgradig verteiltes System dar. Im Bereich der Prozeßebene umspannt es die dezentral agierenden Steuerungssysteme, wie in Kapitel 2.1.3 dargestellt. Neue Klassen von Anwendungen, wie die Ferndiagnose und Fernwartung, benötigen einen direkten Zugriff auf diese Steuerungssysteme von beliebigen Orten aus. Dabei erfolgt der Aufbau der Kommunikationsverbindungen nicht nur innerhalb des Firmennetzes, sondern auch über das Internet. Die Frage nach geeigneten Sicherheitskonzepten wird deshalb immer vordringlicher. In einer Arbeit, die sich im Kern mit dem Zugriff auf Geräte der Prozeßebene beschäftigt, muß das Thema "Sicherheit" daher auch angesprochen werden. Jedoch ist die Problematik insgesamt so komplex, daß sie nicht einfach nebenbei abgehandelt werden kann, sondern einer eigenen Untersuchung bedarf. Dies ist vielleicht auch der Grund, warum in verschiedenen Forschungsberichten zwar technische Lösungen für Fernzugriffe über das Internet präsentiert werden, die Frage nach der Sicherheit aber ausgeklammert wird.

Um ein besseres Verständnis zu gewinnen, wie umfangreich die in diesem Zusammenhang auftretenden Probleme sind, soll ein kurzer Überblick über die Sicherheit in verteilten Systemen gegeben werden. Zunächst ist zu klären, was unter dem Begriff "Sicherheit" genau verstanden werden soll. In Anlehnung an *Summers* [55] wird von folgender Definition eines sicheren Systems ausgegangen:

Definition 5: *Sicheres System*

Ein sicheres System weist drei Eigenschaften auf:

a) *Vertraulichkeit: Vertraulichkeit bedeutet, daß Informationen nur nach festen Regeln freigegeben werden.*

b) *Integrität: Integrität bedeutet, daß Informationen nicht zerstört oder manipuliert werden und das System ein korrektes Verhalten aufweist.*

c) *Verfügbarkeit: Verfügbarkeit bedeutet, daß die Systemdienste dann verfügbar sind, wenn sie benötigt werden.*

Sind diese drei Eigenschaften nicht gewährleistet, so spricht man umgekehrt von den drei Grundbedrohungen.

Um ein sicheres System zu erlangen, das vor allem auch den Aufwand und den Nutzen der betriebenen Sicherheitsvorkehrungen in einem angemessenen Rahmen hält, muß ein geeignetes *Sicherheitskonzept* erstellt werden. Hier hilft zunächst ein methodischer Ansatz, wie er im Rahmen der *Risikoanalyse* vorgeschlagen wird. Diese hat zur Aufgabe, die schützenswerten Objekte zu identifizieren, für jedes Objekt eine detaillierte Bedrohungsanalyse zu erstellen, um dann, über die Zuordnung von Eintrittswahrscheinlichkeiten und Schadenshöhe, eine Risikobewertung für jede mögliche Kombination aus "Objekt" und "Bedrohung" durchführen zu können. Ergebnis der Risikoanalyse ist damit eine Priorisierung der Risiken, auf deren Basis geeignete *Sicherheitsmaßnahmen* im Rahmen eines Sicherheitskonzepts definiert werden können.

Die zur Verfügung stehenden Sicherheitsmaßnahmen lassen sich klassifizieren in (vgl. [55], [56]):

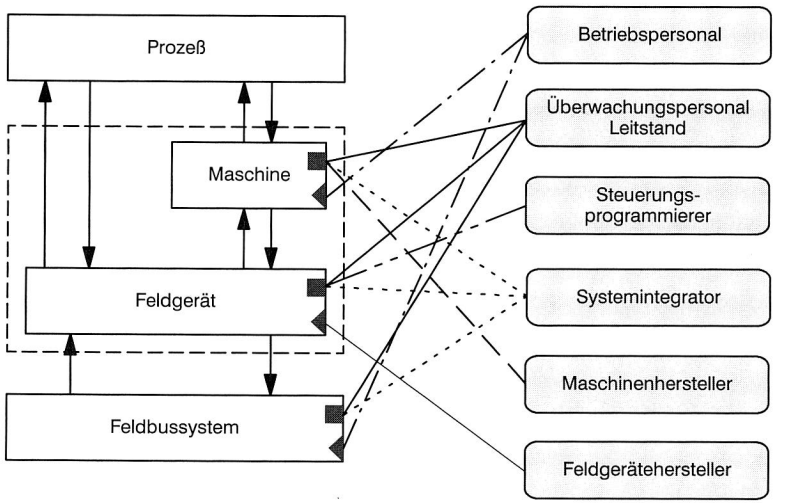
- präventive und vermeidende Maßnahmen, mittels derer eine Bedrohung im Vorfeld vermieden werden soll,
- überwachende Maßnahmen, mittels derer eine Bedrohung erkannt und abzuwehren versucht wird,
- reaktive Maßnahmen, mittels derer versucht wird, nach Eintritt der Bedrohung die Schadensfolgen zu minimieren.

Die Dienste, die ein System anbietet, um Sicherheit zu unterstützen, werden *Sicherheitsdienste* genannt. Einige wichtige Sicherheitsdienste sind nachfolgend aufgeführt (vgl. [57]):

- *Authentifikation:* Authentifikationsmechanismen prüfen die Identität von Personen, Organisationen, Programmen usw.
- *Zugriffskontrolle:* Die Zugriffskontrolle sorgt dafür, daß nur berechtigte Zugriffe auf Ressourcen stattfinden. Sie wird meist der Authentifikation nachgeschaltet.
- *Auditing:* Das Auditing zeichnet sicherheitsrelevante Ereignisse chronologisch auf, um so im Zuge der überwachenden Maßnahmen eingetretene Bedrohungen entdecken zu können.
- *Verschlüsselung:* Durch den Einsatz von Verschlüsselungstechniken wird versucht, die Vertraulichkeit und Integrität einer Nachricht zu gewährleisten.

Die Umsetzung eines Sicherheitskonzepts erfordert immer die Abbildung der realen Welt, d.h. der Regeln und Prinzipien, wie innerhalb eines Unternehmens Informationen geschützt werden, welche Personengruppen welche Rechte für welche Ressourcen besitzen usw. Hierbei kann es hilfreich sein, für ein zu schützendes Objekt verschiedene anwendungsspezifische *Rollen* der Subjekte (Personengruppen, Programme usw.) zu unterscheiden [55]. In Bild 17 ist ausgehend von einem Steuerungssystem und dessen bedrohten Objekten sowie für bestimmte Personengruppen beispielhaft dargestellt, wie ein Rollendiagramm aussehen könnte.

In der linken Hälfte ist die Steuerungsarchitektur mit den bedrohten Objekten abgebildet. Dabei wird von einer Anbindung über ein Feldbussystem ausgegangen, die ein Feldgerät realisiert. Der eigentliche Prozeß kann nun entweder direkt von einem Steuerungsprogramm des Feldgeräts aus beeinflußt werden, oder aber durch eine mit diesem verbundene Bearbeitungsmaschine. Jede in Software realisierte Funktion ist dabei prinzipiell ein bedrohtes Objekt – im Bild werden diese vereinfacht durch die Hardware repräsentiert. Rechts sind verschiedene Subjekte, genauer Personengruppen in verschiedenen Rollen, dargestellt. Je nach Rolle, besitzen sie bestimmte Rechte auf den Objekten. Exemplarisch werden Nur-Leserechte und Lese-/Schreib-/Ausführungsrechte unterschieden.



(a) bedrohte Objekte

(b) verschiedene Rollen der Subjekte "Personen"

■ *lesen, schreiben, ausführen*

◀ *lesen (z.B. Prozeßwerte, Diagnosedaten)*

Bild 17: *Subjekte, Objekte und deren Rechte in einem Sicherheitskonzept*

Von den universellen Feldbussystemen unterstützt nur PROFIBUS-FMS die Vergabe von Zugriffsrechten für Kommunikationsobjekte. Dabei können objektspezifisch Paßwörter für den lesenden und den schreibenden Zugriff vergeben werden, Zugriffsgruppen definiert werden und die Zugriffsrechte auf die Gruppen selbst festgelegt werden. Beim Einsatz anderer Feldbussysteme müssen vergleichbare Mechanismen somit erst implementiert werden. Für die anderen gezeigten Objekte existieren überhaupt keine vordefinierten Sicherheitsdienste.

Zusammenfassend betrachtet, stellt bereits die Umsetzung der Rollen in einem Sicherheitskonzept eine äußerst komplexe Aufgabenstellung dar. Hierbei sind die Abwehr konkreter Bedrohungen und die spezifischen, in Verbindung mit dem Internet zu berücksichtigenden Sicherheitsproblematiken noch gar nicht genauer dargelegt worden. Eine ergänzende Behandlung des Themas "Sicherheit" im Rahmen dieser Arbeit gestaltet sich insofern schwierig, da es keine grundlegenden Arbeiten im Umfeld der Fertigungsautomatisierung zu diesem Thema gibt, auf die verwiesen werden könnte. Umgekehrt würde damit *jede* Behandlung des Themas zwangsläufig auf eine grundlegende Betrachtung hinauslaufen, die damit den Rahmen der eigentlichen Arbeit sprengen würde. Sicherheitsfragen sollen deshalb im weiteren Verlauf explizit unberücksichtigt bleiben.

2.7 Zusammenfassung

Dieses Kapitel stellte zunächst die Struktur der Kommunikationssysteme innerhalb eines Unternehmens vor, um dann auf die sich ergebenden Probleme bei der Integration der Prozeßebene detailliert einzugehen. Die fortschreitende Dezentralisierung der Steuerungssysteme und der damit verbundene Einsatz intelligenter Feldgeräte bedingt in diesem Zusammenhang die Entwicklung und den Einsatz neuer Kommunikationskonzepte. Mit der Vorstellung der verteilten, objektorientierten Client/Server-Architekturen CORBA und COM sowie der mit dem Internet eng verbundenen Technologien XML und Java wurden die wesentlichen Grundlagen beschrieben, die für die Konzeption einer umfassenden Lösung notwendig sind. Die Frage nach einem geeigneten Sicherheitskonzept beim Zugriff auf Daten und Funktionen von Feldgeräten wurde als äußerst wichtig jedoch für eine Behandlung im Rahmen der Arbeit als zu komplex erkannt.

3 Anforderungsanalyse und Bewertung aktueller Systeme und Modelle

Dieses Kapitel formuliert die qualitativen Anforderungen, die eine umfassende technische Lösung zur Integration der Prozeßebene in die Informationsverarbeitung eines Unternehmens erfüllen muß. Gemäß der im vorherigen Kapitel eingeführten Begriffe wird hinsichtlich der Anforderungen unterschieden zwischen der Heterogenität auf Ebene der Implementierung und auf Ebene der Anwendungslogik.

Im Anschluß wird der derzeitige Stand der Technik für beide Ebenen genauer betrachtet. Problematisch ist allerdings die daraufhin folgende Beurteilung an Hand der formulierten Anforderungen. Die vorhandenen Technologien wurden unter gänzlich anderen Zielvorstellungen entwickelt und erfüllen die Zwecke für die sie konzipiert wurden im allgemeinen recht gut. Der Integrationsgedanke spielte bei der Entwicklung, wenn überhaupt, nur am Rande eine Rolle. Mit der Bewertung sollen daher auch lediglich die Schwachpunkte der alten Technologien unter den neuen Bewertungskriterien aufgezeigt werden. Die dabei gewonnenen Erkenntnisse fließen in die Kapitel 4 bis 6 ein, in denen versucht wird, eng angelehnt an die bestehenden Lösungen, durch geeignete technologische Erweiterungen, der Erfüllung der Anforderungen einen wesentlichen Schritt näher zu kommen.

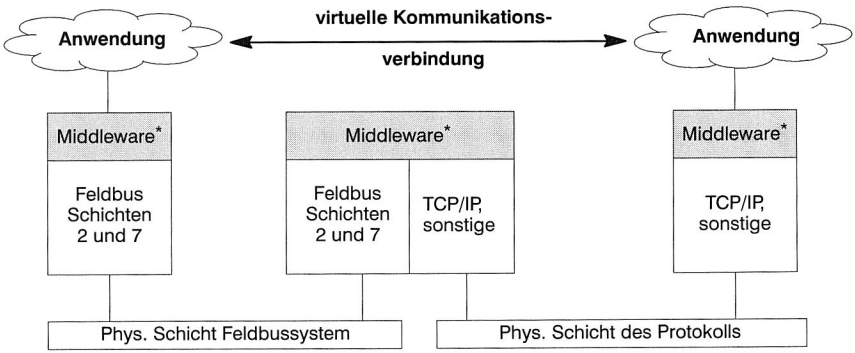
Im einzelnen werden exemplarisch für die universellen Feldbussysteme alle drei Varianten des PROFIBUS genauer untersucht. Seitens der Modelle erfolgt eine allgemeine Darstellung der Gerätebeschreibungssprachen, wobei dann im Detail die Konzepte der Gerätebeschreibungsdateien, der "Electronic Device Description" und des "Field Device Tool" betrachtet werden. Den Abschluß bildet die Vorstellung von "OLE for Process Control".

3.1 Grundlegende Anforderungen

Zur Überwindung der Heterogenität auf Implementierungsebene ist ein Systemkonzept erforderlich, das von den Unterschieden in den vorhandenen Betriebssystemen, Hardwareplattformen, Kommunikationsprotokollen und Programmiersprachen abstrahiert und dabei sowohl in der Prozeßebene als auch in den höheren Ebenen der Kommunikationshierarchie eingesetzt wird. Im Kontext der Begriffsdefinitionen aus Kapitel 2.2 handelt es sich bei dem erforderlichen Systemkonzept zunächst um Middleware, deren Hauptaufgabe die Bereitstellung einer virtuellen Kommunikationsverbindung zwischen den beteiligten Kommunikationspartnern ist (vgl. Bild 18).

Die Einheitlichkeit des Systemkonzepts ist eine wichtige Voraussetzung für einen aus Anwendungssicht transparenten Übergang zwischen den Kommunikationsebenen, die unterschiedliche Protokolle und physikalische Übertragungsverfahren aufweisen. Dadurch wird aber notwendigerweise auch die Komplexität der angestrebten Lösung

eingeschränkt, da hinsichtlich der verfügbaren Ressourcen und Geräteeigenschaften immer von der Schnittmenge aller denkbaren Geräte auszugehen ist. Für die im Rahmen dieser Arbeit betrachteten Feldgeräte soll eine Mindestausstattung von 256 kByte Speicher, das Vorhandensein eines Single-Task Betriebssystem sowie einer Kommunikationsanbindung über ein Feldbussystem angenommen werden.



spezifische Gerätetreiber sowie das Betriebssystem sind nicht dargestellt

Bild 18: *Prinzip der virtuellen Kommunikationsverbindung zwischen zwei Anwendungen*

Der Entwurf eines geeigneten Middleware-Konzepts muß diese Randbedingungen beachten, damit die Implementierung hinsichtlich der Ressourcen nicht mit der eigentlichen Steuerungsaufgabe des Geräts, der die oberste Priorität zuzuordnen ist, in Konflikt gerät. Middleware setzt, wie in Kapitel 2.2 dargelegt, auf dem Betriebssystem und dem vorhanden Kommunikationssystem auf. Eine leichte Integrierbarkeit der Lösung in bestehende Systeme soll es in diesem Zusammenhang ermöglichen, auf der vorhandenen Systemumgebung aufbauen und mit nur geringen Modifikationen die neue Funktionalität anbieten zu können. Für die Umsetzung einer Migrationsstrategie ist der parallele Betrieb von herkömmlichen Feldbus-Anwendungen neben den funktional erweiterten Anwendungen wünschenswert. Dabei darf jedoch keinesfalls der unter Echtzeitbedingungen ablaufende Produktivdatenverkehr durch die Nutzung der zusätzlichen Kommunikationsdienste beeinträchtigt werden.

3.2 Anforderungen an die Implementierungsebene

Prinzipiell kann die Kommunikation zweier Anwendungen nach deren Zugehörigkeit zu den verschiedenen Kommunikationsebenen eingeteilt werden. Im Rahmen dieser Arbeit sind die beiden in Bild 19 dargestellten Fälle von zentraler Bedeutung, bei de-

nen die Prozeßebene beteiligt ist. Die nachfolgend formulierten Anforderungen greifen diese auf und bestimmen den notwendigen Funktionsumfang der anwendungsunabhängigen Middleware-Dienste.

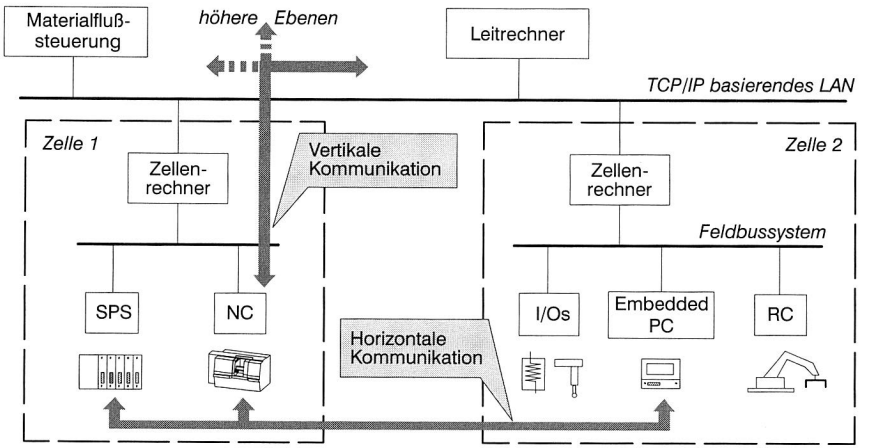


Bild 19: *Unterscheidung der horizontalen und vertikalen Kommunikation unter Beteiligung der Prozeßebene*

Anforderung K1: Unterstützung der vertikalen Kommunikation

Spricht man von der Integration der Prozeßebene, so meint man zumeist die Unterstützung der vertikalen Kommunikation. Dies liegt sicherlich an der Zahl und der Bedeutung der Anwendungen, die auf eine derartige Verbindung angewiesen sind. Allgemein sind dies zunächst alle übergeordneten steuernden und überwachenden Anwendungen in der Zellen- und Leitebene. Beispielsweise benötigt ein Zellenrechner zur Steuerung des zelleninternen Materialflusses Kenntnis von der genauen Position der Werkstückträger und zur Synchronisation der Bearbeitungsgeräte die Information, wann und mit welchem Ergebnis ein Bearbeitungsschritt beendet ist. In der Leitebene laufen die Informationen der einzelnen Zellenrechner insbesondere über den momentanen Arbeitsfortschritt zusammen. Diese Informationen dienen als Grundlage für kurzfristige, tagesaktuelle Planungen und werden u.a. im Zuge der Einplanung von Eilaufträgen benötigt. Darüber hinaus findet hier die Überwachung der kompletten Fertigungslinie statt, indem die aktuellen Fertigungszustände, aktuelle Maschinendaten und Störmeldungen dem Bedienpersonal in graphisch aufbereiteter Form präsentiert werden. In den dispositiven Bereichen kann es im Rahmen der betriebsbegleitenden Simulation sinnvoll sein, aktuelle Betriebsparameter direkt in die laufenden Simulationsberechnungen zu integrieren. Ein weiteres Anwendungsgebiet, das auf der vertikalen Kommunikation aufbaut, ergibt sich in Verbindung mit der Nutzung des In-

ternet zur Fernwartung und Ferndiagnose komplexer technischer Anlagen. Auch hier ist der Zugriff auf die internen Parameter einer Maschine über das im Prozeß vorhandene Kommunikationssystem das Ziel.

Die zu entwickelnde Kommunikationsarchitektur muß somit den Zugriff einer Anwendung außerhalb der Prozeßebene über das Feldbussystem auf ein Feldgerät unterstützen. Umgekehrt muß auch ein Feldgerät in der Lage sein, beispielsweise eine auftretende Störung direkt an den Leitrechner zu melden. Es wird daher die Unterstützung für eine bidirektionale vertikale Kommunikation gefordert. Aus Sicht der Anwendung ist es wichtig, daß von den verschiedenen Protokollen und physikalischen Übertragungsverfahren abstrahiert wird. Für einen Anwender muß sich die vertikale Kommunikation wie eine virtuelle Punkt-zu-Punkt Verbindung darstellen, über die er direkt mit der Gegenseite kommunizieren kann.

Anforderung K2: Unterstützung der horizontalen Kommunikation

Eine umfassende Integration der Prozeßebene kommt nicht umhin, sich mit einer sehr speziellen Problematik auseinanderzusetzen: der horizontalen Kommunikation zweier Feldgeräte, die an verschiedenen und eventuell unterschiedlichen Bussystemen innerhalb der Prozeßebene eingesetzt sind. In der Praxis wird man versuchen, die einzelnen Fertigungszellen so auszulegen, daß *ein* Feldbussystem genügt, um den Informationsaustausch der Feldgeräte untereinander bzw. mit dem Zellenrechner zu realisieren. Dennoch kann es vorkommen, daß eine bestimmte Gruppe von Feldgeräten ein spezielles Kommunikationssystem erfordert, oder eine zeitkritische Datenübertragung innerhalb einer Gruppe von Busteilnehmern nur dann sicher gewährleistet ist, wenn sich ausschließlich diese Teilnehmer an einem separaten Bussystem befinden. In solchen oder ähnlichen Szenarien ist es erforderlich, zwei oder mehr Bussysteme innerhalb einer Zelle einzusetzen, wobei jedoch weiterhin alle Feldgeräte der Zelle untereinander koordiniert werden müssen.

Andererseits ist es möglich, daß zwei Feldgeräte benachbarter Fertigungszellen miteinander kommunizieren müssen, wenn beispielsweise eine direkte Informationsweitergabe an die nachfolgende Bearbeitungsstation notwendig ist. Dazu muß nicht nur eine Kommunikationsverbindung zwischen den Zellenrechnern aufgebaut werden, sondern es müssen auch die beteiligten Rechner für den Austausch der Daten entsprechend konfiguriert werden. Dies erweist sich als besonders problematisch, wenn hierbei speicherprogrammierbare Steuerungen, eventuell sogar verschiedener Hersteller, beteiligt sind.

Zusammenfassend betrachtet ist der Implementierungsaufwand für eine Einzellösung zur Realisierung der horizontalen Kommunikation in der Prozeßebene in jedem Fall beträchtlich. Da es sich zudem um eine herstellerspezifische Lösung handelt, ist sie schlecht wiederverwendbar.

Wesentlich eleganter ist es, wenn die Middleware diese Art der Kommunikation über verschiedene Bussysteme hinweg unterstützt. Dabei wird wiederum eine bidirektio-

nale Kommunikation gefordert, wobei aus Sicht der einzelnen Feldgeräte eine virtuelle Kommunikationsverbindung zum jeweils anderen Gerät bestehen soll.

Anforderung K3: Unterstützung für ein verteiltes Client/Server-Modell

Die beiden vorausgegangenen Anforderungen betrafen die Unterstützung horizontaler und vertikaler virtueller Kommunikationsverbindungen. Damit wurde noch keine Aussage getroffen über das Kommunikationsmodell, das zwischen den beteiligten Anwendungen zum Einsatz kommt. Angesichts der genannten Anwendungszenarien scheidet ein *Publisher-Subscriber* Modell für sich alleine jedoch aus. Dieses ist dadurch gekennzeichnet, daß ein Empfänger bei einer Nachrichtenquelle bestimmte Nachrichten abonnieren kann, die er dann automatisch zugesandt bekommt. Das Publisher-Subscriber Modell würde für die Mitteilung von Prozeßwertänderungen oder das Absetzen von Störmeldungen ausreichend sein, jedoch müssen beispielsweise bei der Ferndiagnose die jeweiligen Diagnosefunktionen aufgerufen werden können oder im Rahmen der Ermittlung der Kenngrößen einer Maschine diese Werte einmalig ausgelesen werden können. Für diese Art von Anwendungen, die eine bestimmte Funktionalität des Kommunikationspartners nutzen wollen, ist das Client/Server-Modell ideal. Es ermöglicht einem Feldgerät seine Funktionalität in Form von Diensten anzubieten, so daß sie ein Client über eine Dienstanforderung nutzen kann. Umgekehrt können auch Feldgeräte die Dienste von Anwendungen außerhalb der Prozeßebene anfordern und so im Störungsfall entsprechende Meldungen an übergeordnete Anwendungen absetzen. Daher wird die Unterstützung für ein verteiltes Client/Server-Modell auf Basis der virtuellen Kommunikationsverbindungen gefordert.

3.3 Anforderungen an die Ebene der Anwendungslogik

Die Anforderungen, die an die Ebene der Anwendungslogik gestellt werden, bestimmen den Funktionsumfang, der zur gezielten Unterstützung der Integration von Anwendungen notwendig ist. *Wedekind* [10] schlägt hier die Definition eines allgemein verwendbaren Datenschemas für ein bestimmtes Anwendungsgebiet vor, darauf aufbauend die Festlegung einer Programmierschnittstelle und eine softwaretechnische Realisierung in Komponententechnologie. Dieser Ansatz wird mit den nachfolgend formulierten Anforderungen etwas modifiziert. Es wird von einem generischen Modellkonzept ausgegangen, das zudem selbstbeschreibend ist. Die angebotenen Programmierschnittstellen sollen dabei nicht fest vorgegeben werden, sondern zur Laufzeit abrufbar sein.

Anforderung M1: Umfassendes, generisches Modellkonzept

Die Dienste, die ein Feldgerät anbietet, werden normalerweise nicht einfach über eine ungeordnete Sammlung von Schnittstellen angeboten, sondern sind typischerweise in ein Gerätemodell eingebettet. Obwohl die Gerätemodelle feldbusspezifisch sind, besitzen sie eine gewisse Ähnlichkeit, da sie in der Regel von MMS abstammen. Wenn

nun eine Anwendung auf die Dienste eines Feldgeräts zugreifen will, so muß sie dessen Modell genau kennen. Eine Lösung wäre die Definition eines feldbusneutralen Modells, was jedoch in der Praxis kaum durchsetzbar sein dürfte. Eine weitere Lösung besteht darin, überhaupt erst eine geeignete, gemeinsame Modellierungssprache festzulegen, in der dann nicht nur das jeweilige Gerätemodell, sondern auch beliebige Anwendungsmodelle beschrieben werden können. Das in Verbindung mit der Sprache benutzte *Modellkonzept* muß also umfassend und generisch sein. In Zusammenhang mit der Anforderung M4, die die Fähigkeit zur Selbstbeschreibung fordert, kann dann ein Client das Gerätemodell und das Anwendungsmodell zur Laufzeit erkunden und muß dies vorab nicht mehr explizit kennen.

Anforderung M2: Erweiterbarkeit

In dem sich ständig ändernden Produktionsumfeld ist es zweifelhaft, daß Gerätemodelle und prozeßnahe Daten- und Schnittstellenmodelle über einen längeren Zeitraum unverändert bleiben.

- Neue Softwareversionen eines Feldgeräts und neue Gerätegenerationen beinhalten in der Regel eine erweiterte Funktionalität.
- Der Austausch von Maschinen oder die Aktualisierung der zugehörigen Steuerungssoftware hat meist ebenfalls Änderungen der Funktionalität in den Feldgeräten zur Folge.
- Neue Anwendungsfelder, wie sie durch die Ferndiagnose und Fernwartung schrittweise erschlossen werden, bedingen zusätzliche Schnittstellen, um – je nach konkretem Anwendungsfall – die relevanten Informationen über das Kommunikationssystem, die Feldgeräte oder die Maschinen abrufen zu können.

In den genannten Beispielen sind zusammen mit der neuen bzw. erweiterten Funktionalität stets auch Modelländerungen bzw. Modellerweiterungen notwendig. Das in der Anforderung M1 geforderte Modellkonzept muß daher erweiterbar sein, um neu auftretende Modellierungsaspekte berücksichtigen und ohne Änderungen der zugrunde liegenden Modellierungssprache abbilden zu können.

Anforderung M3: Leichte Integrierbarkeit in bestehende Anwendungen

Mit der Forderung nach leichter Integrierbarkeit ist die einfache Nutzung der von einem Feldgerät angebotenen Dienste in bestehenden Anwendungen gemeint. Muß der Client dazu explizite Kenntnisse des Gerätemodells besitzen, so zählt der Aufwand um diese zu Erlangen ebenso wie der Aufwand, um einen Dienst anzusprechen. Insofern genügt es nicht, bei der Implementierung Komponententechnologie einzusetzen, da diese zwar eventuell den Aufwand für die *technische Integration* verringern hilft, aber damit noch nichts über die Einfachheit der Integration des Dienstes auf *semantischer Ebene* ausgesagt ist.

Anforderung M4: Selbstbeschreibung

Der Aufwand für die Bereitstellung und Nutzung neuer Daten- und Schnittstellenmodelle liegt wesentlich im Programmieraufwand begründet, da oft vielfältige Anpassungen vorgenommen werden müssen. Das Problem entsteht im Kern dadurch, daß ein und dieselben Informationen von verschiedenen Maschinen und Softwaresystemen auf unterschiedliche Weise angeboten werden, da keine allgemeingültigen Modelle existieren. Die Informationen werden unterschiedlich organisiert, die Schnittstellen, obwohl sie funktional äquivalent sein können, sind anders bezeichnet, haben verschiedene Parameter-Reihenfolgen oder verwenden jeweils andere Datenformate. Eine Lösung diese Problematik zu umgehen, liegt in der Einführung von Metadaten [58]. Diese beschreiben das Gerätemodell und das Anwendungsmodell, so daß ein Client es zur Laufzeit erkunden kann. Der Client kann zudem abfragen, welche Schnittstellen vorhanden sind, welche Parameter bei einem Aufruf erwartet werden und wie diese aufgebaut sein müssen. Er erhält die gleichen Informationen auch über die Rückgabewerte. Damit ist auf Anwendungsebene nicht mehr ein einheitliches Modell Voraussetzung für die Kooperation von Client und Server.

3.4 Vorstellung und Bewertung aktueller Technologien

In diesem Abschnitt wird der Stand der Technik im Bereich der Feldbussysteme und der Modellierung detailliert vorgestellt. Wie bereits angesprochen, ist es schwierig, eine abschließende Bewertung hinsichtlich der gestellten Anforderungen vorzunehmen, da jede dieser Technologien für ein ganz bestimmtes Einsatzgebiet entwickelt wurde, jedoch nicht mit dem primären Ziel, die Integration der Prozeßebene zu unterstützen. Dennoch ermöglicht gerade diese Bewertung es, die bestehenden Defizite zu verdeutlichen, um dann in den folgenden Kapiteln 4 bis 6 geeignete technologische Erweiterungen der Systeme vorzuschlagen.

Auf Implementierungsebene werden exemplarisch für die universellen Feldbussysteme alle drei Varianten des PROFIBUS vorgestellt. Seitens der Modelle werden zunächst allgemein die verschiedenen Gerätebeschreibungen betrachtet. Da diese stets herstellerspezifisch sind, werden die Konzepte der Gerätebeschreibungsdateien, der Electronic Device Description und des Field Device Tool nochmals gesondert herausgegriffen und genauer beschrieben. Zum Schluß erfolgt die Darstellung von OLE for Process Control.

3.4.1 Das Feldbussystem PROFIBUS

Überblick über PROFIBUS

Das Feldbussystem PROFIBUS (Process Field Bus) wurde im Rahmen eines vom damaligen Bundesministerium für Forschung und Technik geförderten Verbundprojek-

tes von vierzehn Herstellern und fünf technisch-wissenschaftlichen Instituten entwickelt. Dabei stand die Abdeckung einer möglichst großen Zahl von Anwendungsgebieten im Mittelpunkt. Es erfolgte 1991 die Verabschiedung als deutsche Norm DIN 19254 und 1996 die Verabschiedung als europäische Norm EN 50170-2. Insgesamt existiert der PROFIBUS in den drei Varianten FMS (Fieldbus Message Specification), DP (Dezentrale Peripherie) und PA (Prozeßautomatisierung).

Die Variante FMS ist speziell für die ereignisgesteuerte Kommunikation komplexerer Automatisierungsgeräte in der Zellenebene ausgelegt. Sie ist objektbasiert und bietet im Dienstumfang eine Teilmenge der in MMS angebotenen Dienste. Mit der Variante DP wurde versucht, den PROFIBUS auch im Bereich der dezentralen Ein-/Ausgabe kostengünstig einzusetzen. Dazu wurde eine gegenüber FMS weiter reduzierte Schichtenarchitektur (nur Schichten 1 und 2) definiert, die schnelle Übertragungsraten bis zu 12 Mbit ermöglicht. Schließlich wurde die Variante PA für das Anwendungsgebiet der Prozeßautomatisierung entworfen, die u.a. Eigensicherheit und Stromversorgung über den Bus bietet. Die Protokollarchitektur der verschiedenen Ausprägungen des PROFIBUS ist in Bild 20 dargestellt.

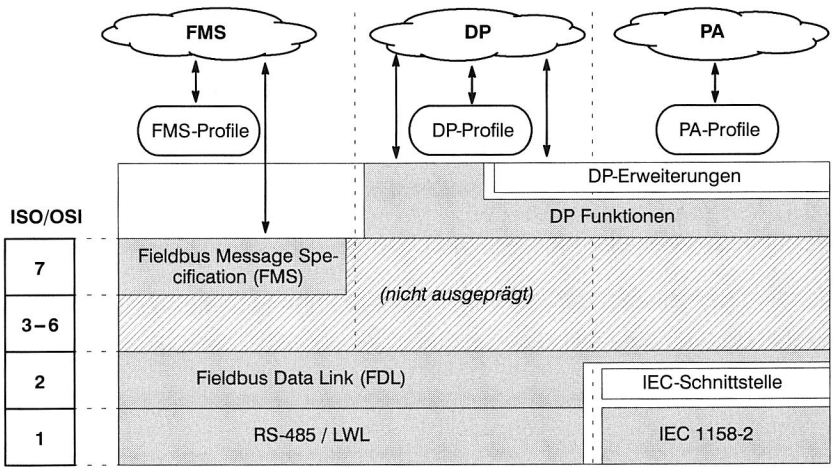


Bild 20: *Protokollarchitektur von PROFIBUS-FMS, -DP und -PA (nach [59])*

Eigenschaften der Sicherungsschicht

PROFIBUS unterscheidet Master-Geräte und Slave-Geräte. Erstere bestimmen den Datenverkehr auf dem Bus, sobald sie im Besitz der Buszugriffsberechtigung (Token) sind. Die Buszugriffsberechtigung wird zwischen den verschiedenen Master-Geräten, die am Feldbus angeschlossen sind, in Form eines logischen Tokenrings weitergereicht. Slave-Geräte sind typischerweise Peripheriegeräte wie Ein-/Ausgabegeräte,

Ventile oder Antriebe. Für das weitere Verständnis der PROFIBUS Protokollfamilie und der in den unterschiedlichen Varianten angebotenen Dienste, ist eine detailliertere Betrachtung der Sicherungsschicht (FDL) notwendig. Diese arbeitet verbindungslos und ermöglicht neben der Punkt-zu-Punkt Datenübertragung auch Broadcast- und Multicast-Kommunikation. Dabei werden drei azyklische und ein zyklischer Dienst angeboten (vgl. Tab. 1).

Dienst	Funktion	DP	FMS
SDA	Daten senden mit Quittung / Send Data with Acknowledge		•
SDR	Daten senden und empfangen mit Quittung / Send and Request Data with Reply	•	•
SDN	Daten unquittiert senden / Send Data with no Acknowledge	•	•
CSRD	zyklisches Senden und Empfangen von Daten / Cyclic Send and Request Data with Reply		•

Tab. 1: *Die Dienste der PROFIBUS Sicherungsschicht*

Eigenschaften der Anwendungsschicht

In der Variante DP stehen die in der Tabelle aufgeführten Dienste zum zyklischen Datenaustausch von E/A-Daten zwischen Master und Slave zur Verfügung. Dazu sind entsprechende Lese- und Schreiboperationen definiert, wobei die Adressierung nicht über logische Variablennamen, wie in FMS, sondern gerätespezifisch über die Angabe von *Slot* und *Index* erfolgt. Die Slot-Nummer adressiert dabei die in einem Gerät verfügbaren E/A-Module, der Index die im Modul vorhandenen Datenblöcke. Hinsichtlich der Feldbuskommunikation erweist es sich als Nachteil, daß nur Master-Geräte die Kommunikation mit anderen Geräten anstoßen können (fehlende Möglichkeit der *Slave-Initiative*). Slave-Geräte können bei PROFIBUS-DP nur empfangene Nachrichten quittieren oder auf Anfrage eines Masters Nachrichten senden. Zudem können nicht zwei verschiedene Master auf ein und den selben Slave schreibend zugreifen. Dies soll einen konkurrierenden Schreibzugriff mit der Folge möglicher Inkonsistenzen verhindern. Intelligente Feldgeräte, die selbst Vorverarbeitungsaufgaben übernehmen und daher nur ereignisgesteuert mit der überlagerten Steuerung kommunizieren, müssen damit fast zwangsläufig als Master-Geräte konfiguriert werden. Für die Kommunikation zweier Master stehen jedoch nur administrative Dienste zur Verfügung. Diese umfassen Funktionen zum Laden und Aktivieren von (Parameter-) Datensätzen sowie zum Lesen von Diagnosedaten.

Mit speziellen azyklischen Erweiterungen, die 1998 insbesondere für den PROFIBUS-PA definiert wurden [60], stehen nunmehr bei neueren DP-Geräten Kommunikationsdienste für den azyklischen Datenverkehr zwischen Master- und Slave-Geräten zur Verfügung. Interessant ist in diesem Zusammenhang der Dienst *MSAC2_Data_Transport*, der es einem Master erlaubt, in einer Übertragung azyklisch Daten an einen

Slave zu senden und sofort eine Antwort von diesem zu empfangen. Ein erweitertes Adressierungsmodell erlaubt es dabei erstmals, beim Verbindungsaufbau *(MSAC2_Initiate)* verschiedene Programminstanzen *(API, Application Process Instance)* in einem Feldgerät zu unterscheiden und gezielt anzusprechen. Dieser eigentlich für die PROFIBUS-PA Kommunikation mittels *Process Function Blocks* entwickelte Dienst, wird u.a. herstellerspezifisch zur Realisierung synchroner *RPC*-Aufrufe genutzt. Auch bei dieser Lösung erweisen sich jedoch die feste Adressierungsart, die fehlende Slave-Initiative und die systemnahen Schnittstellen als zu inflexibel und zu wenig komfortabel für eine breite Einsetzbarkeit.

Mit der Variante FMS stehen, angelehnt an MMS, anwendungsseitig komfortable Kommunikationsdienste zur Verfügung. Diese umfassen (vgl. [9])

- die Verbindungsverwaltung (*Context Management*)
- die Übertragung von Datenblöcken *(Domain Management)*
- die Unterstützung für virtuelle Feldgeräte *(VFD Support)*
- den Zugriff auf Variablen *(Variable Access)*
- die Behandlung von Ereignissen *(Event Management)*
- Dienste für die Verwaltung von Objektverzeichnissen *(OD Management)*

Die von FMS bereitgestellten verbindungsorientierten Kommunikationsdienste erlauben es, Punkt-zu-Punkt Verbindungen zwischen Anwendungsprozessen aufzubauen. Die eigentliche Kommunikation wird auf logischer Ebene über selbstdefinierbare Kommunikationsobjekte abgewickelt. Dabei sind die Verbindungsarten Master-Slave zyklisch/azyklisch und Master-Master azyklisch möglich. Indem bei FMS auch Slave-Geräte die Möglichkeit haben, Daten aktiv zu übertragen und untereinander direkt zu kommunizieren, sind die bei PROFIBUS-DP bzw. -PA genannten Nachteile befriedigend gelöst. In der Praxis erweist es sich jedoch als problematisch, daß nicht alle in FMS spezifizierten Dienste tatsächlich implementiert werden. Dies betrifft insbesondere die Dienste zum Laden von Datenblöcken und zum Starten und Stoppen von Anwendungen.

Bewertung

Nachfolgend wird eine Bewertung[3] der drei Varianten des PROFIBUS hinsichtlich der Anforderungen K1 bis K3 vorgenommen:

Anforderung K1: Unterstützung der vertikalen Kommunikation 

Am Übergang zur Zellenebene erfolgt eine Entkopplung der Kommunikation mit dem Feldbussystem derart, daß außerhalb des PROFIBUS angesiedelte Clients keinen di-

3. Die Bewertung wird graphisch folgendermaßen dargestellt:

 nicht erfüllt – teilweise erfüllt – vollständig erfüllt

rekten Zugriff auf die am Feldbus angeschlossenen Geräte haben. Im Arbeitskreis "PROFIBUS on Ethernet" der PROFIBUS Nutzerorganisation wird seit Mitte 1998 jedoch an einer Lösung dieses Problems gearbeitet. Dabei ist vorgesehen, eine in drei Stufen erfolgende Integration des PROFIBUS-DP in TCP/IP-basierte Netze und insbesondere das Internet zu definieren [61]. In der ersten Stufe ist der Zugriff mittels RPCs von in der Leitebene angesiedelten Clients auf die Geräte vorgesehen. Die zweite Stufe soll, durch Unterstützung des HTTP-Protokolls über PROFIBUS, eine weitergehende Integration in das Internet realisieren. Die dritte Stufe, die frühestens im Jahr 2002 verfügbar sein soll, wird schließlich eine durchgängige, bidirektionale Kommunikation ermöglichen. Erst mit der zuletzt genannten Lösung würde die geforderte virtuelle Kommunikationsverbindung adäquat unterstützt werden, da dabei ein als Slave agierendes Feldgerät selbst die Initiative zur Kommunikation übernehmen kann. Für die PROFIBUS Variante FMS sind derzeit keine vergleichbaren Erweiterungen geplant.

Anforderung K2: Unterstützung der horizontalen Kommunikation 

Das PROFIBUS Protokoll bietet standardmäßig keine Unterstützung für die horizontale Kommunikation zweier getrennter PROFIBUS Netze unter Beteiligung eines Zellen- oder Leitrechners als Vermittlungsrechner. Es existieren lediglich herstellerspezifische Lösungen, die auf dem Prinzip des *Tunneling* der PDUs (Protocol Data Units) basieren.

Anforderung K3: Unterstützung für ein verteiltes Client/Server-Modell 

Der Aufbau einer virtuellen Punkt-zu-Punkt Verbindung ist bei PROFIBUS nur zwischen Geräten möglich, die am selben Feldbus angeschlossen sind. In DP stehen dann die vorgegebenen Dienste der Anwendungsschicht zur Verfügung. Komplexere, selbstdefinierte Funktionen eines Feldgeräts können – wie bereits dargelegt – nur über ein vom Anwender selbst zu entwickelndes proprietäres Protokoll im Rahmen des *MSAC2_Data_Transport* bei PROFIBUS-PA angesprochen werden. FMS bietet ein eingeschränktes MMS-basierendes Objektmodell, wobei nur die dort vordefinierten Dienste nutzbar sind.

3.4.2 Modellierung von Feldgeräten durch Gerätebeschreibungssprachen

Überblick über die Gerätebeschreibungssprachen

An einem Feldbus werden üblicherweise Geräte von verschiedenen Herstellern betrieben, die sich hinsichtlich ihrer Leistungsfähigkeit und Funktionalität mehr oder weniger stark voneinander unterscheiden können. Wichtige Charakteristika sind in diesem Zusammenhang die Anzahl der E/A-Signale, das Vorhandensein bzw. der Umfang von Diagnosemeldungen und die möglichen Busparameter wie Baudrate und Zeit-

überwachungen. Diese sind individuell je nach Gerätetyp und Hersteller und werden im Gerätehandbuch dokumentiert. Die elektronische Form des Gerätehandbuchs stellt die Gerätebeschreibungsdatei *(Device Description)* dar. Existierende Gerätebeschreibungssprachen sind beispielsweise die *Device Description Language* des HART-Protokolls (HART-DDL), die *Device Description Language* der Fieldbus Foundation (FF-DDL) oder die firmenspezifische *Device Description and Offline Simulation* (DDOS) von Endress + Hauser [62,63].

Der Nutzen der Gerätebeschreibungsdatei liegt darin, daß sie in Projektierungs-, Diagnose- und Inbetriebnahmewerkzeuge sowie in Leitsysteme eingelesen werden kann. Dabei wird ein Interpreter-Konzept verfolgt, d.h. die Gerätebeschreibung wird ausgewertet und je nach Werkzeug unterschiedlich dargestellt bzw. aufbereitet. Grundsätzlich ist die Abbildung folgender Informationen in der Gerätebeschreibungsdatei sinnvoll [63]:

- Kommunikationsverhalten des Geräts
- durch das Feldgerät unterstützte Funktionen
- gerätespezifische Parameter
- interne Abhängigkeiten der gerätespezifischen Parameter
- Anzeige- und Bedienoberfläche

Heutige Feldbussysteme unterstützen in ihren jeweils spezifischen Gerätebeschreibungen praktisch alle der aufgeführten Punkte mit Ausnahme der Gerätebedienung. Es werden jedoch in den jeweiligen Nutzerorganisationen Anstrengungen unternommen, feldbusübergreifende Standards zu entwickeln (vgl. [64]).

Ein anderer Lösungsansatz für die Integration von Feldgeräten in verschiedene Anwendungssysteme der Zellen- und Leitebene wird mit der Entwicklung und Bereitstellung von vorgefertigten Softwarekomponenten verfolgt. Dabei liefert der Hersteller keine beschreibenden Informationen, sondern bereits ausführbare Programmteile, deren Integration in bestehende Software über vordefinierte Schnittstellen erfolgt. Denkbar ist in diesem Zusammenhang auch, daß die Software-Komponenten weitgehend automatisch aus der Gerätebeschreibungsdatei selbst erzeugt werden (vgl. das FDT-Konzept weiter unten).

Da die verschiedenen Konzepte, die auf dem Gebiet der Gerätebeschreibungssprachen und der Feldgeräteintegration in Leitsysteme existieren, allesamt noch herstellerspezifisch sind, wird nun auf die Lösungen, die PROFIBUS anbietet, detaillierter eingegangen.

Gerätebeschreibungsdatei des PROFIBUS

Beim PROFIBUS werden die charakteristischen Kommunikations- und Leistungsmerkmale eines Gerätes in Form eines elektronischen Gerätedatenblatts (Geräte-

stammdatendatei, GSD-Datei) beschrieben. Die dort hinterlegten Informationen dienen der vereinfachten Konfiguration und Einbindung von PROFIBUS-Geräten unterschiedlicher Hersteller in ein PROFIBUS-Netz. Sie ermöglichen eine Gerätedatenprüfung schon bei der Projektierung eines PROFIBUS-Systems und helfen somit Fehler bei der Projektierung zu vermeiden. Beispielsweise können eingegebene Parametrierungsdaten auf die Einhaltung von Grenzwerten und Zulässigkeit hinsichtlich der Leistungsfähigkeit des Geräts untersucht werden. Durch das festgelegte Dateiformat der GSD-Datei können darüber hinaus herstellerneutrale Projektierungswerkzeuge für PROFIBUS-Systeme realisiert werden. GSD-Dateien werden vom Hersteller individuell für jeden Gerätetyp erstellt und dem Anwender zur Verfügung gestellt. Die Unterscheidung der GSD-Dateien erfolgt über eine explizite Angabe der Hersteller- und Gerätebezeichnung.

Die GSD-Datei selbst liegt im ASCII-Format vor. Bei PROFIBUS-DP gliedert sich der Aufbau in drei Abschnitte [59]. In den allgemeinen Festlegungen stehen Angaben wie Hersteller- und Gerätename, Angabe des Versionsstandes, Signalbelegungen, Überwachungszeiten und unterstützte Baudraten. In einem zweiten Abschnitt werden die einen PROFIBUS-Master betreffenden Informationen hinterlegt wie die maximal mögliche Anzahl an Slaves und die Upload- und Download-Möglichkeiten. Der dritte Abschnitt enthält Slave-spezifische Angaben, u.a. die Anzahl und Art der E/A Kanäle, die Festlegung von Diagnosetexten sowie, bei modular aufgebauten Geräten, die Angabe der verfügbaren Module.

Aufgrund der Protokollunterschiede zwischen PROFIBUS-DP und FMS ist auch der Aufbau der GSD-Datei für FMS dementsprechend unterschiedlich. Insbesondere enthält letztere Angaben über die Kommunikationsbeziehungsliste, das virtuelle Feldgerät und das Objektverzeichnis. Einzelheiten können der GSD-Spezifikation für PROFIBUS-FMS [65] entnommen werden.

Electronic Device Description

Während die GSD-Dateien lediglich die Inbetriebnahme vereinfachen, bietet die *Electronic Device Description* (EDD, [66]) eine Methodik zur Beschreibung der Feldgeräteeigenschaften, die eine einheitliche und plattformunabhängige Gerätebedienung in der Leitebene erlauben sollen. Bereits 1993 wurde im *InterOperable Systems Project* (ISP, [67]) eine Gerätebeschreibung auf Basis der HART-Technologie (Highway Addressable Remote Transducer, [68]) entwickelt, deren Weiterentwicklung nun die EDD für PROFIBUS-Geräte darstellt.

Die EDD gliedert sich in drei Teile (vgl. Bild 21). Den Kern bildet die Definition von Variablen und Funktionen, die zusammen den nach außen sichtbaren Zustand und das Verhalten des Gerätes repräsentieren. Dieser Teil ist insoweit technologieunabhängig, als geräteseitig kein Implementierungsaufwand anfällt. Zur Abbildung der von einem Feldgerät zur Verfügung gestellten Variablen stehen dabei folgende Variablentypen

zur Verfügung: Fließkomma, Integer, String, Index (als Verweis auf ein Array-Element, s.u.), Bit-String sowie Datum- und Zeit-Typen. Die physikalische Adressierung der Variablen erfolgt PROFIBUS-DP spezifisch über Slot/Index und über symbolische Namen (Labels). Als Strukturelemente sind Collections und Arrays nutzbar. Funktionen werden innerhalb der "COMMAND" Struktur definiert, wobei nur das Lesen und Schreiben von Variablen bzw. Datenblöcken unterstützt wird. Die entsprechenden Operationen können mittels Transaktionen geklammert werden. Um Inkonsistenzen beim Lesen oder Schreiben von Variablenwerten zu verhindern, können frei definierbare Aktionen vorher und nachher ausgeführt werden. Die Aktionen werden in der EDD über eingebetteten "C"-Code formuliert.

Im Rahmen der Visualisierung kann mittels der Elemente "Menü" und "Tabelle" die Bedienung des Feldgeräts beschrieben werden. Die Umsetzung der Visualisierung erfolgt beispielsweise im Leitsystem, indem die in der EDD festgelegte Struktur der graphischen Elemente geeignet auf das jeweilige Betriebssystem abgebildet wird. Im Kommunikationsteil wird beschrieben, wie auf Variablen des Feldgeräts, beispielsweise vom Leitsystem aus, zugegriffen werden kann. Dieser Teil ist der komplexeste, da hier tatsächlich eine Methodik entworfen wurde, die den Anspruch hat, unabhängig von einem bestimmten Feldbussystem zu sein. Die konkrete Implementierung ist dann natürlich feldbusspezifisch.

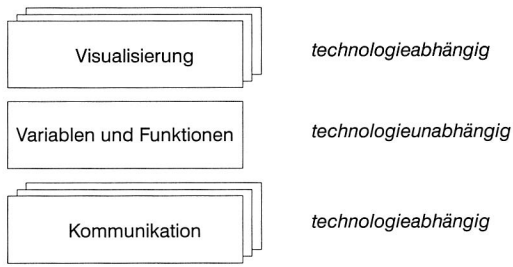


Bild 21: *Schematische Darstellung des Umfangs der Electronic Device Description (nach [66])*

Das nachfolgende Beispiel einer EDD-Datei soll einen Eindruck vermitteln, wie die Definition von Variablen und Funktionen erfolgt. Beschrieben wird eine Temperaturvariable "temperatur", die vom Typ "LOCAL" ist. Damit dient sie lediglich der Repräsentation einer Feldgeräte-Variablen und wird nicht auf dem Feldbus ausgetauscht. Nach jeder Änderung der Variablen wird die Funktion "postscale_temperature" aufgerufen. Diese enthält eingebetteten "C"-Code (hier nicht dargestellt), der unter Umständen eine Skalierung des neuen Wertes beinhalten könnte. Der aktuelle Temperaturwert wird über einen Lesezugriff, der in der angegebenen "COMMAND"-Struktur spezifiziert ist, vom Feldgerät angefordert. Die Leseanfrage und die Antwort werden in einer Transaktion geklammert. An Hand der Definition der Temperaturvariable als Integer-

wert, weiß das Gerät beim lesenden Zugriff, wieviele Bytes ab der durch Slot/Index festgelegten Adresse, übertragen werden müssen.

```
/* Beispiel EDD Datei */

MANUFACTURER 11,
DEVICE_TYPE 11,
DEVICE_REVISION 1,
DD_REVISION 1

VARIABLE temperatur {
   LABEL "Temperaturwert";
   CLASS LOCAL;
   TYPE INTEGER  {
      DEFAULT_VALUE 20;
      MIN_VALUE 15;
      MAX_VALUE 25;
   }
   POST_EDIT_ACTIONS {
      postscale_temperature
   }
   HANDLING READ & WRITE;
}

METHOD postscale_temperature {
   LABEL "Local Method";
   DEFINITION {
      /* C-Code */
   }
}

COMMAND read_3_156 {
   SLOT 3;
   INDEX 156;
   OPERATION READ;
   TRANSACTION {
      REQUEST {
      }
      REPLY {
         temperature;
      }
   }
}
```

Das "Field Device Tool" Konzept

Während feldgeräteseitig die Beschreibung des Funktionsumfangs und der Kommunikationsfähigkeiten mit Hilfe einer eigenen Beschreibungssprache durchaus seinen Zweck erfüllt, stellt die Integration der Gerätefunktionen insbesondere zum Bedienen und Beobachten in der Leitebene ein Problem der Anwendungsintegration dar. Das im Rahmen des ZVEI-Arbeitskreises "Field Device Tool" (FDT, [69]) und des PROFIBUS Arbeitskreises "Gerätebeschreibung" verfolgte Ziel ist es, über die Definition von Softwareschnittstellen die Entkopplung von einer feldbusspezifischen Gerätebeschreibungssprache zu erreichen. Dabei werden die Funktionen des Feldgeräts in ei-

nem vom Hersteller zur Verfügung gestellten Objekt, dem *Device Type Manager* (DTM), eingebettet. Ein DTM stellt eine Software-Komponente dar und verfügt über fest definierte Schnittstellen. Die Realisierung erfolgt als ActiveX-Komponente mit entsprechenden COM-Schnittstellen (vgl. Bild 22). Die im DTM realisierbaren Funktionen umfassen:

- Konfigurationsdialoge für das Gerät
- Plausibilitätsprüfung der Parameter
- automatische Generierung abgeleiteter Parameter
- Vorgabe der Bearbeitungssequenzen von komplexen Kalibrier-, Abgleich- und Einstellvorgängen bei Feldgeräten mit entsprechendem Einsatzgebiet
- lesen/schreiben von Parametern
- spezifische Diagnosefunktionen
- Bereitstellung der Typ-Daten für den Kommunikationsaufbau
- spezifische Dokumentation

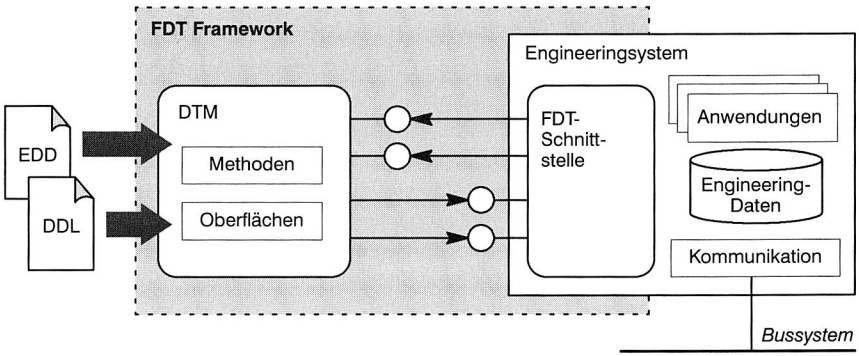


Bild 22: *Das FDT Framework legt die Schnittstellen zwischen dem DTM und dem Engineering-System fest (nach [62])*

Die DTMs müssen nicht in jeden Fall neu entwickelt werden. So soll[4] es auch möglich sein, aus den Gerätebeschreibungen, beispielsweise aus der EDD, eine einfache DTM ("Default DTM") automatisch zu generieren.

Bewertung

Nachfolgend wird eine Bewertung der EDD vorgenommen, da diese das eigentliche Modellkonzept beinhaltet, während die FDT eher der Implementierungsseite zuzuordnen ist.

4. Bisher existieren noch keine entsprechenden Softwarewerkzeuge.

Anforderung M1: Umfassendes, generisches Modellkonzept 

Die EDD selbst bietet kein generisches Modellkonzept, sondern ist für die oben genannten Anwendungszwecke und in erster Linie für PROFIBUS-DP entwickelt worden. Bei der Abbildung der Variablen werden keine strukturierten Datentypen und kein Objektmodell unterstützt. Da bei PROFIBUS-DP die Nutzung der von einem Feldgerät angebotenen Anwendungsdienste indirekt über das Lesen/Schreiben von Variablen und/oder Datenblöcken erfolgt, bietet die EDD auch nur diese Funktionalität in der "COMMAND"-Struktur an.

Anforderung M2: Erweiterbarkeit 

Da es sich bei der EDD nicht um eine Modellierungssprache, sondern um ein fertiges Modellkonzept handelt, müssen neue Modellierungsaspekte durch Erweiterungen der Spezifikation integriert werden. Hier erweist es sich als Nachteil, daß kein echtes Geräte- und Anwendungsmodell existiert. Neue Modellierungsaspekte, die über bloße Erweiterungen bestehender syntaktischer Konstrukte hinausgehen, lassen sich daher nicht systematisch in die Spezifikation einarbeiten.

Anforderung M3: Leichte Integrierbarkeit in bestehende Anwendungen 

Die EDD dient als Ausgangspunkt für einen Compiler, der aus den abstrakt beschriebenen Visualisierungskomponenten eine Bedienoberfläche erzeugen soll und aus den Zugriffsfunktionen die auf dem Feldbus auszuführenden Lese- und Schreiboperationen. Werden – wie im Fall der FDT – lediglich Softwarekomponenten erzeugt und keine fertigen Anwendungsprogramme, so kann die Anforderung nach leichter technischer Integrierbarkeit erfüllt werden. Auf semantischer Ebene bewirkt das zu einfache Modellkonzept, daß ein Client detailliert Vorkenntnisse über das Feldgerät besitzen muß, um die in der EDD definierten Dienste einsetzen zu können.

Anforderung M4: Selbstbeschreibung 

Das mit der EDD definierte Modell ist nicht selbstbeschreibend.

3.4.3 OLE for Process Control

Überblick über OPC

OPC steht für "OLE for Process Control" und bezeichnet einen mittlerweile etablierten Industriestandard für den Austausch von Daten zwischen Anwendungen der Automatisierungstechnik [70–73]. OPC baut dazu auf der (D)COM-Technologie auf (vgl. Kapitel 2.4). Das Ziel der OPC-Spezifikation ist es, eine einheitliche Softwareschnittstelle zu definieren, die für den Nutzer einfach zu handhaben sowie für den Anbieter von Automatisierungssystemen einfach zu implementieren ist. Um eine breite Akzeptanz für OPC sicherzustellen, wurde 1996 die OPC Foundation in den USA gegründet, zu deren Mitgliedern heute über 150 Unternehmen wie Siemens, ABB, Fisher-Rose-

mount, Rockwell Automation oder National Instruments zählen. OPC bietet über die Vorteile einer einheitlichen Softwareschnittstelle hinaus weitere Vorzüge, die seine rasche Verbreitung und Etablierung begründen. Wie nachfolgend genauer beschrieben, sind dies im wesentlichen

- die Auflösung von Hersteller- und Geräteabhängigkeiten,
- der Multi-Client Zugriff sowie
- die Netzwerkfähigkeit.

Nach dem bisherigen Stand der Technik wird zum Datenaustausch zwischen der Prozeßhardware und einer (PC-basierten) Anwendung ein spezieller Treiber benötigt. Die Implementierung des Treibers und somit seine Schnittstelle hängen weitgehend vom Betriebssystem ab. OPC bietet in diesem Punkt eine Standardschnittstelle für Server-Anwendungen an, unabhängig von der Hardware und der Treiberausführung. Ein solcher *OPC-Server* kann allgemein jede Automatisierungskomponente sein, die Daten an potentielle Clients bereitstellen möchte. In vielen Anwendungen wird heute der "Zugangspunkt" zum Feldbussystem, d.h. das Gateway zur überlagerten Zellenebene, in Form eines OPC-Servers realisiert. Für die Hersteller von Client-Softwareprodukten, z.B. Bedien- und Beobachtungssysteme, Prozeßvisualisierungssysteme oder Meß- und Steuerungssysteme, bedeutet dies, daß sie lediglich über eine OPC Client-Schnittstelle verfügen müssen. Durch die Standardisierung der Server-Schnittstelle kann der Hardware- oder der Treiberhersteller seine Implementierung ändern, ohne daß Anpassungen in den existierenden Client-Anwendungen notwendig werden. Der Endkunde erhält damit wiederum die Möglichkeit, zwischen verschiedenen Anbietern von Hard- und Softwarekomponenten frei zu wählen. So kann er sich ganz auf die Funktionalität der Komponenten konzentrieren, anstatt auf ihre Kompatibilität achten zu müssen.

Im Gegensatz zu herkömmlicher Treibersoftware, ist ein OPC-Server standardmäßig in der Lage, Anfragen von mehreren Klienten zu bearbeiten (vgl. Bild 23). Verschiedene Softwarepakete, ob Visualisierungsapplikation oder Archivierungsanwendung, können gleichzeitig Daten vom OPC-Server erhalten. Dabei bedarf es keiner herstellerspezifischen Vereinbarung oder eines zusätzlichen Implementierungsaufwands, da der OPC-Server die von der COM-Technologie zur Verfügung gestellten Mechanismen zur Multi-Client-Fähigkeit automatisch nutzt.

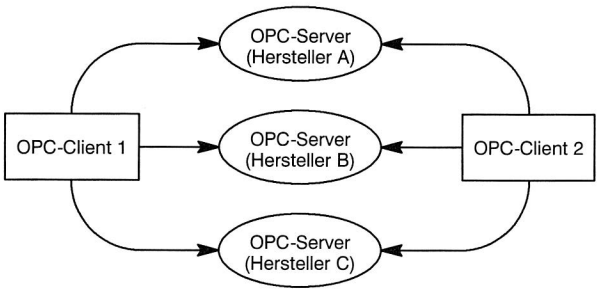


Bild 23: *Die COM-Technologie ermöglicht den gleichzeitigen Zugriff mehrerer Clients auf den OPC-Server (nach [70])*

Im Rahmen von DCOM ist es möglich, daß ein OPC-Server und die jeweiligen OPC-Clients auch über ein Netzwerk kommunizieren können. Die DCOM-Laufzeitbibliothek verbirgt die Details des Zugriffs (Ortstransparenz).

Die Architektur von OPC

Die OPC-Architektur ist für die Anwendung in der Zellen- und Leitebene konzipiert. Dabei stellen die OPC-Server Prozeßdaten durch die Schnittstellen von COM-Objekten zur Verfügung. Jedoch können die OPC-Schnittstellen an unterschiedlichsten Stellen innerhalb einer komplexen Automatisierungsaufgabe eingesetzt werden. Wie in Bild 24 dargestellt, können sie auf der untersten Ebene zum Datentransfer von physikalischen Geräten zum Leitstand oder von solchen Systemen zu den dispositiven Anwendungen verwendet werden.

Die OPC-Spezifikation definiert zwei Ausprägungen von Schnittstellen, die von einem OPC-Client angesprochen werden können: COM *Custom*-Schnittstellen für funktionspointerfähige Programmiersprachen wie C++ und *Automation*-Schnittstellen für Skriptsprachen.

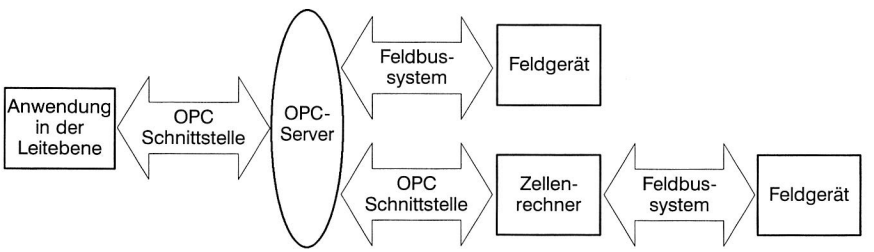


Bild 24: *Einsatzmöglichkeiten eines OPC-Servers – direkte Verbindung zum Feldgerät oder einem Zellenrechner nachgelagert (nach [70])*

Das Objektmodell

Ein OPC-Server implementiert eine dreistufige Objekthierarchie, wie sie in Bild 25 zu sehen ist. Auf der höchsten Ebene dieser Hierarchie steht das *OPCServer* Objekt, das die einzelnen Datenquellen innerhalb des Servers abbildet. Die *OPCGroup* Objekte dienen zum Gruppieren von *OPCItems* sowie zum Datenaustausch zwischen OPC-Server und OPC-Clients. Dazu organisiert der Client die Items zunächst in eine oder mehrere OPC-Gruppen, anschließend können die Daten von den Items gelesen bzw. in die Items geschrieben werden. Die Items korrespondieren direkt mit den auszutauschenden Prozeßwerten, während durch die Gruppen lediglich eine beliebige logische Strukturierung der Items vorgenommen wird.

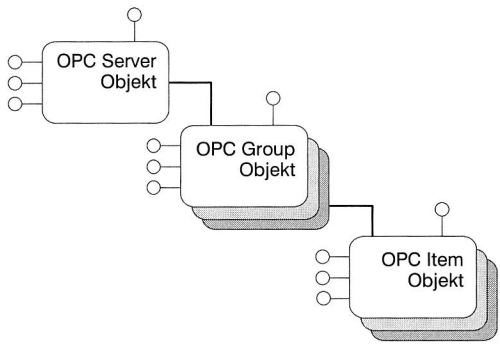


Bild 25: *Die dreistufige OPC Objekthierarchie (nach [70])*

Bewertung

Die Bewertung von OPC erfolgt hinsichtlich der anwendungsspezifischen Eigenschaften auf Anwendungsebene, losgelöst von der tatsächlichen Implementierung. Einige Nachteile von OPC umfassen jedoch auch die Implementierungsebene und werden deshalb vorab angesprochen. Dies ist zum einen die derzeitige Verfügbarkeit, die sich auf Windows-basierte Plattformen beschränkt. Darüber hinaus stellt OPC einen Industriestandard dar, der zwar auf breite Akzeptanz der Hersteller und Anwender setzen kann, jedoch noch keinen internationalen Standard verkörpert.

Ein Grundgedanke bei der Entwicklung von OPC war es, den Zugriff auf die verschiedenen Feldbussysteme durch eine einheitliche Zugriffsschnittstelle zu erleichtern. Obwohl dieser Ansatz einen wichtigen Beitrag in Bezug auf offene und herstellerunabhängige Schnittstellen für das Lesen und Schreiben von Prozeßdaten leistet, verbleiben zwei wesentliche Nachteile. Durch die Vereinheitlichung wird letztlich nur ein Grundgerüst an Kommunikationsdiensten unterstützt, das in etwa der Schnittmenge der Dienste der jeweiligen Feldbussysteme und Automatisierungsgeräte entspricht.

Konkret bedeutet das eine Beschränkung auf das synchrone bzw. asynchrone Lesen und Schreiben von Variablen sowie auf einen Ereignisdienst. Andererseits bietet OPC keinen direkten Zugriff auf die physikalischen Geräte, wenn es als Gateway zu einem darunterliegenden Feldbussystem eingesetzt wird. Dementsprechend können keine Anwendungsfunktionen des Geräts selbst aufgerufen werden. Üblicherweise wird dies umgangen, indem proprietäre Anwendungsprotokolle definiert werden, die – in größere Datenobjekte verpackt – auf Sende- bzw. Empfangsseite manuell kodiert bzw. dekodiert werden müssen. Die aufgeführten Nachteile von OPC sind bekannt, und es wird bereits versucht, Lösungen u.a. im Rahmen der *Complex Data Working Group* zu erarbeiten.

Anforderung M1: Umfassendes, generisches Modellkonzept 

OPC unterstützt hinsichtlich der Modellierung nur eine dreistufige Objekthierarchie. Die Festlegung auf eine bestimmte Tiefe dürfte für die meisten Anwendungsfälle keine entscheidende Einschränkung darstellen. Ein schwerwiegenderer Nachteil ist allerdings in den beschränkten Diensten von OPC zu sehen, die sich unmittelbar in der Anwendungsmodellierung niederschlagen. OPC unterstützt beispielsweise keine komplexen und keine hierarchischen Datentypen. Damit sind Automatisierungsaufgaben, die eine der unten genannten Punkte umfassen, in OPC derzeit nur indirekt über das Lesen und Schreiben von einfachen Variablen abbildbar:

- Der Austausch hierarchischer und/oder strukturierter Daten mit einem Automatisierungsgerät.
- Die Abwicklung eines eigenen komplexeren Protokolls zwischen dem OPC-Client und dem eigentlichen Automatisierungsgerät.
- Das Senden von Kommandos, die Überwachung der Ausführung und das Warten auf eine entsprechenden Rückmeldung.

Die Anforderung zur Unterstützung eines umfassenden Modellkonzepts ist daher nicht vollständig erfüllt.

Da der OPC-Standard anwendungsneutral ist, kann er im Rahmen der vorgegebenen Objekthierarchie und Dienste für beliebige Anwendungen eingesetzt werden. Somit erfüllt er die Forderung nach einem generischen Modellkonzept.

Anforderung M2: Erweiterbarkeit 

OPC definiert keine Modellierungssprache. Insofern müssen neue Modellierungsaspekte über eine Erweiterung des Standards aufgenommen werden. Im Gegensatz zu EDD, auf die diese Aussage ebenfalls zutrifft, besitzt OPC jedoch ein Objektmodell, so daß der Aufwand hierfür relativ gering ist.

Anforderung M3: Leichte Integrierbarkeit in bestehende Anwendungen 

Da OPC auf COM basiert, ist die technische Integrierbarkeit in Anwendungen, die auf der Grundlage von COM erstellt sind, sehr gut. Anders sieht es mit der semantischen

Integration der OPC-Objekte in eine Anwendung aus. Prinzipiell erstellt nämlich der Client die OPC-Gruppen und organisiert anschließend die OPC-Items entsprechend. Dazu muß der Client bereits wissen, welche logischen Gruppen er anlegen möchte und wie die Zuordnung der Prozeßwerte auszusehen hat. Dies entspricht aber einem detaillierten Wissen um den tatsächlichen Automatisierungsprozeß bzw. dessen Strukturierung in die drei Dezentralisierungsaspekte Hardware, Funktion und Software (vgl. Kapitel 2.1.3).

Beispielsweise kann es in einer Anwendung sinnvoll sein, die Prozeßwerte entsprechend dem Feldgerät, dem diese Werte zugeordnet sind, zu organisieren, in einer verteilten Anwendung kann eine logische Gruppierung besser geeignet sein. Eine "zentrale Instanz", die für die Erzeugung der Gruppen-Struktur im OPC-Server verantwortlich ist und diese eventuell automatisch aus dem darunterliegenden Feldbussystem und den dort angeschlossenen Feldgeräten erzeugt, wäre hier wünschenswert.

Anforderung M4: Selbstbeschreibend 

OPC legt generell nicht fest, wie *Daten*, die von einem Automatisierungsgerät stammen, im Client zu interpretieren sind. Daraus resultiert das dargelegte Problem, daß zusätzlich herstellerspezifische Protokolle über die ausgetauschten Prozeßwerte abgewickelt werden müssen, um Metadaten transportieren zu können.

Ein OPC-Server wird in Form eines COM-Servers implementiert, das OPC-Objektmodell in Form von COM-Objekten. Damit können Clients zur Laufzeit die COM-eigenen Methoden zur Abfrage der verfügbaren *Schnittstellen* der Objekte nutzen. Somit bietet OPC zwar keine Selbstbeschreibung der Daten an, jedoch – aufgrund der Implementierung – eine Selbstbeschreibung der Schnittstellen der OPC-Objekte. Dies stellt jedoch keine besondere Eigenschaft von OPC dar, sondern ist allein der Basistechnologie zuzurechnen.

3.5 Zusammenfassung

Die aktuellen Technologien in den Bereichen der Feldbuskommunikation und der Geräte- bzw. Anwendungsmodellierung wurden nicht mit der Zielvorstellung der einfachen Integration in die übergeordneten Ebenen entwickelt. Dementsprechend sind sie auch nicht in der Lage, die gestellten Anforderungen vollständig zu erfüllen. An Hand der vorgenommenen Bewertung konnten jedoch die wesentlichen Schwachstellen identifiziert werden.

Bei den Feldbussystemen fehlen "Integrationsdienste", die ein Feldgerät nutzen kann, um beispielsweise eine virtuelle Kommunikationsverbindung zu einer Anwendung in der Leitebene oder zu einem Feldgerät, das sich an einem anderen Feldbussystem befindet, aufbauen zu können. Umgekehrt könnten diese auch von Anwendungen außerhalb der Prozeßebene zum direkten Zugriff auf ein Feldgerät genutzt werden. Idea-

lerweise müßten die Dienste, die konzeptionell der Middleware zuzuordnen sind, Bestandteil des Feldbusprotokolls selbst sein, um eine effiziente Implementierung gewährleisten zu können.

Im Bereich der Geräte- und Anwendungsmodellierung ist ein wesentlicher Schwachpunkt in den für jedes Feldbussystem unterschiedlichen Konzepten zu sehen. Diese sind, wie das Beispiel EDD und OPC zeigt, noch dazu nicht in der Lage, die anwendungsnahe Modellierung komplexerer Abläufe zu unterstützen.

4 Einsatz universeller Geräteprofile in der Leitebene

Universell einsetzbare Feldbussysteme können einen großen Bereich von Anwendungen in der Automatisierungstechnik umfassen – von der Prozeßebene bis hin zur Zellen- und Leitebene. Diese Vielfalt der möglichen Anwendungen setzt einen entsprechend weiten Funktionsumfang der Protokolle voraus. In einer konkreten Anwendung wird davon jedoch nur eine Untermenge benötigt. Die Auswahl einer solchen Untermenge in Verbindung mit weiteren Festlegungen zur Anwendung ist Aufgabe eines Profils. Speziell in einem Geräteprofil werden Aussagen über die im Produktivbetrieb ausgetauschten Informationen und über den Ablauf komplexerer Geräteaktionen gemacht. Es bestimmt damit indirekt die Bedien- und Beobachtungsschnittstelle zu einem Feldgerät. Das nachfolgend vorgestellte Konzept beschreibt in diesem Zusammenhang eine Architektur, die auf der Basis von Geräteprofilen funktional äquivalente Softwarekomponenten anbietet, die in der Zellen- bzw. Leitebene eingesetzt werden können und dabei eine direkte Kommunikationsverbindung zu dem jeweiligen Feldgerät unterhalten. Durch den Einsatz von Softwarekomponenten bei der Implementierung wird die Integration in bestehende Anwendungen wesentlich vereinfacht.

4.1 Problemstellung und Zielsetzung

Die Steuerung und Überwachung des Produktionsprozesses ist eine Hauptaufgabe der Zellen- bzw. Leitebene. Ziel es ist, dem Bediener einer Anlage, trotz der räumlichen Trennung vom Prozeß, ein ausreichendes Maß an Informationen über den Prozeßzustand zur Verfügung zu stellen, so daß dieser, wenn nötig, regelnd in den Prozeß eingreifen kann. Diese Funktion wird auch als Bedienen und Beobachten bezeichnet und durch spezielle HMI-Software (Human Machine Interface) realisiert. Die Gerätehersteller liefern dazu den Kunden bis heute spezielle, auf die Gerätefamilie abgestimmte Programme, um so die Funktionalität des jeweiligen Geräts optimal ansprechen zu können. Der langfristige Trend geht jedoch zu herstellerneutralen HMI-Systemen.

Allgemein muß die HMI-Software mindestens das folgende Aufgabenspektrum abdecken können [12]:

- Visualisierung des Prozeßstatus durch die Abbildung der Prozeßwerte auf graphische Elemente
- Visualisierung des Gerätestatus aller Feldgeräte
- Aktivierung von speziellen Gerätefunktionen eines Feldgeräts
- Anlauf bzw. Herunterfahren eines Feldgeräts bzw. des betreffenden Prozesses
- Unterstützung der gleichzeitigen Koordination mehrerer Feldgeräte bzw. Prozesse

Die HMI-Software bildet somit den zur Erfüllung der Bedien- und Beobachtungsaufgabe notwendigen Teil an Informationen und Funktionalitäten in der Zellen- bzw. Leitebene ab. Indem bisher für jede Automatisierungsaufgabe die Abbildung neu vorgenommen werden muß und in diesem Zusammenhang auch keine standardisierten Bedienkonzepte existieren, ergeben sich für den Anwender gravierende Nachteile:

- die Bedienphilosophie und die Bedienoberfläche differieren bei gleichen Automatisierungsaufgaben
- der Einarbeitungsaufwand für die Benutzung der HMI-Software steigt
- damit steigt die Wahrscheinlichkeit von Bedienungsfehlern [74]

Außerdem muß die HMI-Software eine Kommunikationsverbindung zum Feldbussystem unterhalten, um auf die Geräte- bzw. Prozeßdaten zugreifen zu können. Hierfür werden noch häufig feldbusspezifische Treiber eingesetzt, mit dem Ziel, die Funktionalität des Bussystems optimal ausschöpfen zu können. Dies schränkt jedoch in der Prozeßebene die Austauschbarkeit der Hardware ein und verhindert die einfache Wiederverwendbarkeit der HMI-Software.

Das im folgenden vorgestellte Konzept setzt dazu auf den einheitlichen und feldbusneutralen Zugriff auf die Daten der Prozeßebene unter der Verwendung von universellen Softwarekomponenten für immer wieder vorkommende Automatisierungsaufgaben. Diese müssen lediglich für den konkreten Anwendungsfall und für das spezielle Feldgerät geeignet parametriert werden. Damit eröffnet sich die Möglichkeit des schnellen und kostengünstigen Erstellens von Anwendungen in der Zellen- bzw. Leitebene durch das bloße Konfigurieren fertiger Softwarebausteine im Sinne eines "Rapid Prototyping". Zudem wird durch den Einsatz von Softwarekomponenten bei der Implementierung der Aufwand für die Integration in bestehende Anwendungen wesentlich vereinfacht.

4.2 Vorstellung der Architektur

Von zentraler Bedeutung ist die Konzeption der Softwarekomponenten. Diese müssen in der Lage sein, häufig wiederkehrende Automatisierungsfunktionen sowie Bedien- und Beobachtungsaufgaben abzubilden. Sowohl für die Identifikation als auch für die Implementierung dieser Aufgaben erweist sich ein Blick auf die existierenden Feldbusprofile als sinnvoll.

Die im Rahmen dieser Arbeit betrachteten universell einsetzbaren Feldbussysteme können einen großen Bereich von Anwendungen in der Automatisierungstechnik abdecken. Dies setzt allerdings einen entsprechend weiten Funktionsumfang der Protokolle voraus, von dem in einer konkreten Anwendung jedoch nur eine Untermenge tatsächlich benötigt wird. Ein Profil wählt eine solche Untermenge aus und trifft zudem weitere Festlegungen zur Anwendung. Eine der charakteristischen Eigenschaften ei-

nes Profils ist, daß es den Informationsaustausch zwischen unterschiedlichen Geräten verschiedener Hersteller ermöglicht.

Entsprechend dem Gehalt der Informationen und den festgelegten Gerätereaktionen unterscheidet man drei Arten von Profilen [75]:

- Das *Kommunikationsprofil* bildet die Grundlage für den Informationsaustausch zwischen den Geräten. Auf dieser Stufe werden kommunikationsspezifische Festlegungen getroffen, aber noch keine Aussagen über den Gehalt der Informationen und über die Gerätereaktionen gemacht.
- Im *Geräteprofil* werden auf der Grundlage des Kommunikationsprofils für eine fest umrissene Gerätegruppe Aussagen über den Gehalt der Informationen, die benutzten Datenformate und über die Gerätereaktionen getroffen. Der Umfang der Festlegungen ist durch den Anwendungsbereich der Geräteprofile gegeben.
- Mit dem *Branchenprofil* werden für einen Anwendungsbereich unter Einbeziehung von Kommunikations- und Geräteprofilen geräteübergreifende Gesamtabläufe festgeschrieben.

Beispiele für Geräteprofile sind die von der PROFIBUS Nutzerorganisation definierten Profile für drehzahlveränderliche Antriebe, für Sensoren und Aktoren, für NC/RC-Steuerungen oder für HMI-Systeme.

Aus Sicht der komponentenorientierten Softwareentwicklung können solche Kommunikationsprofile in der Prozeßebene als eine universelle Schnittstelle für eine breite Gruppe von Automatisierungsgeräten und -aufgaben angesehen werden. In der Leitebene stellen sie, aufgrund der genauen Spezifikation der Datenformate und der Semantik der ausgetauschten Informationen, eine implizite Beschreibung der notwendigen Funktionen der Benutzerschnittstelle dar. Die Vereinigung dieser beiden Aspekte ist Gegenstand des Konzepts. Dabei dienen die Geräteprofile als Ausgangspunkt für die Ableitung von Softwarekomponenten, die in der Leitebene in bestehende Anwendungen integriert werden können und dort zwei Kernaufgaben übernehmen:

- Implementierung der "Logik", die in einem Geräteprofil hinterlegt ist. Je nach Profil kann dies beispielsweise die Überprüfung auf korrekt gesetzte Parameterwerte bedeuten, die Überwachung, ob bestimmte Abhängigkeiten zwischen den Parametern eingehalten werden, oder die Abwicklung komplexerer Funktionen im Rahmen des Geräteengineerings oder des Produktivbetriebs.
- Integration der Geräteprofile in bestehende Bedien- und Beobachtungssysteme, indem die mit einem Geräteprofil verbundenen Parameter, Variablen und Dienste graphisch angeboten werden, wobei die Realisierung durch Softwarekomponenten erfolgt.

Bild 26 gibt einen konzeptionellen Überblick über die Architektur. Die mit einem Feldgerät verbundene Automatisierungsaufgabe wird dabei soweit wie möglich auf beste-

hende Geräteprofile abgebildet. In der Leitebene erfolgt die Auswahl der zugehörigen Softwarekomponenten. Diese werden im Zuge eines Konfigurationsprozesses dynamisch an das jeweilige Feldgerät gebunden und können damit gerätespezifisch in bestehende Anwendungen integriert werden. Die Abwicklung der Kommunikation mit der Prozeßebene erfolgt unter Verwendung des OPC-Standards. Damit können prinzipiell alle Feldbussysteme, die über eine OPC-Schnittstelle verfügen, gleichermaßen angesprochen werden, ohne daß Änderungen in den Softwarekomponenten notwendig sind.

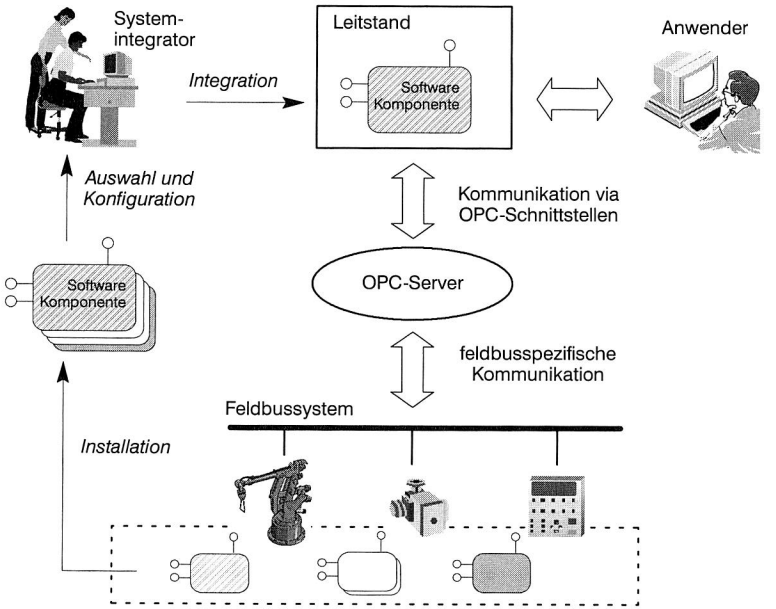


Bild 26: *Einsatz universeller Geräteprofile – konzeptioneller Überblick über die Architektur*

4.3 Abgrenzung zu aktuellen Technologien

Richtlinie VDI/VDE 2187

Im Rahmen der seit 1996 vorliegenden Richtlinie VDI/VDE 2187 "Einheitliche Anzeige- und Bedienoberfläche auf Personalcomputern für digitale Feldgeräte" wird für die Konfiguration, Parametrierung und Instandhaltung von Feldgeräten eine einheitliche Anzeige-Bedienoberfläche standardisiert. Bei der Umsetzung dieser Richtlinie verhalten sich die Feldgerätehersteller und auch die Feldbusorganisationen sehr zögerlich (vgl. [76]).

Fraglich ist in diesem Zusammenhang, ob in einem technisch orientierten Umfeld eine Top-down Standardisierung, d.h. aus der Sicht des Anwenders kommend, ausreichende Akzeptanz findet. Zudem stellt die konkrete Ausprägung einer Benutzerschnittstelle, deren Bedienerfreundlichkeit und Konfigurierbarkeit ein entscheidendes Alleinstellungsmerkmal des Geräteherstellers bei sonst identischem Funktionsumfang dar.

Das vorliegende Konzept geht daher von einem Bottom-up Ansatz aus, indem es konkrete Automatisierungsaufgaben an Hand der Geräteprofile in entsprechende Benutzerschnittstellen der Leitebene umsetzt. Die Funktionalität, d.h. die "Logik", orientiert sich deshalb an den technischen Aufgaben. Zur Erstellung der Benutzerschnittstelle können, wie im nächsten Kapitel beschrieben, vorgefertigte und konfigurierbare Software-Komponenten eingesetzt werden, jedoch kann der Hersteller auch seine eigenen Oberflächen einbinden.

Das "Field Device Tool"-Framework

Im FDT-Framework wird die Funktionalität eines Feldgeräts im Device Type Manager eingebettet. Der DTM kann dabei beliebig komplex sein und Unterstützung für alle Einsatzphasen eines Feldgeräts bieten: von der Konfiguration und Parametrierung über das Bedienen und Beobachten bis hin zur Dokumentation und Diagnose. Dabei ist der (Mindest-) Umfang des DTM selbst nicht standardisiert, so daß vor dessen Einsatz, beispielsweise im Rahmen einer komplexen Leitstandanwendung, die angebotenen Funktionen genau geprüft werden müssen. Da der DTM herstellerabhängig ist, können Versions- oder Geräteänderungen schnell problematisch werden. Wiederum gilt, daß der Funktionsumfang der DTM nicht an konkreten Automatisierungsaufgaben ausgerichtet ist. Die Bedienerschnittstellen sind im DTM zudem fest hinterlegt.

4.4 Konzeption der Softwarekomponenten

Bei der Konzeption der Softwarekomponenten erweist sich eine strikte Trennung der beiden Kernfunktionen, welche die Logik bzw. die graphisch-interaktiven Schnittstellen realisieren, als sinnvoll. Prinzipiell wäre es zwar möglich, daß eine einzige Komponente beide Aufgaben implementiert. Erfahrungsgemäß ist es aber gerade in Hinblick auf die verschiedenen Ansprüche an eine Bedienerschnittstelle besser, dem Benutzer oder Anwendungsentwickler eine Reihe von relativ kleinen und leicht konfigurierbaren Komponenten zur Verfügung zu stellen, die miteinander beliebig kombiniert werden können.

Somit besteht jede Softwarekomponente ihrerseits wiederum aus zwei Teilkomponenten. Die erste Teilkomponente, das *Steuerelement* (vgl. Bild 27), ist für die Abwicklung der Logik des Geräteprofils zuständig. Sie implementiert gewissermaßen das Protokoll zwischen der Bedienerschnittstelle und dem Feldgerät und muß dazu eine Kommunikationsverbindung mit dem OPC-Server aufbauen und unterhalten.

Für die Oberflächengestaltung der Bedienerschnittstelle kann aus verschiedenen Teilkomponenten gewählt werden. Dabei muß es möglich sein, eine Laufzeitverbindung zum Steuerelement herzustellen, so daß die Aktualisierung der Bedienerschnittstelle automatisch, d.h. ohne zusätzlichen Programmieraufwand, erfolgen kann. Das Steuerelement muß somit die Methoden, die an der Benutzerschnittstelle angeboten werden, implementieren und dabei auch die Konfiguration gewisser Eigenschaften der Kommunikationsverbindung zum OPC-Server ermöglichen. Zudem existiert eine Ereignisschnittstelle, über die beispielsweise Verbindungsprobleme oder Fehlermeldungen an die Bedienerschnittstelle oder an zusätzliche externe Anwendungsmodule weitergeleitet werden können. Eine detaillierte Beschreibung der Schnittstellen des Steuerelements wird im nächsten Kapitel im Rahmen eines konkreten Anwendungsbeispiels gegeben.

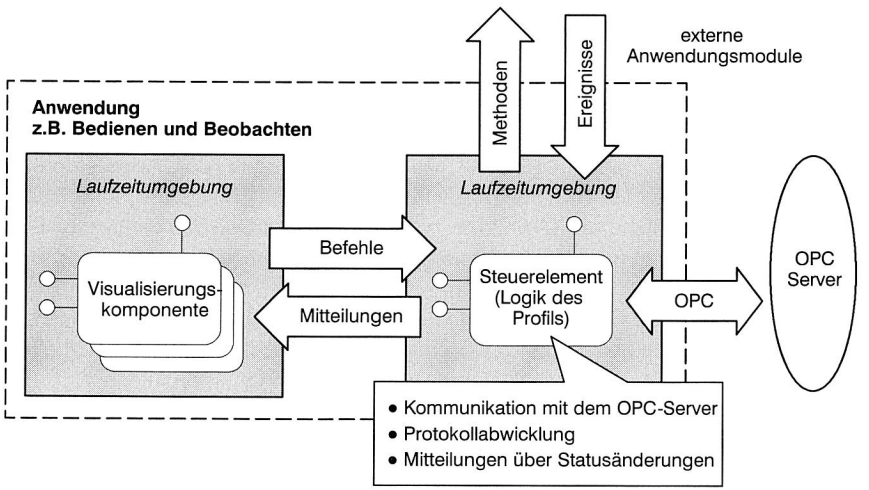


Bild 27: *Konzeption der Softwarekomponenten*

4.5 Umsetzung des Konzepts am Beispiel des NC/RC-Geräteprofils des PROFIBUS

4.5.1 Vorstellung des NC/RC-Profils

Das Profil für Robotersteuerungen und Numerische Steuerungen wurde vom Arbeitskreis NC/RC im Fachausschuß Profilbildung der PROFIBUS Nutzerorganisation erarbeitet. Ziel dieses Profils ist es, auf der Basis von PROFIBUS-DP das Systemverhalten für die aufgeführten Steuerungstypen aus Sicht der PROFIBUS-Kommunikation zu be-

schreiben. Im industriellen Einsatz sind oftmals CNC-, NC- und RC-Steuerungen gleichzeitig anzutreffen, so daß eine identische Schnittstelle für diese unterschiedlichen Steuerungstypen erstrebenswert ist. Folgende Vorteile ergeben sich aus einer einheitlichen Schnittstelle [77]:

- Systemressourcen in der überlagerten Steuerung (z.B. SPS) werden eingespart, da nur eine Schnittstelle zu verwalten ist
- Erleichterungen bei der Inbetriebnahme und der Wartung der Anlage
- geringerer Einarbeitungsaufwand für das Bedienpersonal
- geringere Fehleranfälligkeit

Das im Profil definierte Geräteverhalten beschränkt sich auf die grundlegenden Gerätefunktionen und geht nicht auf die spezifischen Eigenschaften unterschiedlicher Steuerungstypen ein. Folglich muß das Profil selbst auch so allgemein gehalten sein, daß eine Unterscheidung der verschiedenen Steuerungstypen nicht notwendig ist.

Das Geräteprofil beschreibt lediglich die Funktionalität und das Verhalten der Steuerung in der Automatik-Betriebsart, d.h. der Betriebsart, in der das System ferngesteuert wird (vgl. [77]). In dieser Betriebsart müssen die folgenden Betriebszustände der Maschine der überlagerten Steuerung angezeigt und von dieser manipuliert werden können:

- Hochlauf der Maschine nach dem Einschalten und Start eines Anwendungsprogramms, z.B. eines Roboterbewegungsprogramms
- Anhalten der Maschinenbewegung
- Anhalten der Maschinenbewegung mit gleichzeitigem Stillsetzen der Antriebe (Hardware-Stop)
- Fortsetzen der gestoppten Maschinenbewegung
- Programmende, d.h. Ende des gestarteten Anwendungsprogramms
- Anzeige interner Störungen der Maschine an die überlagerte Steuerung

Diese Betriebszustände beschreiben den Minimalumfang, den eine Maschinensteuerung in der Automatik-Betriebsart erfüllen muß, um den Anforderungen des PROFIBUS NC/RC-Profils zu entsprechen.

Gemäß der europäischen Richtlinie EN 60204-1 zur Sicherheit von Maschinen werden drei Stop-Kategorien unterschieden:

- Kategorie 0: Stillsetzen durch sofortiges Ausschalten der Energiezufuhr zu den Maschinenantrieben
- Kategorie 1: Ein gesteuertes Stillsetzen, wobei die Energiezufuhr zu den Maschinenantrieben beibehalten wird, um das Stillsetzen zu erzielen und die Energiezufuhr erst dann unterbrochen wird, wenn der Stillstand erreicht ist.

- Kategorie 2: Ein gesteuertes Stillsetzen, bei dem die Energiezufuhr zu den Maschinenantrieben erhalten bleibt.

Entsprechend dieser Richtlinie werden die Stop-Betriebszustände bei den betrachteten Maschinen in der Automatik-Betriebsart in folgende Klassen eingeteilt:

- Hardware-Stop: Die Maschinenbewegung wird durch ein Hardware-Stopsignal entweder intern oder durch eine überlagerte Steuerung angehalten. (Stop der Kategorie 0 oder 1).
- Stop extern: Die Maschinenbewegung wird durch die überlagerte Steuerung angehalten. Die Antriebe der Maschine bleiben nach Stillstand der Bewegung eingeschaltet (Stop der Kategorie 2).
- Stop intern: Die Maschinenbewegung ist durch ein internes Ereignis gestoppt worden. Die Antriebe bleiben auch hier nach Stillstand der Bewegung eingeschaltet (Stop der Kategorie 2).

Die Kommunikation zwischen der überlagerten Steuerung und der Maschinensteuerung erfolgt über das Lesen bzw. Schreiben eines Steuer- und eines Statusworts, das aus zwei Bytes besteht (Bild 28).

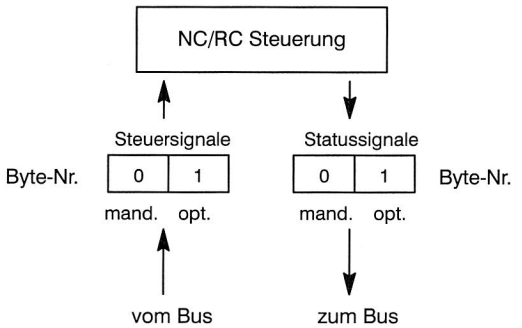


Bild 28: *E/A-Kopplung über Steuer- und Statussignale (nach [77])*

Physikalisch sind die Steuersignale mit den Eingängen und die Statussignale mit den Ausgängen der Maschinensteuerung verbunden. Es sind somit sechszehn Steuer- und Statussignale definiert, wobei zwischen verbindlichen *(mandatory)* und optionalen Signalen unterschieden wird. Die verbindlichen Signale sind jeweils in den ersten acht Bit zu finden, so daß zur Implementierung der minimalen Funktionalität die Abbildung auf ein Byte im Eingangs- und ein Byte im Ausgangsdatenfeld der Maschinensteuerung genügt.

Das Betriebsverhalten der Maschinensteuerung wird im NC/RC-Profil in Form von Timingdiagrammen für die Steuer- und Statussignale dargestellt. In Bild 29 ist beispiel-

haft das Einschalten und Anhalten (Stop der Kategorie 2) eines Anwendungsprogramms dargestellt.

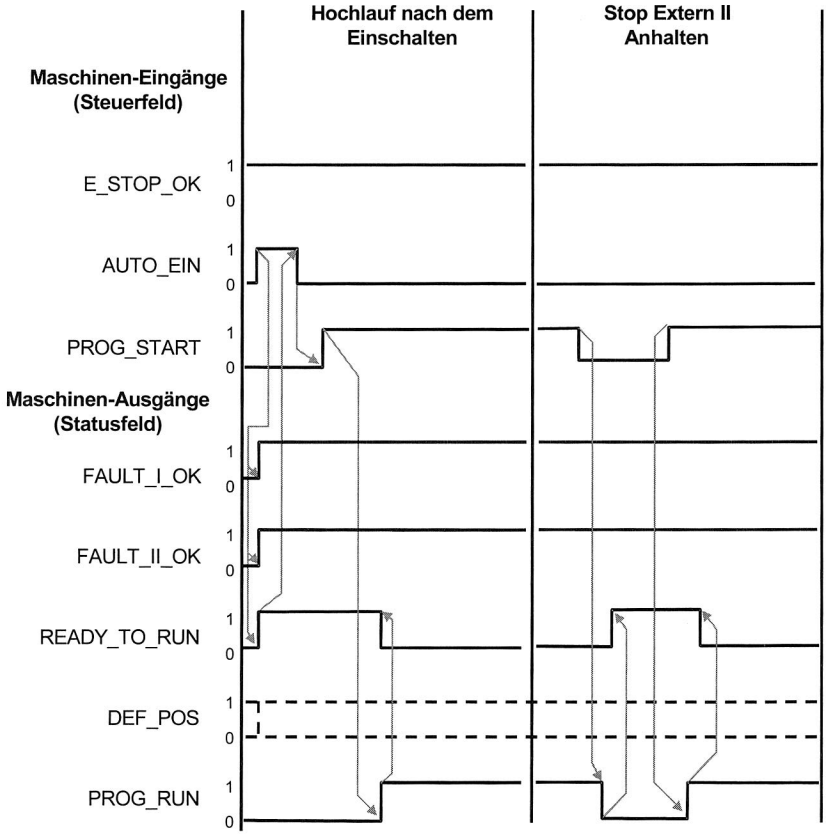


Bild 29: *Zeitdiagramm: Einschalten und Anhalten eines Anwendungsprogrammes*

4.5.2 Konzeption des Steuerelements

Die Implementierung des Steuerelements kann in fünf wesentliche Teile gegliedert werden: die Benutzerschnittstelle (Methoden, Eigenschaften und Ereignisse), der Verbindungsaufbau und die Kommunikation mit dem OPC–Server, die Logik der Protokollabwicklung, die Laufzeit-Konfiguration und die Kommunikation mit den Visualisierungskomponenten. Diese werden im weiteren Verlauf genauer beschrieben.

Die folgenden Methoden werden der Visualisierungskomponente vom Steuerelement angebotenen:

- *Connect* und *Disconnect*, um die Verbindung zu einem OPC-Server auf- bzw. abzubauen
- *StartUp*, *Pause* und *Resume*, um eine Maschinenbewegung einzuleiten, zu unterbrechen (Stop der Kategorie 2) und anschließend wieder fortzusetzen
- *HardwareStop*, um einen Stop der Kategorie 1 durchzuführen
- sowie zwei weitere Methoden, welche die Benutzerschnittstelle zum an- bzw. abmelden am Steuerelement aufrufen muß

Desweiteren sind folgende Eigenschaften des Steuerelements parametrierbar bzw. abrufbar:

- der Name des OPC-Servers
- der Name der Items, die das Status- bzw. das Steuerbyte des Feldgeräts repräsentieren
- der momentane Betriebszustand der Maschine
- die Protokollierungsfunktion des OPC-Servers

Das Steuerelement informiert die Visualisierungskomponenten über folgende Ereignisse:

- Interner Fehler, der zu einem Stop der Kategorie 2 führte bzw. Behebung des internen Fehlers (genauer spezifiziert im NC/RC-Profil)
- Fehlerfreier Abschluß einer Maschinenbewegung
- Programmstart, Programmpause und Hardware-Stop
- sonstige Fehler

Um die Logik des Geräteprofils abzubilden, ist im Steuerelement ein zu den Timingdiagrammen des Profils äquivalenter endlicher Automat implementiert.

4.5.3 Implementierung der Softwarekomponenten

Für die Implementierung der Softwarekomponenten bietet sich die Wahl von ActiveX-Komponenten an. Diese basieren technologisch gesehen auf COM, ebenso wie der OPC-Server, so daß eine nahtlose kommunikationstechnische Integration zwischen dem Steuerelement und dem OPC-Server gegeben ist. Um die Konfiguration des Steuerelements zur Laufzeit für den Anwendungsentwickler einfacher zu gestalten, werden durch die Implementierung der COM-spezifischen *IPropertyPages*-Schnittstelle mehrere sogenannte *Property Pages* angeboten, die eine Konfiguration der Komponenten zur Laufzeit erlauben.

Mit Hilfe der in Bild 30 dargestellten Property Page kann einer der im System vorhandenen OPC-Server als OPC-Datenquelle für das Steuerelement ausgewählt werden.

Die Items des OPC-Servers werden entweder in Form eines Baumes – falls der Server seinen Namensraum als "hierarchisch" deklariert – oder in Form einer Liste dargestellt. Es besteht hierbei die Möglichkeit, die Items nach Datentyp, Zugriffsrechten und/oder Namensmustern zu filtern. Zwei von diesen Items muß der Benutzer als Repräsentanten für das Status- bzw. Steuerwort der zu steuernden Maschine angeben, wodurch das Steuerelement fest an ein bestimmtes Feldgerät gebunden wird.

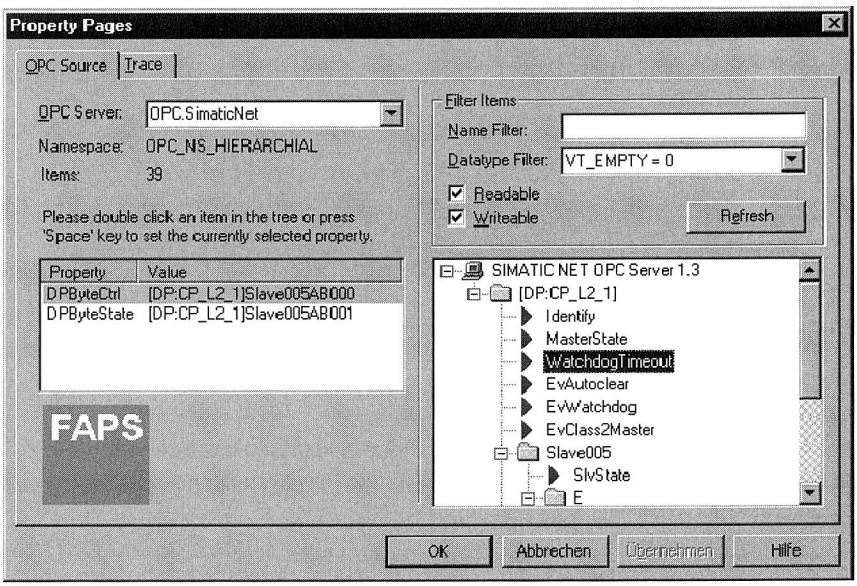


Bild 30: *Die OPC-Source Property Page zur Konfiguration des Steuerelements*

Eine weitere Property Page verwaltet die Eigenschaften, die das Protokollieren in einer Datei steuern. Hier kann der Benutzer den Dateinamen angeben, die Protokollierung ein- oder ausschalten und die zu protokollierenden Ereignisse konfigurieren. Daneben werden drei weitere Property Pages angeboten – zwei davon sind Standard-Property Pages für Farben- bzw. Fontauswahl, die dritte Property Page ermöglicht die Maskierung einzelner Bits.

Auf Basis des Steuerelements und dessen Schnittstellen ist es nun innerhalb weniger Minuten möglich, eine graphische Oberfläche, wie in Bild 31 gezeigt, zu erstellen und einzusetzen.

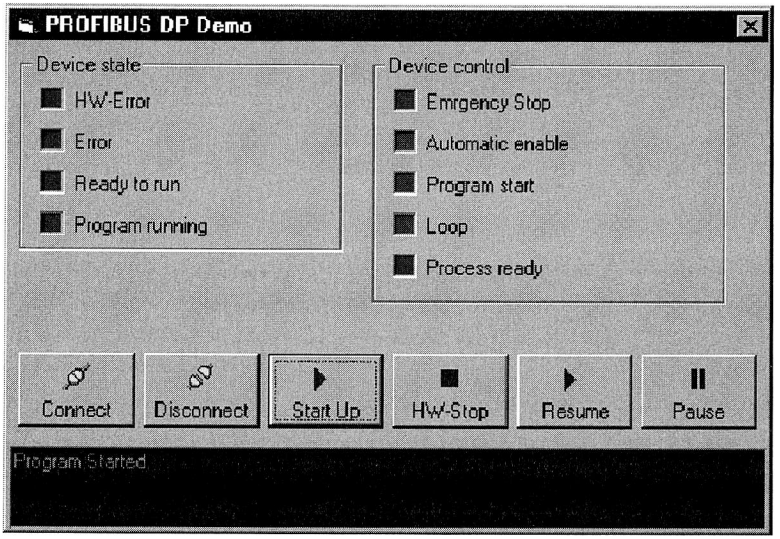


Bild 31: *Realisierung einer einfachen Bedienoberfläche auf Basis des implementierten Steuerelements*

Interessant ist in diesem Zusammenhang, daß ActiveX-Komponenten im Internet-Explorer ausgeführt werden können und damit eine Fernbedienung über das Internet durchführbar ist. Hinsichtlich der Sicherheit dieser Variante, stellt allein die Trace-Option ein potentielles Risiko dar, da diese Art von Ausgaben in der Regel auf eine Datei umgelenkt werden. Die vorliegende Implementierung unterstützt daher explizit die COM-Schnittstelle *IObjectSafty.*

In Bild 32 ist der Einsatz des Steuerelements im Web-Browser zu sehen. In ihrem Funktionsumfang gleicht die gezeigte HTML-Seite im wesentlichen der oben vorgestellten Bedienoberfläche. Der Zustand der Maschinensteuerung wird mit Hilfe von einfachen Visualisierungskomponenten angezeigt, die dabei an ein unsichtbares Steuerelement gebunden sind. Der Aufruf der Methoden des Steuerelements erfolgt mittels eingebetteter VBScript-Makros. Eine solche HTML-Seite, plaziert auf dem Server, ermöglicht das Steuern und Überwachen der jeweiligen Maschine oder des Roboters von jedem Arbeitsplatz aus. Dabei bedarf es auf Seite des Client keines zusätzlichen Konfigurations- und Pflegeaufwands: die Steuerelemente werden auf dem Rechner automatisch installiert oder durch die neueste Version ersetzt. Handelt es sich um mehrere PROFIBUS-DP Slaves, die auf diese Weise angesprochen werden sollen, so wäre es denkbar, die HTML-Seiten auf einem Intranet-Servers durch ein CGI-Skript dynamisch generieren zu lassen.

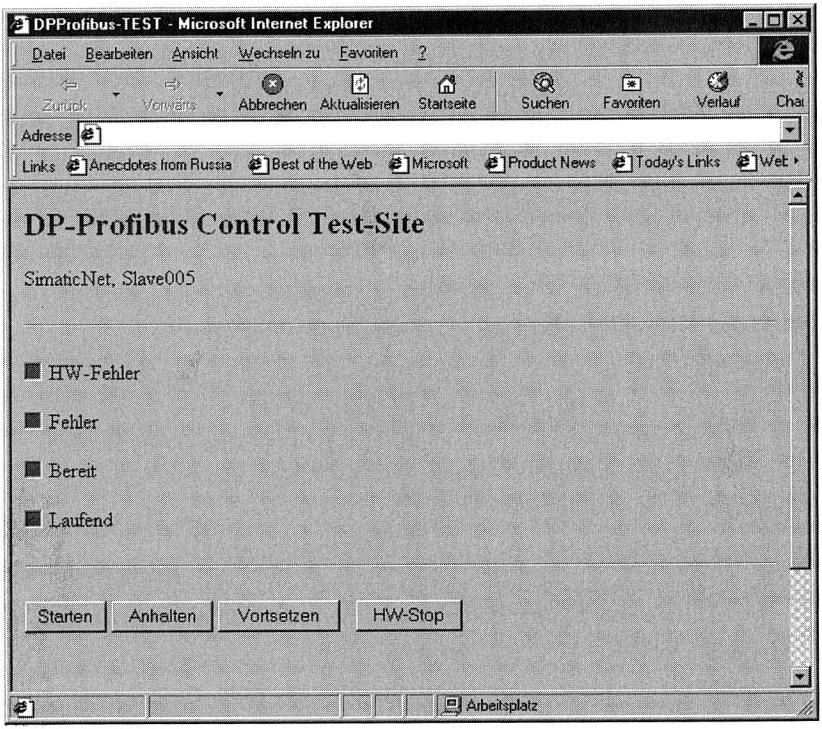


Bild 32: *Realisierung der Bedienung über den Web-Browser*

4.6 Zusammenfassung und Bewertung

Das vorliegende Konzept deckt Aspekte der Implementierungsebene und der Ebene der Anwendungslogik gleichermaßen ab. Die Vorteile liegen in dem schnellen und wenig fehleranfälligen Prozeß der Anwendungserstellung von HMI-Software für häufig vorkommende Automatisierungsaufgaben, da bereits vorgefertigte Softwarekomponenten lediglich konfiguriert werden müssen. Mit der Implementierung als ActiveX-Komponenten wird die Integration in bestehende COM-basierte Anwendungen wesentlich erleichtert. Zudem werden durch die Nutzung von OPC eine Vielzahl von Feldbussystemen unterstützt. Hinsichtlich der Feldgeräte fällt keinerlei Implementierungsaufwand an. Ein Nachteil ist darin zu sehen, daß noch keine feldbusübergreifend einheitlichen Geräteprofile existieren. Desweiteren muß der Client COM-Schnittstellen anbieten – sei es im Betriebssystem oder durch zwingende Verwendung des Internet Explorers als Web-Browser.

Nachfolgend wird eine detaillierte Bewertung hinsichtlich der in Kapitel 2 formulierten Anforderungen vorgenommen:

Anforderung K1: Unterstützung der vertikalen Kommunikation 

Das Steuerelement stellt eine virtuelle Punkt-zu-Punkt Verbindung zum Feldgerät her. Dies wird zur Laufzeit durch eine Bindung an entsprechende OPC-Items, die wiederum Prozeßvariablen des Feldgeräts repräsentieren, erreicht. Über die Items wird das im Geräteprofil definierte Protokoll abgewickelt. Das Feldgerät kann dabei selbständig Meldungen an die so verbundenen Clients absetzten, jedoch ist es ihm nicht möglich, beliebige Clients anzusprechen.

Anforderung K2: Unterstützung der horizontalen Kommunikation 

Die horizontale Kommunikation wird nicht explizit unterstützt.

Anforderung K3: Unterstützung für ein verteiltes Client/Server-Modell 

Im Rahmen der im Profil definierten Funktionen wird ein verteiltes Client/Server-Modell im Steuerelement fest hinterlegt. In seiner Komplexität ist es allerdings, durch die Notwendigkeit der Abbildung auf Items des OPC-Servers, beschränkt.

Anforderung M1: Umfassendes, generisches Modellkonzept 

Das vorgestellte Konzept basiert auf der Umsetzung *vorhandener* Geräteprofile. Der Schritt der Modellierung ist damit bereits vollzogen, weshalb keine Bewertung hinsichtlich dieses Kriteriums vorgenommen werden kann.

Einige weiterführende Überlegungen sollen dennoch angestellt werden:

- Die Forderung nach einem generischen Modellkonzept richtet sich in erster Linie an die Ersteller der Profile. In der Praxis basieren nicht nur gleichnamige Profile unterschiedlicher Feldbussysteme auf verschiedenen Modellkonzepten, sondern auch die verschiedenen Profile ein und des selben Feldbussystems. Eine Vereinheitlichung über die Feldbussysteme hinweg, würde sowohl die Pflege alter als auch die Erstellung neuer Profile wesentlich vereinfachen.
- Etwas differenzierter kann man die Forderung nach einem umfassenden Modellkonzept betrachten. Würden die vorhandenen Geräteprofile alle denkbaren Anwendungsfälle abdecken können, so wäre der Zweck erfüllt und die Forderung hinfällig. Problematisch ist in diesem Zusammenhang, daß Geräteprofile immer nur eine Grundmenge an Funktionalitäten bereitstellen, um von konkreten Automatisierungsgeräten, deren Funktionsvielfalt oft weit größer ist, abstrahieren zu können. Ohne eine weitere Spezialisierung der Profile ist diese Forderung also nicht abzudecken.

Anforderung M2: Erweiterbarkeit 

Da die Modellierung nicht Gegenstand des Konzepts ist, wird keine Bewertung hinsichtlich dieses Kriteriums vorgenommen. Die Erweiterbarkeit wäre in diesem Fall jedoch mit einer neuen, erweiterten Version eines Geräteprofils gleichzusetzen.

Anforderung M3: Leichte Integrierbarkeit in bestehende Anwendungen 

Aufgrund der Implementierung der Software in Form von ActiveX-Komponenten ist die technische Integrierbarkeit in COM-basierte Anwendungen sehr einfach. Über eine spezielle COM/CORBA-Bridge, wie sie in Kapitel 5.3.4 genauer beschrieben wird, können auch COM-basierte Anwendungen wie native CORBA-Anwendungen angesprochen werden. Auf semantischer Ebene ist nachteilig anzumerken, daß prinzipbedingt die Profillogik dem Ersteller einer Anwendung bekannt sein muß, da er die graphische Benutzerschnittstelle zum Ansprechen des Steuerelements konzipieren muß.

Anforderung M4: Selbstbeschreibung 

Da die Modellierung nicht Gegenstand des Konzepts ist, wird keine Bewertung hinsichtlich dieses Kriteriums vorgenommen. Selbstbeschreibung ist in den bisher definierten Geräteprofilen jedoch nicht vorgesehen.

5 Erweiterung der Funktionalität von Feldgeräten durch den Einsatz gerätespezifischer Java-Applets

Was im Umgang mit vielen Geräten des Alltags üblich ist, das sogenannte "Plug and Play", womit das Anschließen und der sofortige produktive Betrieb des Geräts gemeint ist, ist bei der Inbetriebnahme eines Feldbussystems eine seit langem gestellte Forderung der Anwender. Prinzipiell sollte es möglich sein, ein für eine bestimmte Aufgabe vorgesehenes Feldgerät an das Feldbussystem anschließen und unverzüglich betreiben zu können. Um dies zu erreichen, müssen die Geräte selbst mit soviel eigener "Intelligenz" ausgestattet sein, daß keine manuellen Software-Installationen mehr notwendig sind. Dieser Kerngedanke wird in einem Konzept umgesetzt, das gerätespezifische Software-Komponenten im Feldgerät integriert und die notwendigen Kommunikationsdienste zur Übertragung dieser Komponenten an potentiellen Clients bereitstellt. Das "Plug and Play" bei der Inbetriebnahme ist dann nur *ein* entscheidender Vorteil neben weiteren.

5.1 Problemstellung und Zielsetzung

Grundsätzlich können bei der Inbetriebnahme von Feldgeräten zwei Fälle unterschieden werden. Feldgeräte mit stark anwendungsspezifischer Ausprägung werden in der Regel nur einmal für die gesamte Lebensdauer installiert. Demgegenüber werden Feldgeräte oftmals erst bei ihrer Einbindung in den Prozeß funktionsspezifisch angepaßt und müssen während der gesamten Lebensdauer flexibel parametrierbar sein.

Die Inbetriebnahme von Feldgeräten in der Prozeßebene wird in der Zellen- und Leitebene durch den Einsatz verschiedener Softwaresysteme für die Inbetriebnahme und das Geräteengineering unterstützt. Die Programme sind meist herstellerspezifisch, so daß bei einem herstellerübergreifenden Austausch der Feldgeräte die Software ebenfalls ausgetauscht werden muß. Neue Geräteversionen erfordern zudem Software-Updates, die mit viel Aufwand, Risiken und hohen Kosten verbunden sind [64]. Falls identische Geräte in unterschiedlichen Versionen eingesetzt werden, müssen auch die entsprechenden Softwareversionen immer verfügbar sein. Darüber hinaus ist der Anwender, aufgrund der Vielzahl verschiedener Gerätetypen, gezwungen, unterschiedliche Werkzeuge zu beherrschen sowie Daten manuell zwischen einzelnen Werkzeugen auszutauschen. Die Konsistenz einer auf diese Weise erstellten Konfiguration kann dabei letztlich nur über einen intensiven Anlagentest sichergestellt werden [62].

Als Lösung würde sich prinzipiell anbieten, Zellen- bzw. Leitrechner als zentrale Arbeitsplätze für das Engineering, die Diagnose und die Wartung einzusetzen. Jedoch sind sie gegenwärtig nicht in der Lage, den vollen Umfang der Funktionalität der Feldgeräte abzudecken, da hierzu gerätespezifische Werkzeuge notwendig sind. Diese können meist nur direkt am Feldbus oder an einem bestimmten Feldgerät ange-

schlossen und betrieben werden. Beispiele sind Programmier- und Parametriergeräte, die in Form von Handterminals feldbus- und gerätespezifisch vorliegen.

Mit der Verbreitung einheitlich aufgebauter Gerätebeschreibungsdateien, wie sie beispielsweise beim PROFIBUS bereits üblich sind, kann zumindest das Projektierungswerkzeug zur Konfiguration des Bussystems und der angeschlossenen Feldgeräte im Rahmen der Inbetriebnahme herstellerunabhängig ausgelegt werden. Das Problem der verschiedenen Versionsstände der herstellerspezifischen Gerätebeschreibungen bleibt indes erhalten. In Bild 33 ist dazu das prinzipielle Vorgehen bei der Projektierung eines PROFIBUS-Systems dargestellt. Die Gerätebeschreibungsdateien des Herstellers werden in das Projektierungswerkzeug eingelesen. Dort erfolgt die Konfiguration des gesamten Netzes inklusive der angeschlossenen Geräte. Im letzten Schritt wird die Konfiguration auf den (die) PROFIBUS-Master geladen.

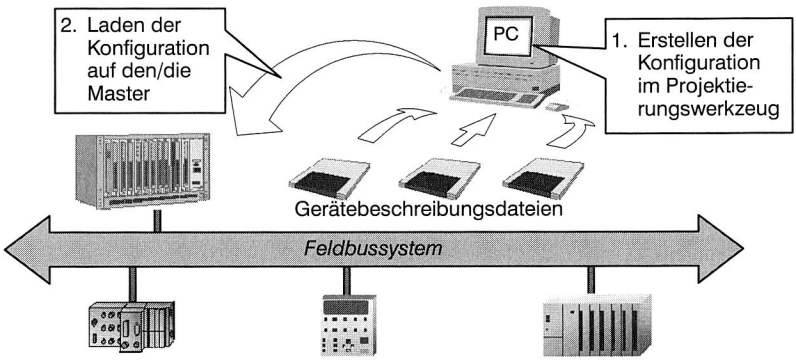


Bild 33: *Projektierung der Feldgeräte mittels Gerätebeschreibungsdateien (nach [8])*

Neben den genannten Problemen bei der Inbetriebnahme der Feldgeräte entsteht während der Betriebsphase weiterer Aufwand für die Erstellung und Pflege der HMI-Software (vgl. Kapitel 4.1). Die Gerätehersteller liefern den Kunden bis heute spezielle, auf die Gerätefamilie abgestimmte Programme, um so die Funktionalität des jeweiligen Geräts optimal unterstützen zu können. Für die HMI-Software in der Zellen- bzw. Leitebene ergeben sich dadurch folgende Nachteile [64]:

- die Bedienung der Feldgeräte von Fremdherstellern ist nicht oder nur mit großem Aufwand möglich
- die Bedienphilosophie und die Bedienoberfläche differieren von einem Feldgerätehersteller zum anderen, so daß sich die Wahrscheinlichkeit von Bedienungsfehlern erhöht [74]
- mit jeder Änderung im Feldgerät geht in der Regel auch ein Software-Update in der HMI-Software einher.

- Software-Updates sind stets mit viel Aufwand, Risiken für das Anlagenverhalten und hohen Kosten verbunden

Die Problematik wird durch immer schnellere Wechsel der Gerätegenerationen verstärkt, da dies eine Beschleunigung des Softwareentwicklungszyklusses zur Folge hat. Gleichzeitig erhöht sich durch neue Anforderungen an die Flexibilität der Anwendungen die Komplexität der HMI-Software, so daß auch für den Hersteller hohe Kosten mit der Erstellung und Pflege seiner Anwendungen verbunden sind.

In Tabelle 2 sind zusammenfassend die verschiedenen Aufgaben eines Zellen- bzw. Leitrechners den entsprechend benötigten Gerätefunktionen bzw. Informationen gegenübergestellt. In allen Bereichen entsteht dabei erheblicher Aufwand, der v.a. in der Vielzahl der zu verwendenden und schlecht integrierten Werkzeuge begründet ist.

Aufgaben / Funktionalität	Inbetrieb-nahme	Engineering	Bedienen, Beobachten	Diagnose, Wartung
Prozeß-E/A	X		X	
Prozeßstatus	X		X	X
Gerätestatus	X		X	X
Gerätekonfiguration	X	X		
Geräteparametrierung	X	X		X
Funktionsaktivierung	X		X	X
Anlauf/Abfahren	X		X	
Kooperative Funktionen	X		X	

Tab. 2: *Bereitstellung von Gerätefunktionen und Informationen zur Erfüllung der verschiedenen Aufgaben eines Zellen- bzw. Leitrechners (nach [12])*

Die nachfolgend vorgestellte Architektur bietet dem Anwender Unterstützung bei allen oben aufgeführten Aufgaben: von der Inbetriebnahme über die Steuerung und Überwachung bis zur Diagnose und Wartung. Dabei wird ein feldbusneutraler Zugriff auf die Feldgeräte realisiert.

5.2 Vorstellung der Architektur

Konzeptionell können die meisten der oben aufgeführten Nachteile dadurch gelöst werden, daß, wie in Bild 34 gezeigt, die Programme, die für die Installation, Wartung, Parametrierung aber auch Bedienung und Beobachtung der Feldgeräte notwendig sind, auf dem Feldgerät selbst gespeichert sind. Damit ist gewährleistet, daß der Funktionsumfang des Feldgeräts ohne die Probleme der Pflege verschiedener Softwarestände in der Zellen- bzw. Leitebene jederzeit verfügbar ist. Im Gegensatz zur Verwendung von Gerätebeschreibungsdateien, die jeweils noch gerätespezifisch an-

gepaßt werden müssen, kann der Hersteller bereits "seine" spezielle Konfiguration im EPROM hinterlegen. Damit verringert sich bereits der Engineering-Aufwand beträchtlich.

Vorteile:

- Gerätebeschreibungen "on board"
- keine Probleme mit der Versionsverwaltung
- keine weiteren gerätespezifischen Anpassungen
- Einbindung in existierende Systeme zur Unterstützung der Aufgaben des Zellen- bzw. Leitrechners

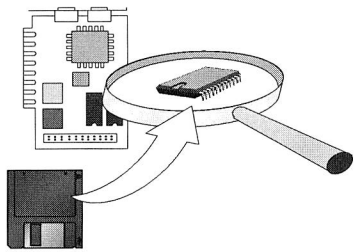


Bild 34: *Speicherung gerätespezifischer Eigenschaften im EPROM des Feldgeräts*

In Bild 35 ist die aus diesem Konzept resultierende Architektur dargestellt. Beim Anschluß eines Feldgerätes an den Feldbus, bei dem gerätespezifische Programme im EPROM hinterlegt sind, werden diese im ersten Schritt an einem speziellen Zellenrechner registriert *(2)*. Dabei erfolgt eine physikalische Übertragung der Programme auf den Zellenrechner. Dieser fungiert als Gateway zwischen dem Feldbussystem und dem in der Leitebene eingesetzten TCP/IP-basierten Netzwerk und stellt somit den Zugangspunkt zu den am Feldbus angeschlossenen Feldgeräten dar. Clients, die die in den Feldgeräten hinterlegte Funktionalität nutzen wollen, verbinden sich mit dem Zellenrechner und können dort die Liste der registrierten Programme durchsuchen und die gewünschten laden *(3)*. Im Client werden diese dann ausgeführt *(4)*, wobei eine *virtuelle Kommunikationsverbindung* zu dem Feldgerät aufgebaut wird, von dem sie ursprünglich stammen. Darüber ist die Ausführung der eigentlichen Funktionen direkt auf dem Feldgerät möglich, beispielsweise zur Parametrierung, zur Diagnose oder zum Abrufen von Prozeßwerten *(5)*.

Da die Programme im EPROM des Feldgeräts gespeichert werden und der dort verfügbare Speicherplatz sehr beschränkt ist, müssen die hinterlegten Programme sehr klein und kompakt sein. Hier ist der Einsatz der Programmiersprache Java ideal. Ein weiterer Vorteil liegt in der Plattformunabhängigkeit von Java, so daß für die Clients alle Rechnerplattformen eingesetzt werden können, die die Virtuelle Maschine von Java implementieren. Werden anstelle von Java-Programmen Java-Applets im Feldgerät hinterlegt, so ist die Ausführung innerhalb eines normalen Web-Browsers möglich. Für weitere Einzelheiten des Konzepts sei auf [78,79,80] verwiesen.

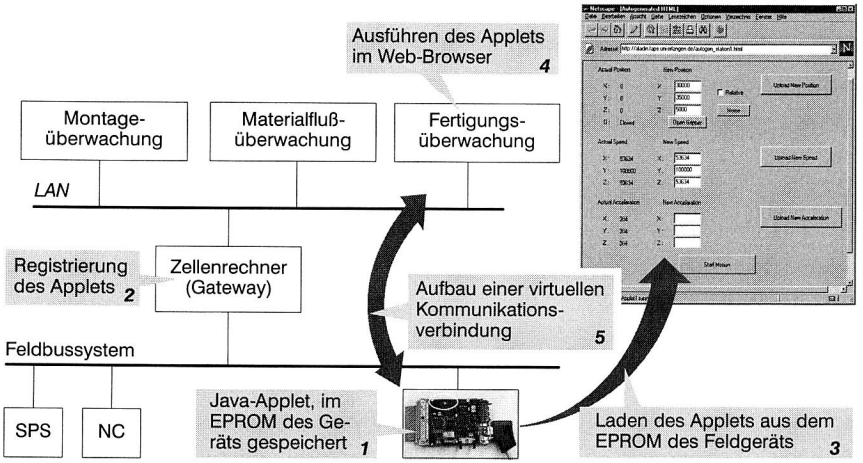


Bild 35: *Architektur zur Speicherung und Nutzung der in den Feldgeräten in Form von Java-Applets hinterlegten gerätespezifischen Eigenschaften*

Typische Aufgabenbereiche für derartige Java-Applets sind beispielsweise die

- Installation, Parametrierung und Konfiguration
- Alarmbehandlung
- Visualisierung von Prozeßwerten
- Diagnose
- Fernwartung und Fernbedienung

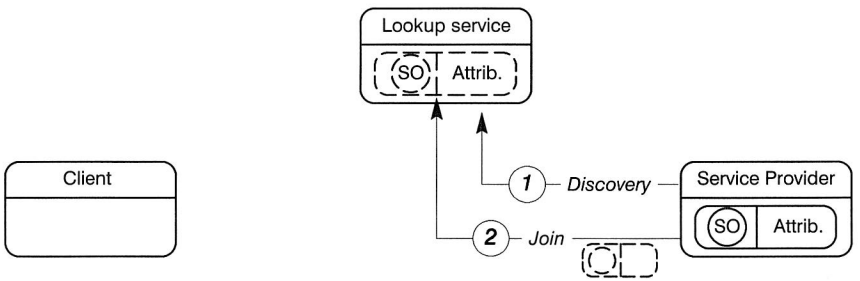
Die in Bild 35 beschriebene Architektur wurde am Lehrstuhl FAPS implementiert und zur Steuerung von NC-Achsen über im Web-Browser ablauffähige Java-Applets eingesetzt. Mit der "Java Intelligent Network Infrastructure" entwarf Sun Microsystems kurze Zeit darauf eine ähnliche, auf Java basierende Architektur. Das ursprüngliche Konzept wurde daher, um die Kompatibilität zu der neuen Technologie zu gewährleisten, entsprechend weiterentwickelt. Die Darstellung dieser Ergebnisse stellt den Schwerpunkt der folgenden Kapitel dar. Für ein besseres Verständnis ist zunächst ein Überblick über die von Sun entworfene Architektur notwendig.

5.3 Konzeption eines Jini-basierten Frameworks zum Zugriff auf Feldgeräte

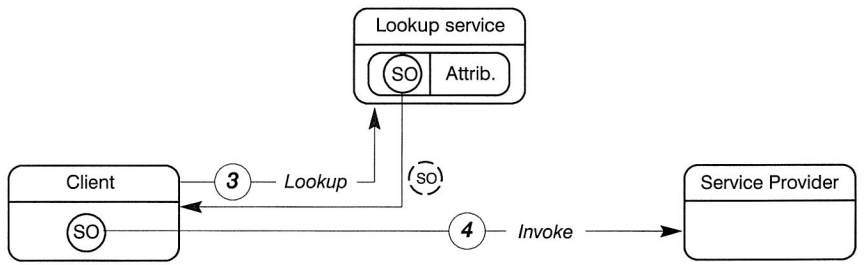
5.3.1 Die Java Intelligent Network Infrastructure

Das Ziel der *Java Intelligent Network Infrastructure* (Jini, [81,82]) ist es, Netzwerkressourcen für potentielle Clients leichter zugänglich zu machen. Die Architektur von Jini

wird durch entsprechende Spezifikationen beschrieben. Im wesentlichen besteht Jini aus zwei Protokollen, dem *Discovery Protocol* und dem *Lookup Protocol* [83–86]. Sie werden von einem Server dazu eingesetzt, um seine verfügbaren Dienste an einer zentralen Instanz zu registrieren, und von einem Client, um Dienste zu finden und zu nutzen. Wenn sich ein Server an das Netzwerk anbindet, führt er einen sog. *Discovery* Vorgang durch, d.h. er versucht via Multi- oder Broadcast einen *Lookup Service* zu lokalisieren. Mittels des Discovery-Protokolls kann er seine spezifischen Dienste beim Lookup-Dienst registrieren *(join)*. Das bedeutet, daß ein *Service Object* physikalisch beim Lookup-Dienst hinterlegt wird (vgl. Bild 36).



(a) Discovery/Join-Service



SO: Service Object (Proxy)

(b) Lookup-Service

Bild 36: *Die Jini-Architektur im Überblick (nach [82])*

Clients nutzen ihrerseits das Lookup-Protokoll, um sich zum Lookup-Dienst zu verbinden und einen spezifischen Dienst anzufordern. Dabei können Clients einen Dienst an Hand seines Namens oder seiner Attribute auffinden. Clients bekommen eine Kopie des im Lookup-Dienstes hinterlegten Service-Objekts. Bei Ausführung des Service-Objekts im Client baut dieses eine Kommunikationsverbindung zum Server auf und stellt eventuell eine graphische Benutzeroberfläche für die angebotenen Dienste zur

Verfügung. Um seitens der Client-Plattformen betriebssystemunabhängig zu sein, werden bei Jini die Service-Objekte in der Programmiersprache Java erstellt.

Zusätzlich zu den genannten Mechanismen, unterstützt Jini *Leasing*, um Dienste exklusiv aber zeitbegrenzt nutzen zu können, ein auf *Principals* und *Access Control Lists* basierendes Sicherheitskonzept, flache *Transaktionen* und *Events.* Weitere Informationen hierzu sind in den jeweiligen Spezifikationen [87–92] zu finden.

Aus der Jini-Architektur ergeben sich drei wesentliche Vorteile. Das Service-Objekt wird immer direkt vom Server zur Verfügung gestellt, so daß keine Versionskonflikte auftreten können. Clients interagieren über wohldefinierte Schnittstellen mit dem Lookup-Dienst, die Implementierung der Kommunikation bleibt dem Service-Objekt überlassen. Damit kann das Service-Objekt beliebig komplexe Kommunikationsdienste unter Ausnutzung Server-spezifischer Details implementieren. Zudem können Clients einen Dienst nutzen ohne explizit Software installieren zu müssen, da die Aufbauphase der Kommunikation auch transparent abgewickelt werden kann. Das Jini-Framework stellt damit die ideale Plattform für die Umsetzung des in Kapitel 5.2 dargestellten Konzepts dar.

5.3.2 Adaption der Jini-Architektur

Obwohl die Jini-Architektur ursprünglich nicht für den Einsatz in Fertigungssystemen gedacht war, kann das zugrundeliegende Konzept entsprechend angepaßt werden, um den Zugang zu Feldgeräten über Feldbussysteme zu ermöglichen. Jedoch müssen einige Besonderheiten in Betracht gezogen werden. Da die in Feldgeräten verfügbaren Ressourcen typischerweise knapp bemessen sind, müssen die Service-Objekte klein sein. Insbesondere müssen Änderungen am Feldgerät selbst auf ein Minimum reduziert werden, um bestehende Anwendungen leicht modifizieren und damit migrieren zu können. Dies impliziert auch, daß das zugrundeliegende Feldbusprotokoll so wenig wie möglich modifiziert werden darf.

Aufgrund der Ressourcenknappheit bietet sich – wie bei Jini vorgesehen – der Einsatz von Java für die Implementierung der Service-Objekte an. Java bietet nicht nur die geforderte Plattformunabhängigkeit auf der Client-Seite, sondern auch die Möglichkeit besonders kleine Anwendungen zu schreiben. Darüber hinaus erweist es sich als ratsam den Lookup-Dienst außerhalb der Prozeßebene zu implementieren. Dann muß lediglich das Discovery-Protokoll auf das jeweilige Feldbusprotokoll abgebildet werden, wohingegen das Lookup-Protokoll über TCP/IP implementiert werden kann. Das Discovery/Join-Protokoll und die Kommunikation zwischen Client und Server kann bei der Verwendung eines OPC-Servers unabhängig vom darunterliegenden Feldbussystem konzipiert werden. Damit ist der Lookup-Service und der Client mit dem Feldgerät über den OPC-Server verbunden (Bild 37).

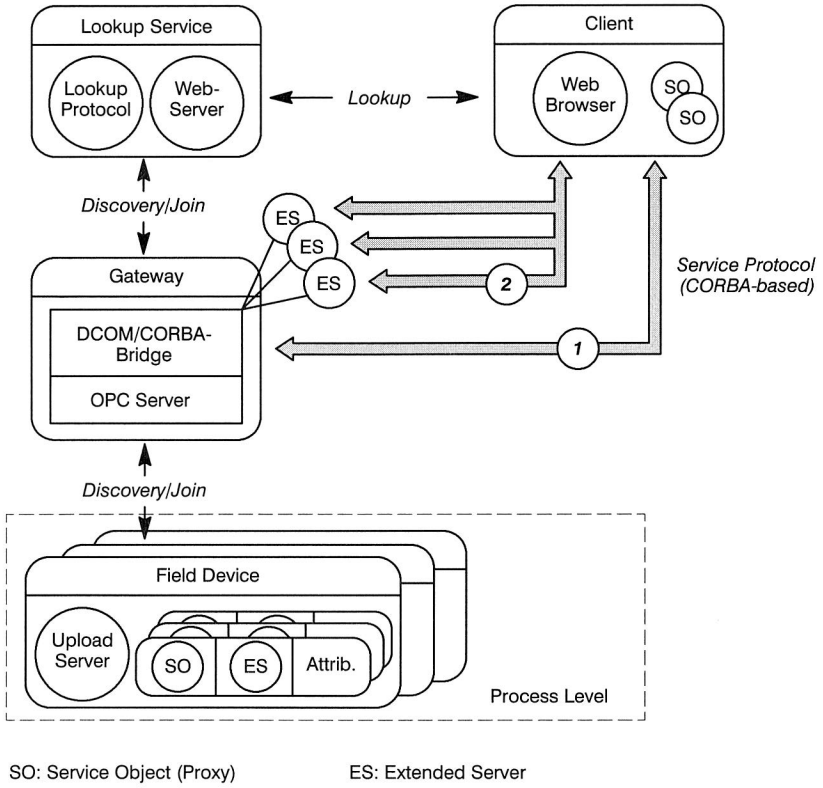


Bild 37: *Ergänzung der Architektur um eine Servererweiterung*

Wie in Kapitel 3.4.3 dargestellt, sind mit der Verwendung eines OPC-Servers eine Reihe von Nachteilen verbunden. Insbesondere die geringe Funktionalität der angebotenen Dienste erschwert zum einen die Implementierung des Discovery/Join-Protokolls und beeinflußt, weit schwerwiegender, die Kommunikation zwischen dem Service-Objekt des Clients mit dem Feldgerät. Deshalb wurde der Ansatz dahingehend erweitert, daß ein sogenannter *Extended Server* (ES) durch den Server bereitgestellt wird. Der ES wird automatisch an dem Rechner installiert, der den OPC-Server beinhaltet. Das Service-Objekt des Clients interagiert mit dem ES, der wiederum transparent die Kommunikation mit dem Feldgerät abwickelt. Damit kann der ES dem Client komplexere Dienste anbieten, die er eigenverantwortlich auf die Dienste des OPC-Servers abbildet (Bild 37). Die weiteren Details der Architektur, z.B. die Verwendung der *DCOM/CORBA-Bridge* und der Einsatz eines Web-Servers, wird in den nachfolgenden Kapiteln erläutert.

5.3.3 Das Discovery/Join-Protokoll

Gemäß der Jini-Spezifikation wird der Discovery/Join-Prozeß vom Server angestoßen. Typischerweise würde der Server zunächst einen Broadcast aussenden, um einen Lookup-Dienst zu lokalisieren. Diese Konzeption würde zu einigen Änderungen im Feldgerät selbst sowie im Feldbusprotokoll führen. Deshalb wurde ein anderes Vorgehen gewählt. Dabei wird der Lookup-Dienst dynamisch zu den potentiellen Servern über eine entsprechende Bindung an den OPC-Server gebunden. Der Discovery/Join-Prozeß wird dann vom Lookup-Dienst selbst angestoßen.

Die Frage ist nun, wie der Lookup-Dienst von potentiellen Service-Objekten in den Feldgeräten Kenntnis erhält. Zu diesem Zweck wird ein OPC Item, das einem vordefinierten Namensschema folgt, verwendet. Das Namenspräfix *JA_xxx* steht für ein Service-Objekt, das detaillierter beschrieben wird durch seine spezifischen Attribute: einem Namen, einem beschreibenden Text, einem Index, der die Position im Speicher des Feldgeräts benennt und seine Größe in Kbyte (Bild 38).

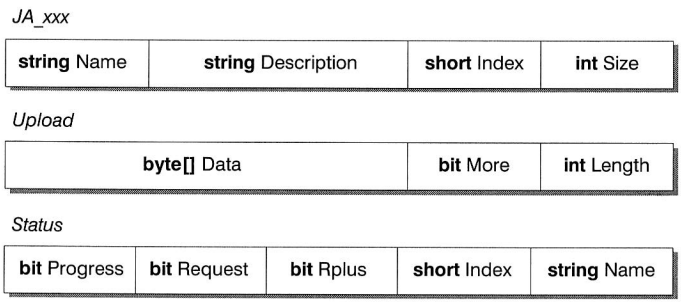
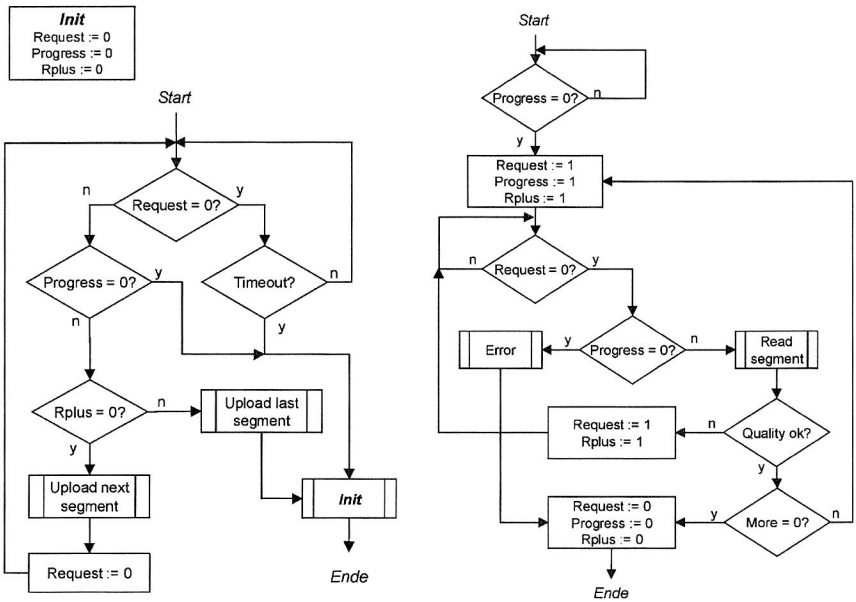


Bild 38: *Konzeption des Discovery/Join-Protokolls*

Nachdem der Lookup-Dienst die Service-Objekte über ihren Namen identifiziert hat, stößt er einen Upload an. Nachdem nur einige Feldbussysteme tatsächlich einen Upload-Dienst unterstützen, wird dazu ein einfacher, auf dem PROFIBUS-FMS [26] Standard basierender Upload-Dienst spezifiziert. Dieser erfordert zwei Feldbus-Variablen beliebiger Länge, *Upload* und *Status*. Diese werden benutzt für das segmentweise Laden des Service-Objekts bzw. um im laufenden Betrieb den gegenwärtigen Status des Uploads anzuzeigen. Die Variable *Upload* besteht aus einem Byte-Array, das das aktuelle Segment enthält. Ein Bit *(More)* zeigt an, ob noch weitere Segmente nachfolgen. Ein Längenfeld gibt die Größe der effektiv in der Variable *Upload* übertragenden Daten an. Die Variable *Status* dient der Synchronisation des Ladevorgangs. Ein *Request* Bit wird gesetzt, um den Upload eines Segments, das mit *Name* und *Index* bezeichnet werden muß, anzustoßen. Name und Index sind dabei identisch zu den in der Variable *JA_xxx* gespeicherten Werten. Falls gerade ein Ladevorgang statt-

findet, so ist das *Progress* Bit gesetzt. Es kann immer nur ein Upload stattfinden. Nachdem der Lookup-Service ein Datensegment erfolgreich erhalten hat, fordert er über das Setzten des *Request* Bit und des *Rplus* Bit das nächste Segment an. Falls das *Rplus* Bit nicht gesetzt wird, so hat dies eine Wiederholung des zuletzt übertragenen Segments zur Folge. Bild 39 zeigt die vollständigen Flußdiagramme für den Upload sowohl für den Server, d.h. das Feldgerät, als auch für den Client, d.h. den Lookup-Service.



(a) Feldgerät *(b) Lookup-Service*

Bild 39: *Flußdiagramme für das Join-Protokoll*

In den meisten Fällen wird das Service-Objekt aus mehreren Java-Dateien bestehen. Durch Verwendung des *jar* Werkzeuges kann eine einzelne Datei erzeugt werden. Der Extended Server kann dabei durch eine Namenskonvention der Pakete (z.B. *package my_project.extended_server*) und einen festen Namen *(xserver.class)* spezifiziert werden und zusätzlich mit einem optionalen Start-Skript *(xstart)* versehen sein.

Zusammenfassend kann man sagen, daß der bei Jini definierte Discovery/Join-Mechanismus leicht modifiziert werden muß, um den Anforderungen der Prozeßebene und den dort vorherrschenden Kommunikationssystemen gerecht zu werden. Die wichtigste Änderung betrifft dabei die Umkehrung des Discovery/Join Prozesses, der nun vom Lookup-Dienst selbst angestoßen wird.

5.3.4 Konzeption des Lookup-Dienstes und des Lookup-Protokolls

Nach dem Start und nachdem der Lookup-Server an den OPC-Server gebunden ist, prüft dieser in periodischen Abständen, ob neue Service-Objekte vorhanden sind und fügt diese bei Bedarf seiner internen Datenbasis hinzu. Außerdem wartet er auf Lookup-Anfragen von Clients. Um seitens des Clients so unabhängig wie möglich von einer speziellen Hardware- und Betriebssystemumgebung zu sein, wird jede Kommunikationsverbindung außerhalb der Prozeßebene über CORBA oder das HTTP-Protokoll [46,47] (vgl. Bild 37) abgewickelt. Damit genügt es, daß potentielle Clients über eine TCP/IP-basierte Verbindung Kontakt mit dem OPC-Server haben. Um einen CORBA-basierten Zugang zum OPC-Server zu ermöglichen, wird die Verwendung einer CORBA/DCOM-Bridge vorgeschlagen, wie sie beispielsweise Visual Edge [93] anbietet. Es ist anzumerken, daß diese Lösung auch Java-RMI Verbindungen (vgl. Kapitel 2.5.4) zum OPC-Server über die DCOM/CORBA-Bridge beinhaltet. Eine rein Java-basierte Lösung ohne die Verwendung der Bridge ist nicht möglich, da der Zugriff auf den DCOM-basierten OPC-Server realisiert werden muß (siehe hierzu auch [39,54]).

Clients verbinden sich mit dem Lookup-Service über wohldefinierte Schnittstellen. Details können in der Jini-Spezifikation nachgelesen werden und werden an dieser Stelle ausgelassen. An dieser Stelle soll eine alternative, WWW-basierte Methode zur Verbindung mit dem Lookup-Service vorgestellt werden, die integraler Bestandteil des gesamten Frameworks sein soll (Bild 40).

Dabei verbinden sich Clients zunächst via HTTP mit der Startseite eines Webservers. Die Startseite enthält HTML-Code und ein Java-Applet, das einen auf einer graphischen Oberfläche basierenden Lookup-Client darstellt. Wenn der Lookup-Client im Web-Browser des Client ausgeführt wird, so baut dieser eine Kommunikationsverbindung zum Lookup-Dienst auf und listet die dort verfügbaren Service-Objekte. Service-Objekte, die in Form von Java-Applets vorliegen, können geladen und direkt im Web-Browser ausgeführt werden. Andernfalls werden die Service-Objekte via HTTP zur späteren Ausführung auf die Festplatte des Client-Rechners geladen.

Falls ein Service-Objekt in Verbindung mit einem Extended Server verwendet wird, so kann dieser durch den Lookup-Dienst entweder nach dem Join-Prozeß gestartet werden oder bei der ersten Anfrage durch einen Client. Die Aufgabe der Synchronisation oder das Starten weiterer Extended Server, z.B. je einen per Client, fällt in die Verantwortung des Extended Server selbst. Falls der Extended Server als CORBA-Server implementiert ist, so wird dies automatisch durch das zugrundeliegende CORBA-System erledigt und es entfällt der entsprechende Implementierungsaufwand im Service-Objekt.

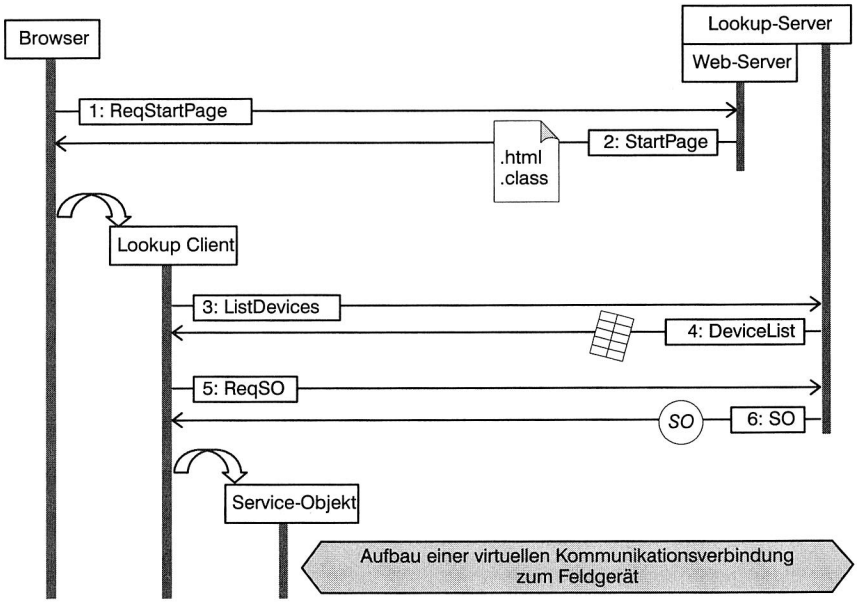


Bild 40: *Der WWW-basierte Zugriff auf den Lookup-Service*

5.3.5 Die Service-Objekt-Kommunikation

Die Jini-Architektur schreibt absichtlich kein Protokoll zwischen dem Service-Objekt und dem Service-Provider vor, um seitens der Implementierung die größtmögliche Flexibilität zu gewährleisten. Andererseits kann dies zu beträchtlichem Implementierungsaufwand führen, nämlich dann, wenn jedes Service-Objekt sein eigenes TCP/IP-basiertes Protokoll verwendet. Gerade im Bereich der prozeßnahen Kommunikation vor dem Hintergrund der geringen Speicherkapazität in Feldgeräten ist es wünschenswert, ein eher anwendungsorientiertes Basisprotokoll einzusetzen.

Im Rahmen des Jini-Frameworks wird deshalb ein CORBA-basiertes Protokoll als Basis vorgeschlagen. Wie bereits angemerkt, wird damit auch die Kommunikation über Java-RMI abgedeckt. Zudem ergibt sich nur ein geringer Aufwand für die Implementierung der tatsächlichen Kommunikation. In Verbindung mit der Programmiersprache Java für die Erstellung der Service-Objekte resultieren relativ kleine Programme, die ca. 50–100 Kbyte inklusive graphischer Benutzeroberfläche umfassen. Die Verwendung der CORBA/DCOM-Bridge ermöglicht in diesem Zusammenhang die Nutzung der kompletten Funktionalität des OPC-Servers über dessen *Automation-Interface* – entweder direkt durch das Service-Objekt oder aber durch den dazwischengeschalteten Extended Server.

5.3.6 Speicherung der Komponenten auf dem Feldgerät

Die Speicherung der Java-Komponenten auf dem EPROM erfolgt zusammen mit dem Brennvorgang des eigentlichen Steuerungsprogramms, indem die Komponenten in den noch freien Speicher geschrieben werden. Eine feldbusspezifische Bibliothek realisiert die Funktionen zum Lesen der Daten und der Übertragung zum Gateway ("Upload"). Dazu muß die Anfangsadresse im EPROM und die Codegröße der Java-Komponente bekannt sein und in das Steuerungsprogramm einkompiliert werden (Bild 41).

Bei der Verwendung eines Flash-EPROM ist die Ersetzung der Java-Komponenten im Produktivbetrieb im Zuge eines Updates möglich. Die komfortabelste Lösung ist die Übertragung der neuen Java-Komponenten über das Feldbussystem, wobei das Feldgerät den Update selbst ausführt. Hierzu muß allerdings das Feldbussystem einen zum "Upload" äquivalenten "Download"-Mechanismus bereitstellen, der das versenden und evtl. notwendige segmentieren eines größeren, zusammenhängenden Datenblocks erlaubt. Falls allerdings ein solcher Datenübertragungsdienst nicht zur Verfügung steht, so kann unter Zuhilfenahme zweier ausgezeichneter Prozeßvariablen, ähnlich wie in Kapitel 5.3.3 bei der Implementierung des Join-Protokolls beschrieben, der Dienst auf praktisch jedem beliebigen Feldbussystem nachgebildet werden. Die Bibliothek zur Datenübertragung beinhaltet dann die notwendigen Dienste sowie das Programm zur Speicherung der neuen Java-Komponenten im EPROM.

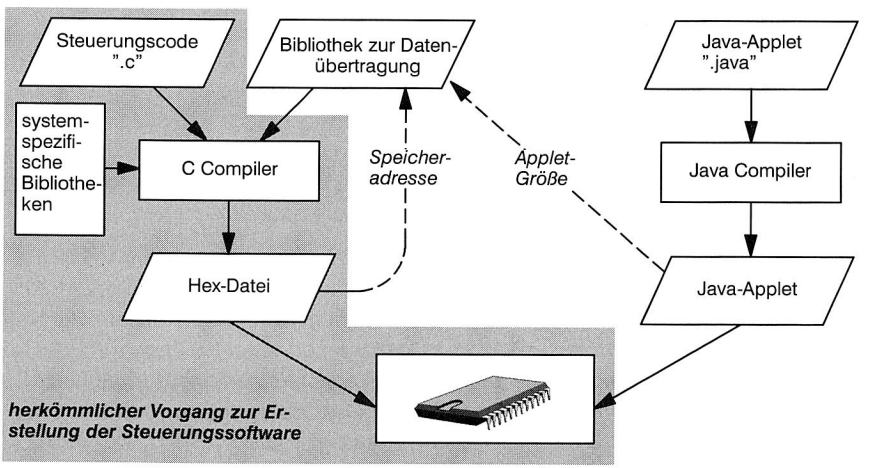


Bild 41: *Speicherung der Java-Komponenten auf dem Feldgerät*

5.4 Beispielhafte Realisierung der Steuerung und Überwachung von NC-Achsen

Das Jini-basierte Framework inklusive des WWW-basierten Lookup-Service wurde am Lehrstuhl implementiert und zur Steuerung von NC-Achsen über das Internet eingesetzt. Als Feldbussystem kam PROFIBUS-FMS zum Einsatz, da aufgrund der Vielzahl an verfügbaren Produktivdatendiensten die Implementierung eines Upload-Dienstes besonders einfach möglich ist. Die Service-Objekte wurden als Java-Applets konzipiert, so daß die Ausführung im Web-Browser möglich ist.

Zur effektiven Unterstützung der Konfiguration des Lookup-Service wurde der "Leitstand-Monitor" entworfen (vgl. Bild 42). Dieser führt beim Start die für die Erstkonfiguration des Lookup-Service notwendigen Operationen aus. Zum einen legt er die Lookup-Datenbank an, in der anschließend die Service Objekte und deren Attribute gespeichert werden. Zum anderen führt er einen initialen Lookup aus, um die Datenbank mit den aktuell verfügbaren Service Objekten zu füllen. Der Leitstand-Monitor beinhaltet einen minimalen Web-Server und kann somit den WWW-basierten Lookup selbst anbieten.

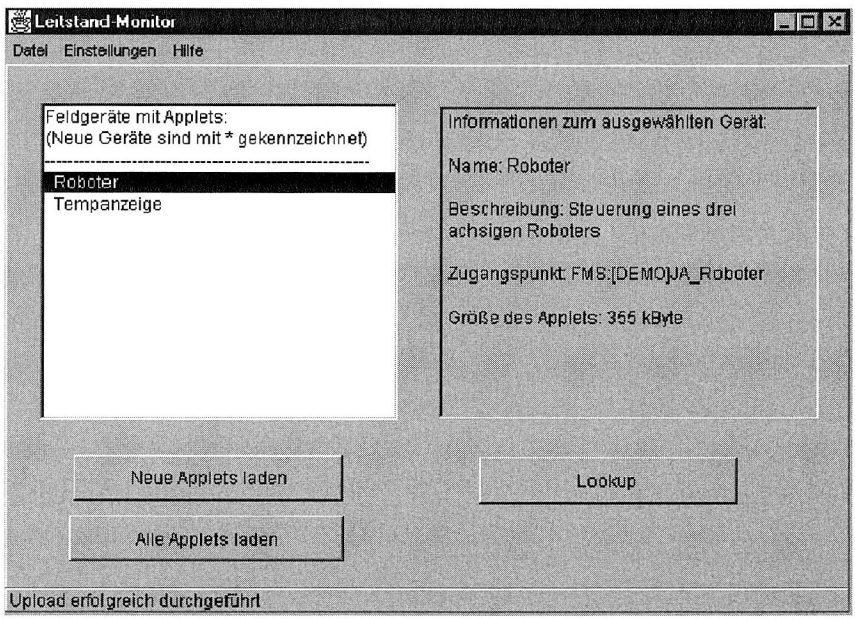


Bild 42: *Die graphische Oberfläche des Leitstand-Monitors*

In der Lookup-Datenbank wird für jedes in einem Feldgerät gespeicherte Java-Applet ein Datensatz mit folgenden Informationen angelegt:

- *Verbindungsname*: Eine Zeichenkette, die die Lokalisierung des Feldgerätes über den OPC-Server ermöglicht
- *Name*: Eine maximal 20 Zeichen lange Zeichenkette, die dem "Namen" des hinterlegten Service-Objekts entspricht
- *Beschreibung*: Eine maximal 100 Zeichen lange Zeichenkette, die eine textuelle Beschreibung der Funktion des Service-Objekts gibt
- *Index*: Ein Zeiger auf die Startadresse des Service-Objekts im EPROM des Feldgeräts
- *Size*: Zeigt die Größe des Service-Objekts in Kbyte an

Diese werden aus den speziellen *JA_xxx* Prozeßvariablen gewonnen, die genau der Übermittlung dieser Informationen dienen. Speziell beim Feldbussystem PROFIBUS-FMS muß, entweder im Rahmen der Projektierung oder durch einen zur Laufzeit generierten Eintrag im Objektverzeichnis, diese Variable dem System "bekannt" gemacht werden, damit der Namensraum des OPC-Servers nach den speziellen Variablen durchsucht werden kann. Dabei ist zu beachten, daß Variablennamen innerhalb des PROFIBUS-Netzes immer eindeutig sein müssen. Zudem dürfen "normale" Prozeßvariablen nicht mit der reservierten Buchstabenkombination beginnen.

Ein potentieller Client verbindet sich via Web-Browser mit dem Rechner, auf dem der Web-basierte Lookup-Service verfügbar ist und ruft eine spezielle Startseite auf. Diese enthält ein Java-Applet, das eine CORBA-Verbindung zum eigentlichen Lookup-Service aufbaut. Dabei werden dem Client diejenigen Feldgeräte des Feldbussystems angezeigt, die über Service-Objekte verfügen. An Hand der Beschreibungen kann der Benutzer ein entsprechendes Service-Objekt auswählen und dieses Laden. In der gewählten Implementierung handelt es sich um Java-Applets, die direkt im Web-Browser ausgeführt werden können. Das Applet baut nun seinerseits eine exklusiv genutzte CORBA-Verbindung zum Gateway auf und bietet die zur Achssteuerung notwendige Funktionalitäten (vgl. Bild 43).

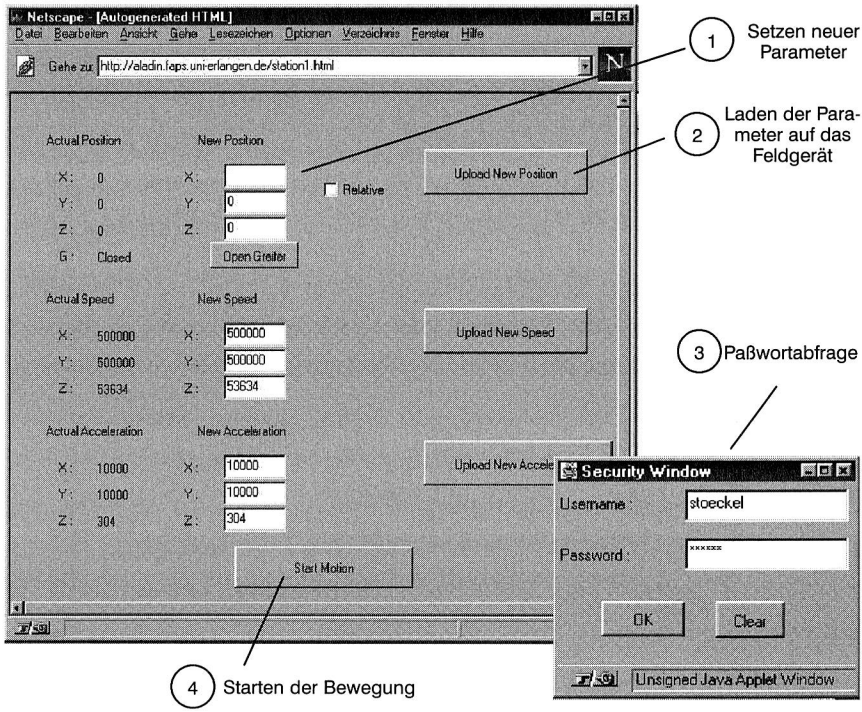


Bild 43: *Realisierung der Steuerung und Überwachung von NC-Achsen*

5.5 Zusammenfassung und Bewertung

Das vorgestellte, auf Jini basierende Framework deckt Aspekte der Implementierungsebene wie der Ebene der Anwendungslogik ab. Seine Stärken liegen darin, daß keinerlei Installationsaufwand beim Client anfällt und die Programme bereits geräte- und anwendungsspezifisch im Feldgerät vorliegen, wodurch ein echtes "Plug and Play" der Feldgeräte ermöglicht wird. Die Bedienoberfläche ist dabei für jeden Client einheitlich – egal ob es sich um ein am Leitstand befindliches Bedien- und Beobachtungssystem handelt, in das die Service-Objekte integriert werden, oder um ein im Feld eingesetztes Handterminal zur Geräteparametrierung. Durch die Nutzung von OPC können alle derzeit relevanten Feldbussysteme gleichermaßen angesprochen werden, so daß sich beim Übergang auf ein anderes Feldbussystem die notwendigen Anpassungen auf den Upload-Dienst des Feldgeräts beschränken. Das Service-Objekt und der Extended Server sind sofort lauffähig. Überhaupt beschränken sich die Erweiterungen auf Seiten der Feldgeräte auf die Funktionalität des Upload-Dienstes.

Im folgenden wird eine detaillierte Bewertung hinsichtlich der in Kapitel 2 formulierten Anforderungen vorgenommen.

Anforderung K1: Unterstützung der vertikalen Kommunikation 

Das Service-Objekt stellt eine virtuelle Kommunikationsverbindung zu dem Feldgerät her, von dem es ursprünglich geladen wurde. Dabei erfolgt am OPC-Server eine Abbildung auf OPC-Items, über die das Protokoll vom und zum Feldgerät abgewickelt wird. Das Feldgerät kann im Rahmen es Protokolls Meldungen an die verbundenen Clients senden. Es ist ihm jedoch nicht möglich, Verbindungen aktiv herzustellen.

Anforderung K2: Unterstützung der horizontalen Kommunikation 

Die horizontale Kommunikation wird nicht explizit unterstützt.

Anforderung K3: Unterstützung für ein verteiltes Client/Server-Modell 

Das Client/Server-Modell wird durch die möglichen Interaktion zwischen dem Service-Objekt und dem Feldgerät festgelegt. In seiner Komplexität ist es allerdings, durch die Abbildung auf Items des OPC-Servers, beschränkt.

Anforderung M1: Umfassendes, generisches Modellkonzept 

Die Modellierung ist nicht Bestandteil des Jini-Framework. Daher wird keine Bewertung hinsichtlich dieses Kriteriums vorgenommen. Um in Bezug auf die angebotenen Daten und Schnittstellen einheitlich aufgebaute Service-Objekte zu gewährleisten, wäre beim Programmentwurf der Einsatz eines geeigneten Modellkonzepts jedoch sinnvoll.

Anforderung M2: Erweiterbarkeit 

Da die Modellierung nicht Gegenstand des Konzepts ist, wird keine Bewertung hinsichtlich dieses Kriteriums vorgenommen.

Anforderung M3: Leichte Integrierbarkeit in bestehende Anwendungen 

Die technische Integrierbarkeit hängt von der Implementierung der Service-Objekte ab, die dem Entwickler frei steht. Mit Java-Beans kann eine Java-basierte Komponententechnologie eingesetzt werden. Java-Applets haben den Vorteil, daß sie direkt im Web-Browser eines Clients ausgeführt werden können. Bietet das Feldgerät genügend Speicherplatz, so können verschiedene Implementierungen ansonsten funktional gleichwertiger Service-Objekte nebeneinander angeboten werden. Auf semantischer Ebene ist es ebenfalls abhängig von der Implementierung der Service-Objekte, welche Vorkenntnisse der Anwender besitzen muß. In der Bewertung ergibt sich letztlich ein Abschlag, da in jedem Fall eine Java-Plattform vorausgesetzt ist.

Anforderung M4: Selbstbeschreibung 

Da die Modellierung nicht Gegenstand des Konzepts ist, wird keine Bewertung hinsichtlich dieses Kriteriums vorgenommen.

6 FIMO – Fieldbus Messaging ORB

Die in den vorangegangenen Kapiteln beschrieben Konzepte sind bereits im Bereich der Middleware-Frameworks einzuordnen. Sie bieten den Anwendungen eine anwendungsspezifische Softwareumgebung an, erlauben jedoch keinen Zugriff auf Dienste der darunterliegende Middleware, insbesondere auf die zum Feldgerät bestehende virtuelle Kommunikationsverbindung. Das Ziel des nun vorgeschlagenen Konzepts ist es, die Integration der Prozeßebene auf Implementierungsebene durch eine einheitliche Middleware zu erreichen. Insbesondere können damit zwei Nachteile der beiden anderen Konzepte umgangen werden: die fehlende Unterstützung der horizontalen Kommunikation und die Beschränkungen, die sich aus der Verwendung des OPC-Servers als Zugangspunkt zum Feldbussystem ergeben. Da die Kommunikation bisher über Items des OPC-Servers abgewickelt wurde, waren die resultierenden Client/Server-Anwendungen auch entsprechend komplex, mußten sie doch das komplette Protokoll fest im Programmcode hinterlegen. Hier verspricht der Einsatz von Middleware eine einfachere Programmerstellung.

6.1 Problemstellung und Zielsetzung

Während die bisher vorgestellten Konzepte aufgrund ihres engen Bezugs zu konkreten Anwendungsproblemen bereits den Middleware-Frameworks zuzuordnen sind (vgl. Definition 3, Kapitel 2.2.2), soll nun die Integration der Prozeßebene auf Implementierungsebene durch eine einheitliche Middleware erreicht werden. Indem die Middleware-Dienste den Anwendungen direkt zur Verfügung stehen, wird eine Reduktion der Komplexität der Clients und der Server erzielt, da diese nicht mehr die komplette Protokollabwicklung im Programmcode selbst vornehmen müssen. Mit dem Konzept wird zudem das Ziel verfolgt, die horizontale Kommunikation effektiv zu unterstützen.

Eine Ersetzung der bisher eingesetzten Kommunikationsprotokolle der Prozeßebene, also der Feldbussysteme, soll dabei nicht stattfinden. Vielmehr soll die Middleware auf den existierenden Kommunikationssystemen aufsetzen, um so

- den Aufwand für Änderungen in den Anwendungen der Prozeßebene gering zu halten,
- für die Echtzeitdatenkommunikation der Teilnehmer innerhalb des Kommunikationssystems die herkömmlichen und bewährten Dienste verfügbar zu haben sowie
- im Sinne einer Migration den schrittweisen Übergang auf die neuen Dienste zu ermöglichen und so "alte" und "neue" Lösungen parallel betreiben zu können.

Für die Implementierung kommen prinzipiell nur die beiden am weitesten verbreiteten Middleware-Architekturen in Frage: COM oder CORBA. Java-RMI entfällt wegen der

Sprachbindung an Java, die im Umfeld der Feldgeräteprogrammierung keine Rolle spielt. Aufgrund der breiten Verfügbarkeit für die verschiedensten Rechner- und Betriebssystem-Plattformen und der frei verfügbaren Spezifikationen fiel die Entscheidung bei der Umsetzung des Konzepts auf die Middleware CORBA. Damit ist das Ziel die Entwicklung einer minimalen, CORBA-konformen ORB Implementierung sowie angepaßter Systemdienste für Feldkommunikationssysteme.

Die folgenden Anforderungen wurden an die Middleware-Implementierung gestellt:

Dienste

Die Middleware soll in erster Linie der Integration der Prozeßebene in die darüberliegenden Ebenen dienen und nicht der Kommunikation von Feldgeräten, die am selben Feldbus angeschlossen sind. Die dafür bereitzustellen Dienste müssen nicht den Anforderungen an eine Echtzeitdatenübertragung erfüllen und sind daher als nicht zeitkritisch anzusehen.

Verhalten im Fehlerfall

Einige Feldbussysteme, wie der PROFIBUS-DP, besitzen mit einem ausgezeichneten Feldgerät, dem Feldbus-Master, einen *Single Point of Failure* – fällt diese Station aus, so können die anderen angeschlossenen Geräte nicht miteinander kommunizieren. Bei anderen Feldbussystemen, beispielsweise WorldFIP, können andere Feldgeräte die Aufgabe übernehmen, so daß die Produktivdatenkommunikation, bis auf eine kleine Verzögerungszeit für das Umschalten auf die neue Station, reibungslos weiterlaufen kann. Aufgrund dieser und weiterer Unterschiede der einzelnen Feldbussysteme im Fehlerfall, wird mindestens gefordert, daß die Middleware-Implementierung kein schlechteres Fehlerverhalten aufweisen soll als das darunterliegende Feldbussystem.

Übertragen auf das Feldbussystem FIP bedeutet dies, daß dort die Middleware-Dienste nicht ausschließlich in einem ausgezeichneten Feldgerät konzentriert werden dürfen. Dies impliziert auch, daß für die verschiedenen Feldbussysteme in der Regel *angepaßte Middleware-Implementierungen* angeboten werden müssen, um im Fehlerfall das geforderte Systemverhalten garantieren zu können.

Sicherheitskonzept

Es wird prinzipiell davon ausgegangen, daß die Prozeßebene innerhalb eines bereits gesicherten Bereichs eines unternehmensinternen Netzwerkes liegt. Zusätzliche Sicherheitsmaßnahmen können am Übergang zur darüberliegenden Zellen- bzw. Leitebene getroffen werden. Die explizite Berücksichtigung eines Sicherheitskonzepts bei der Konzeption der Middleware-Implementierung wird daher nicht gefordert.

6.2 Spezielle CORBA-Spezifikationen

6.2.1 Die "minimumCORBA"-Spezifikation

Die Implementierung eines zum CORBA-Standard konformen ORB ist äußerst umfangreich und bezüglich der zur Laufzeit notwendigen Ressourcen auch nur für entsprechend leistungsfähige Systeme sinnvoll. Um die Vorteile eines einheitlichen, verteilten Objektmodells auch für Embedded Systeme nutzen zu können, wurde 1997 von der OMG ein *Request for Proposol* (RFP) für eine *minimale CORBA Spezifikation* ("minimumCORBA", [94]) ausgerufen. Die Vorstellung ist, daß derartige Systeme nicht die Ressourcen für die Umsetzung der vollständigen Spezifikation bieten, andererseits aber auch zur Laufzeit nicht eine vergleichbare Flexibilität wie Client-Server-Anwendungen in der Bürowelt aufweisen müssen. Daraus resultierte 1998 ein gemeinsam von den Firmen Alcatel, Hewlett-Packard, Inprise, Iona, Lucent, Northern und Sun vertretener Vorschlag, basierend auf der damals aktuellen CORBA-Spezifikation 2.2.

Bereits im RFP wurden einige Eigenschaften der minimumCORBA-Spezifikation als verpflichtend festgelegt. Beispielsweise muß es sich um eine echte Untermenge des CORBA-Standards und der zugehörigen IDL handeln. Darüber hinaus müssen statische Stubs und Skeletons unterstützt werden. Embedded-Systeme reservieren die benötigten Ressourcen größtenteils beim Start des Systems statisch, so daß zur Laufzeit dynamische Funktionen für das Erzeugen und Aktivieren neuer Objekte oder das Abfragen von Objekten selten bis nie genutzt werden. Daher war es das Ziel der minimumCORBA-Spezifikation zum einen die dynamischen Dienste so weit wie möglich einzuschränken, aber andererseits vollständig kompatibel und in jedem Fall interoperabel zum bestehenden CORBA-Standard zu sein. Der zweite Punkt ist deshalb äußerst wichtig, da die aus der Spezifikation resultierenden Embedded Systeme als Teil eines Gesamtsystems auch mit normalen CORBA ORBs kommunizieren können müssen. Vor diesem Hintergrund ist auch die Tatsache zu sehen, daß die minimumCORBA-Spezifikation die komplette IDL unterstützt.

Die nachfolgende Liste beschreibt die wichtigsten Einschränkungen, die im Rahmen der minimumCORBA-Spezifikation getroffen wurden:

- keine Unterstützung des Dynamic Invocation Interface und des Dynamic Skeleton Interface
- keine Unterstützung des dynamischen *any* Datentyps ("DynAny")
- keine Unterstützung des Interface Repository, mit Ausnahme der "RepositoryIds" und der TypeCode-Schnittstelle; letztere wird für den *any* Datentyp benötigt
- keine Unterstützung für die dynamischen Teile der TypeCode-Schnittstelle
- eingeschränkte Unterstützung des Portable Object Adapters (u.a. kein "AdapterActivator" und kein "ServantManager" sowie Einschränkungen in den "Policies")

- keine Unterstützung für die Interoperabilität von COM und CORBA
- keine Unterstützung für Interceptors

Konforme minimumCORBA-Implementierungen können darüber hinaus noch weitere Einschränkungen vornehmen:

- Typprüfungen: Typprüfungen beim "any" Datentyp, bei den TypeCodes und beim "narrow" von Objektreferenzen können entfallen
- Ausnahmen (Exceptions): Unterstützung für Systemausnahmen (system exception) und benutzerdefinierte Ausnahmen (user exception) können entfallen, wenn sie im Programmcode nicht vorkommen
- Vererbung: Die Mehrfachvererbung muß nicht unterstützt werden.

6.2.2 Die "Real-time CORBA"-Spezifikation

Im Gegensatz zur minimumCORBA-Spezifikation, hat die Real-time CORBA-Spezifikation ("RT–CORBA", [95]) die *Erweiterung* des CORBA-Standards zum Ziel, um verteilte echtzeitfähige Anwendungen entwickeln zu können. Entwickler von Echtzeitsystemen müssen über erweiterte Schnittstellen und Methoden verfügen, um Ressourcen reservieren und freigeben zu können und um Aussagen über die Ausführungszeiten, beispielsweise von Methodenaufrufen, treffen zu können. Dabei genügt es nicht, die RT-CORBA-Spezifikation isoliert zu betrachten. Um ein Echtzeitsystem auf RT-CORBA-Basis zu erhalten, müssen mindestens folgende Voraussetzungen erfüllt sein:

- prioritätenbasierte Scheduling-Mechanismen des Betriebssystems
- eine RT-CORBA Implementierung
- ein Kommunikationssystem, das die Vorhersagbarkeit der Übertragung unterstützt
- speziell implementierte Echtzeitanwendungen

Aufgrund der vielfältigen Voraussetzungen bei der Entwicklung "harter" Echtzeitsysteme und der eng miteinander verzahnten Funktionalitäten, beginnend beim Kommunikationssystem über das Betriebssystem bis zur konkreten ORB-Implementierung, zielt die RT-CORBA-Spezifikation auch nicht mehr primär auf die Interoperabilität von Echtzeit- und Nicht-Echtzeit-Anwendungen ab, sondern stellt die Entwicklung verteilter Echtzeit-Anwendungen in den Mittelpunkt. Für die weiteren Ausführungen spielt die RT-CORBA Spezifikation deshalb eine untergeordnete Rolle, weshalb für Details auf die Spezifikation verwiesen wird.

6.3 Vorstellung der Architektur

FIMO (Fieldbus Messaging ORB) definiert eine zu CORBA 2.1 [96] konforme Middleware-Architektur für Feldbussysteme. Im Vergleich zu herkömmlichen CORBA-Imple-

mentierungen, mußten bei der Entwicklung von FIMO die Besonderheiten der Geräte der Prozeßebene genau berücksichtigt werden. Dabei galt es, vor allem die typischerweise geringen Speicherressourcen zu beachten, die breite Spanne an Betriebssystemen, die von multitaskfähigen Echtzeitbetriebssystemen bis hin zu Single-task-Betriebssystemen reicht, sowie die eingeschränkten Möglichkeiten bei der Wahl der in Frage kommenden Programmiersprachen. Die FIMO-Architektur trägt diesen Punkten Rechnung, indem von nur minimalen Anforderungen an die Speicherressourcen und an das Betriebssystem ausgegangen wird. Als Programmiersprache wird "C" vorausgesetzt.

FIMO basiert in weiten Teilen auf der minimumCORBA-Spezifikation und verzichtet deshalb u.a. auf die Implementierung des Interface und Implementation Repository, des Dynamic Invocation Interface und des Dynamic Skeleton Interface. Der Datentyp *any* und Mehrfachvererbung werden nicht unterstützt.

Im Vergleich zu minimumCORBA werden folgende weitere Einschränkungen vorgenommen:

- Keine Unterstützung für den Datentyp *long long*, für Gleitkommazahlen vom Typ *fixed* und für die Zeichen(ketten)-Typen *wchar* und *wstring*
- Keine Unterstützung für *Contexte*
- Eingeschränkte Aktivierungsstrategien des Basic Object Adapters (BOA)[4], die einem Single-task-Betriebssystem Rechnung tragen

Andererseits weist FIMO jedoch zwei zusätzliche Kerndienste auf:

- Ein *verteilter Namensdienst* ermöglicht es, FIMO-Objekte innerhalb der Prozeßebene und CORBA-Objekte außerhalb der Prozeßebene über ihren Namen aufzulösen.
- Zum anderen ermöglicht die *FIMO-Bridge*, die typischerweise auf denjenigen Feldgeräten zum Einsatz kommt, die mit der darüberliegenden Zellenebene verbunden sind, Verbindung mit CORBA-konformen ORBs außerhalb der Prozeßebene aufzunehmen.

Eine erste konzeptionelle Übersicht der Architektur gibt Bild 44.

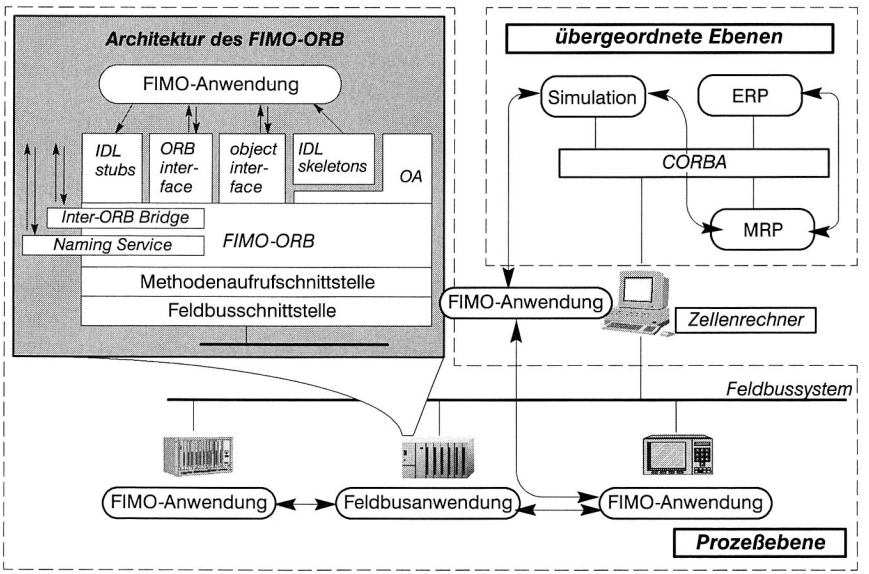
Die FIMO-Kommunikation wird über die *Feldbusschnittstelle* auf den darunterliegenden Feldbus abgebildet. Es ist die Aufgabe der konkreten Implementierung der Feldbusschnittstelle sicherzustellen, daß die FIMO-Kommunikation keine Auswirkungen auf die normale Prozeßdatenkommunikation des Feldbussystems hat, um so dem wichtigen Entwurfskriterium Rechnung zu tragen, daß die Feldbuskommunikation parallel zur FIMO-Kommunikation betrieben werden kann.

4. minimumCORBA basiert bereits auf dem CORBA-Standard 2.2, in der der *Basic Object Adapter* durch den *Portable Object Adapter* (POA) ersetzt wurde.

6.4 Konzeption der einzelnen FIMO-Dienste

6.4.1 Die Feldbusschnittstelle

Die Feldbusschnittstelle stellt den einzigen feldbusabhängigen Teil der FIMO-Architektur dar, indem es den Feldbuszugriff kapselt. Es beinhaltet daher grundlegende Methoden zur Kontrolle des Feldbusses, z.B. für die Initialisierung, das Starten und das Stoppen des Bussystems, und ein autonomes Verbindungsmanagement, um Verbindungen zu entfernten Feldgeräten herzustellen bzw. zu beenden. Zudem wird eine einfache Nachrichtenschnittstelle zur Verfügung gestellt, die im wesentlichen dem Marshaling und Unmarshaling der Funktionsaufrufe und der Basisdatentypen dient. Wie bereits angedeutet, unterstützt FIMO verschiedene Feldbussysteme durch individuell angepaßte Implementierungen der Feldbusschnittstelle. Dabei ist eine Abbildung auf die jeweils vorhandenen Kommunikationsdienste vorzunehmen. Eine explizite Diskussion der Schnittstellen für die Initialisierung und das Verbindungsmanagement soll an dieser Stelle nicht vorgenommen werden. Stattdessen wird die Nachrichtenschnittstelle genauer vorgestellt, die RPC-ähnliche Funktionalität aufweist.



MRP: Manufacturing Resource Planning
ERP: Enterprise Resource Planning
◄──► Anwendungsintegration auf Basis von CORBA

Bild 44: *Die FIMO-Architektur im Überblick*

Die Nachrichtenschnittstelle stellt Methoden für die Zusammenstellung, die Zerlegung und den Transfer von Anfrage-, Antwort- und Abbruchnachrichten (request/reply/cancel) bereit. Der Transfer erfolgt unsynchronisiert, d.h. die Aufgabe der Zuordnung einer Anfrage zu einer Antwort ist einer höheren Schicht der FIMO-Architektur überlassen, der *Methodenaufrufschnittstelle*, die seitens der Anwendungen benutzt wird (auf diese wird im weiteren Verlauf nicht genauer eingegangen, da sie konzeptionell nur eine Kapselung der Feldbusschnittstelle darstellt). Zu diesem Zweck wird eine eindeutige Kennung mit jeder Anfrage übertragen, die in der korrespondierenden Antwort wiederzufinden ist. Üblicherweise werden mit einer Anfrage bzw. einer Antwort weitere Parameter übertragen. In CORBA wird dabei zwischen *in*, *out* und *inout* Parametern unterschieden – je nachdem, ob diese nur Aufrufparameter, Rückgabeparameter oder beides sind. Diese Parameter müssen zur eigentlichen Nachricht angefügt werden und mitübertragen werden. Dazu ist eine Marshaling/Unmarshaling Schnittstelle für die CORBA-Basisdatentypen spezifiziert. Diese stellt die notwendigen *add* und *get* Funktionen zur Verfügung, um Parameter anzuhängen bzw. auf der Gegenseite zu extrahieren. Falls ein Methodenaufruf fehlschlägt, beispielsweise weil das Server-Objekt nicht mehr existiert oder der Client den Aufruf nicht korrekt abgesetzt hat, wird eine systemeigene bzw. benutzerdefinierte Ausnahme angestoßen. Die entsprechenden Schnittstellen, um Ausnahmen zu übertragen und zu behandeln, werden ebenso in der Feldbusschnittstelle reflektiert. Bild 45 zeigt schematisch die Funktionsweise der Feldbusschnittstelle im Falle eines Methodenaufrufs.

Bei der Implementierung der Marshaling/Unmarshaling Schnittstelle können die bei Feldbussystemen vordefinierten Basisdatentypen genutzt werden, um zumindest für die CORBA-Basistypen eine aufwendige Umsetzung zu umgehen. In diesem Fall müssen lediglich die komplexeren Datentypen, wie *CORBA Sequences*, *Structs* oder *Unions*, aufwendiger auf die Feldbusdatentypen abgebildet werden.

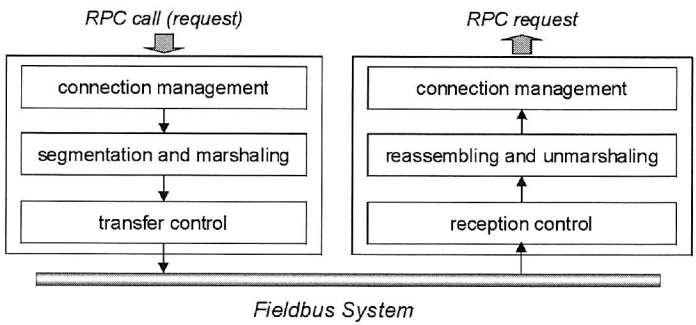


Bild 45: *Die FIMO-Feldbusschnittstelle*

6.4.2 Der FIMO-Dispatcher

Da die FIMO-Kommunikation und die normale Feldbuskommunikation parallel betrieben werden soll, müssen zunächst aus den eingehenden Feldbusnachrichten die FIMO-Nachrichten aussortiert werden. Diese Aufgabe übernimmt der *FIMO-Dispatcher*, dem somit eine zentrale Rolle in der FIMO-Architektur zukommt. Der Dispatcher reagiert auf FIMO-spezifische Nachrichten, indem er intern die entsprechenden Behandlungsroutinen aufruft. Dies wird weiter unten an einem Beispiel erläutert.

In Bild 46 ist der Algorithmus skizziert, den der Dispatcher durchläuft. Bei Aufruf des Dispatchers prüft dieser, ob neue Nachrichten vorhanden sind. Falls es sich dabei um Nachrichten handelt, die im Rahmen der normalen Prozeßdatenkommunikation ausgetauscht werden und nicht FIMO-spezifisch sind, so werden diese an den Anwendungsprozeß weitergegeben[5]. Ansonsten sucht FIMO nach einer zu der Nachricht passenden Behandlungsroutine und stößt diese an.

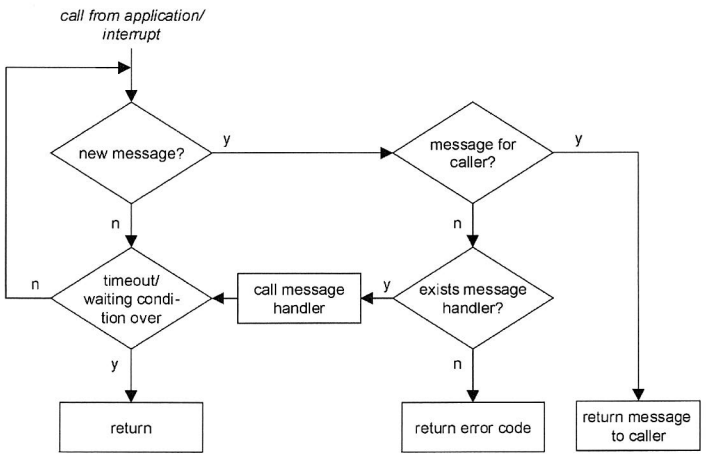


Bild 46: *Der FIMO-Dispatcher*

Falls beispielsweise ein als Server agierendes Feldgerät die Kommunikationsverbindung zum Client schließt, so wird eine entsprechende Nachricht des FIMO-Verbindungsmanagements erzeugt, die die Clients informiert. Der Dispatcher leitet diese Nachricht im Client an dessen Verbindungsmanagement weiter. Sollte bei Aufruf des Dispatchers keine aktuelle Nachricht vorliegen, so kann er derart konfiguriert werden,

5. Dies muß feldbusspezifisch erfolgen. Normalerweise holt eine Anwendung neue Feldbusnachrichten aus einem globalen Eingangspuffer, so daß man entweder die Nachricht wieder in den Puffer zurückschreibt, oder man muß – falls dies nicht unterstützt wird – den Eingangspuffer geeignet kapseln.

daß er eine vorgegebene Zeitspanne blockierend auf Nachrichten und/oder auf die Gültigkeit einer Abbruchbedingung wartet.

Verfügt das Feldgerät über ein multitaskingfähiges Echtzeitbetriebssystem, so stellt der Dispatcher einen entsprechend priorisierten Prozeß dar, der interruptgesteuert bei Eintreffen einer neuen Nachricht angesprungen wird. Andernfalls, d.h. in einem Single-task-Betriebssystem, kann der Dispatcher entweder ebenfalls interruptgesteuert aufgerufen werden oder er muß Anwendungsprogramm regelmäßig aufgerufen werden, falls die Feldbustreiber erstere Möglichkeit nicht vorsehen. Die Modularität der FIMO-Architektur als solche spiegelt sich auch im FIMO-Dispatcher wider. Dieser kann zur Laufzeit flexibel durch das Registrieren neuer Nachrichtenbehandlungsroutinen rekonfiguriert werden.

6.4.3 Die ORB-Schnittstelle

Die ORB-Schnittstelle dient der Initialisierung der CORBA-Funktionalität und der Feldbusschnittstelle. Dabei wird eine an minimumCORBA angelehnte, reduzierte Funktionalität angeboten. Für ein besseres Verständnis der Architektur von FIMO wird die ORB-Schnittstelle etwas detaillierter behandelt. Nachfolgend ist die entsprechende, in Pseudo-IDL (PIDL) formulierte Schnittstellenbeschreibung dargestellt.

```
module CORBA {          // PIDL
   interface OA;
   interface BOA:OA;
   interface BRIDGE;

   typedef sequence<string> arg_list;
   typedef string ORBid;
   typedef string StationType;
   ORB ORB_init(inout arg_list argv,ORBid orb_identifier);

   typedef unsigned short ServiceType;
   typedef unsigned long ServiceOption;
   typedef unsigned long ServiceDetailType;
   struct ServiceDetail {
      ServiceDetailType service_detail;
      Sequence<octet> service_detail;
   };
   struct ServiceInformation {
      sequence<ServiceDetail> service_details;
      sequence<ServiceOption> service_options;
   };
   interface ORB {
      string object_to_string(in Object obj);
      Object string_to_object(in string str);

      boolean get_service_information(in ServiceType service_type,
               out ServiceInformation service_information);
      typedef string ObjectId;
      typedef sequence<ObjectId> ObjectIdList;
      ObjectIdList list_initial_services();
```

```
        Object resolve_inital_references(in ObjectId identifier);

        typedef string OAid;
        BOA BOA_init(inout arg_list argv,in OAid oa_identifier);

        typedef string BridgeID;
        typedef string BridgeType;
        BRIDGE bridge_init(inout arg_list argv,
                  BridgeID bridge_name,
                  BridgeType bridge_type);

        typedef sequence<OA> ObjectAdapters;
        typedef sequence<BRIDGE> Bridges;
        void get_station_context(out StationType station_type,
                  out ObjectAdapters station_list,
                  out Bridges bridge_list);
    };
};
```

Die *CORBA::ORB_init* Methode initialisiert den Feldbus, setzt die feldgerätespezifischen Kommunikationsparameter und registriert das Feldgerät mit seinem Typ und seinem Namen am Namensdienst. ORB-spezifische Optionen werden dabei, konform zur CORBA-Spezifikation, über das Format "–ORB<Kennung> <Wert>" übergeben. Feldbusspezifische Optionen, die intern an die Feldbusschnittstelle weitergegeben werden, mittels des Formats "–FB<Kennung> <Wert>". Der *orb_identifier* enthält die Typbezeichnung des Feldgeräts und dessen Namen wie er im Namensdienst erscheint. Dazu ist der *orb_identifier* im Format "<Typenbezeichnung>&&<Name>" aufzubauen.

Die Methode *ORB::BOA_init* initialisiert den Basic Object Adapter. FIMO assoziiert mit jedem instantiierten Object Adapter (OA) ein Feldgerät ("Station"). Dabei kann jeweils nur *ein* Object Adapter pro Feldgerät instantiiert werden. Der Name des Feldgeräts ergibt sich aus dem *oa_identifier*, der damit dem zweiten Namensteil des *orb_identifier* entspricht.

Mit Hilfe des ORB-Interfaces können auch Inter-ORB-Bridges zu anderen ORBs instantiiert werden. Dies geschieht durch den Aufruf von *ORB::bridge_init*. Die Inter-ORB-Bridge erhält einen Namen und einen Typ und wird damit am Namensdienst registriert.

Beim Methodenaufruf oder zum Aufbau des Namensbaums werden Informationen zu jeder Maschine benötigt: die darauf eingerichteten Stationen und die eventuell vorhandenen Inter-ORB-Bridges. Der Methodenaufruf an *ORB::get_station_context* bietet diese Information anderen Feldgeräten am Feldbus an. Die Methode liefert den Typ der Maschine, wie bei der ORB-Initialisierung angegeben, im Parameter *station_type* und eine Liste der an dieser Maschine angemeldeten Feldgeräte spezifiziert durch ihr *OA-Objekt*. Sind Inter-ORB-Bridges am ORB registriert, wird eine Liste der Inter-ORB-Bridges geliefert.

6.4.4 Die Repräsentation von Objektreferenzen

Mit Hilfe von Objektreferenzen können in einem CORBA-System entfernte Objekte eindeutig bestimmt und angesprochen werden. Die konkrete Implementierung einer Objektreferenz ist ORB-spezifisch, enthält aber in jedem Fall Informationen über den Ort des Objekts, dessen Typ sowie eine Art "Zeiger" auf das Objekt. Nachfolgend ist die Struktur der FIMO-Objektreferenz in PIDL wiedergegeben.

```
module CORBA {// PIDL
   struct Version {
      octet          major;
      octet          minor;
   };
};

module FB {
   struct Station {
      unsigned long    station_key;
   };
   struct ConnOptions_1_0 {
      unsigned long    request_timeout;
   };
   struct OR_light_1_0 {
      CORBA::Version   pbor_version;
      CORBA::Version   obj_version;
      Stationstation_ref;
      string    station_name;
      string    object_key;
      string    object_type;
      string    visible_type;
   };
   struct OR_fat_1_0 {
      OR_light_1_0        light_object_ref;
      ConnOptions_1_0     options;
   };
};
```

In FIMO werden zwei Typen von Objektreferenzen unterschieden. Der vereinfachte ("light") Typ enthält alle Informationen, die bei einem Aufruf benötigt werden. Der erweiterte ("fat") Typ beinhaltet zusätzliche Verbindungsoptionen zu dem bezeichneten Objekt. Beispielsweise kann durch das Setzen eines *request_timeout*, die Zeitdauer angegeben werden, die auf eine Rückantwort bei einem gesendeten Fernaufruf gewartet werden soll, bis das System mit einem Fehler abbricht. Dieser Timeout ist objektabhängig, da die Verarbeitungszeit stark von den Methoden eines Objekts abhängt.

Zur Bezeichnung des Feldgeräts, auf der ein FIMO-Objekt lokalisiert ist, führt die Objektreferenz den Stationsnamen *(station_name)* und eine Stationsreferenz *(station_ref)* mit. Das Objekt selbst wird über den *object_key* referenziert, der dem Objektnamen entspricht, wie er im Namensdienst eingetragen wird. Jede Objektreferenz wird für einen bestimmten Objekttyp generiert. Dieser Typ ist der *object_type*. Durch die *narrow*

Methode kann sich der Typ einer Objektreferenz zur Laufzeit verändern, indem er eine von der Basisklasse abgeleitete Klasse darstellt. Der aktuelle Typ wird daher in *visible_type* hinterlegt.

Die Versionsnummer *pbor_version* bestimmt die ORB-Version. Da sich die Objekt-Referenz in verschiedenen Versionen des ORB unterscheiden kann, trägt die Struktur sowohl im Namen als auch als Parameter die Version des ORB mit sich. Wird das Objekt nach seiner ersten Benutzung verändert und neu compiliert, ändert sich die *obj_version* Versionsnummer, um so anzuzeigen, daß die Objekt-Schnittstellen von Client und Server, die auf die Objekt-Referenz zugreifen bzw. sie implementieren, womöglich inkompatibel geworden sind.

6.4.5 Konzeption des verteilten Namensdienstes

Das Aufrufen von Methoden an entfernten Objekten erfordert die Kenntnis der *Objektreferenz* des Objekts. Die CORBA Spezifikation definiert einen eigenen Dienst, der es erlaubt, Objekte wesentlich komfortabler als dies mit der Objektreferenz möglich wäre, an Hand ihres Namens aufzufinden. Dieser Dienst wird mit *CosNaming Service* [37] bezeichnet und er bietet Schnittstellen, um Objekte mit ihrem Namen zu registrieren bzw. aufzusuchen. FIMO definiert demgegenüber einen eigenen Namensdienst, den *FIMO Naming Service (FNS)*, der auf einem in einigen Punkten vereinfachten *CosNaming Service* basiert und zu diesem kompatibel ist.

Der Hauptunterschied zwischen dem FNS und dem *CosNaming Service* liegt in der Art wie der Dienst erbracht wird. Während der *CosNaming Service* als ein zentraler Dienst konzipiert ist und so auch üblicherweise implementiert wird, wurde der FNS von Beginn an als dezentraler Dienst konzipiert. Ein wichtiger Grund hierfür war die Vermeidung einer zentralen Fehlerinstanz (*single point of failure*), d.h. falls ein Feldgerät ausfällt, so dürfen die anderen Feldgeräte in ihrer Funktion nicht beeinträchtigt werden. Zum anderen würde ein zentraler Namensdienst für das betreffende Feldgerät einen erheblichen Mehraufwand an Ressourcen nach sich ziehen, um alle angeschlossenen Namenskontexte und Objekte verwalten zu können. Ein potentieller Nachteil ist, daß der FNS damit einen Kerndienst darstellt und in jedem Feldgerät implementiert werden muß. Bezüglich der Namensräume ergeben sich im Detail noch weitere Unterschiede zwischen dem Standardnamensdienst und dem FNS. Der FNS unterstützt lediglich dreistufige hierarchische Namen (vgl. Bild 47). Der Root-Kontext spannt die Stationskontexte auf, die in der Regel mit den physikalisch angeschlossenen Feldgeräten identisch sind, die wiederum ihrerseits die einzelnen FIMO-Objekte beinhalten.

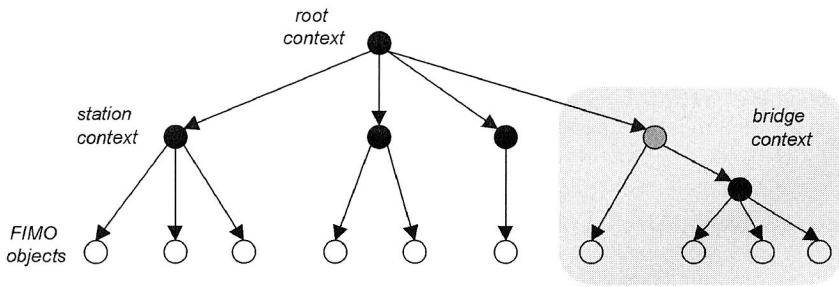


Bild 47: *Struktur des FIMO-Namensdienstes*

Aus Sicht der Implementierung verwaltet jedes Feldgerät seine eigenen Lookup-Tabellen für die Stationskontexte und die FIMO-Objekte. Die dabei zugewiesenen Ressourcen werden statisch konfiguriert, so daß die Lookup-Tabelle eventuell nicht alle Objektrefenzen aufnehmen und damit direkt auflösen kann (Cache-Konzept). Die Lookup-Tabelle eines Feldgeräts wird typischerweise beim Start mit den Stationskontexten aller angeschlossenen Stationen initialisiert, indem das Gerät eine *list*-Operation auf den Root-Context des FNS ausführt. Im laufenden Betrieb ergibt sich dann das Problem der verteilten Cache-Konsistenz, wenn Feldgeräte oder Kommunikationsverbindungen ausfallen bzw. umkonfiguriert werden. Dies würde normalerweise aufwendige Synchronisationsmechanismen nach sich ziehen.

Bei FIMO wird an dieser Stelle ein pragmatischer Ansatz gewählt. Der FNS ist mit dem Verbindungsmanagement gekoppelt und damit in der Lage bei Verbindungsfehlern, also falls Stationen aus dem Netz genommen wurden oder entfernte Objekte gelöscht wurden, eigenständig die internen Tabellen zu aktualisieren. Falls ein Client eine ungültige Objektreferenz benutzt, um mit einem Server zu kommunizieren, so wird eine entsprechende Fehlermeldung generiert und der Client muß das gewünschte Objekt erneut auflösen. Diese Vereinfachung seitens der Implementierung des FNS bzw. der daraus resultierende Mehraufwand auf Client-Seite erscheint gerechtfertigt, da Feldbusanwendungen typischerweise fast alle benötigten Objekte zur Startzeit generieren und während der Betriebsphase ein eher statisches Verhalten zeigen.

6.4.6 Die FIMO-Bridge

Das Ziel der FIMO-Architektur ist eine nahtlose Integration der Prozeßebene in die darüberliegenden Kommunikationsebenen durch eine CORBA-basierte Infrastruktur. Daher muß FIMO selbst in diese CORBA-Infrastruktur integriert werden. Dies wird durch die *FIMO-Bridge* erreicht, die eine CORBA Bridge gemäß der CORBA-Spezifikation 2.1 darstellt. Dabei wird die Bridge in dem Feldgerät implementiert, das mit der Zellen- bzw. Leitebene direkt verbunden ist, so daß sie zwischen FIMO und dem ORB der darüberliegenden Ebene vermitteln kann.

Bild 48 zeigt den Basismechanismus, um Methodenaufrufe an CORBA-Objekte außerhalb der Prozeßebene abzusetzen. Für jede Objektreferenz, die ein FIMO-Objekt vom fremden ORB anfordert, wird ein Pseudo-Proxyobjekt im Feldgerät, das die Bridge implementiert, erzeugt. Eine Objektreferenz auf dieses neu erzeugte Proxyobjekt wird an das aufrufende FIMO-Objekt übergeben. Damit werden automatisch alle Aufrufe des FIMO-Objekts über das Proxyobjekt abgewickelt. Das Proxyobjekt stellt dabei kein Objekt im herkömmlichen Sinn dar, sondern definiert lediglich eine Abbildung von FIMO-Objektrefenzen auf Objektreferenzen des fremden ORB. Die Bridge fängt alle Aufrufe an das Proxyobjekt ab, sucht aus einer internen Lookup-Tabelle die richtige Objektreferenz heraus und leitet den Methodenaufruf über das *Dynamic Invocation Interface* weiter. Da jedoch Objektrefenzen in CORBA generell opaque sind, kann dieses Verfahren nicht angewandt werden, wenn Objekte des fremden ORB Methoden von FIMO-Objekte aufrufen. Stattdessen erzeugt die Bridge in diesem Fall echte Proxyobjekte, die Aufrufe über das *Dynamic Skeleton Interface* entgegennehmen. In beiden Fällen wird das Proxyobjekt erst erzeugt, wenn eine Objektreferenz verlangt wird bzw. eine Methode eines Objekts tatsächlich aufgerufen wird.

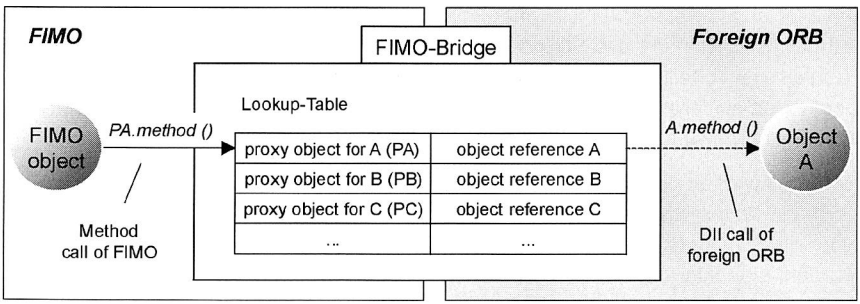


Bild 48: *Proxyobjekte zum Zugriff auf fremde CORBA Objekte*

Eine weitere Aufgabe der Bridge liegt darin, die Namensräume auf die jeweils andere Seite der Bridge zu exportieren, um so zwischen den verschiedenen Namensdiensten zu vermitteln. Dabei ist wiederum zu beachten, daß beide Seiten unterschiedlich behandelt werden müssen.

(1) Auflösen von FIMO-Objekten durch den fremden ORB: Der Zugriff auf FIMO-Objekte durch den fremden ORB wird erreicht, indem der FNS am Namensdienst des fremden ORB registriert wird. Der FNS besitzt zwischen dem Root-Kontext und den tatsächlichen Objekten nur eine Kontextebene, den Bridge- oder Stationskontext. Daher muß die Bridge lediglich Proxyobjekte für die Stationskontexte erzeugen und verwalten. Dies kann bereits während der Startphase der Bridge geschehen, wobei für im laufenden Betrieb neu hinzukommende Feldgeräte automatisch neue Proxyobjekte

erzeugt bzw., falls diese vom Feldbus genommen werden, auch automatisch wieder gelöscht werden müssen.

(2) Auflösen von Objekten des fremden ORB durch FIMO-Objekte: Konzeptionell wird der Namensraum des fremden ORB unter den Bridge-Kontext des Feldgeräts gehängt, das die Bridge implementiert (vgl. Bild 47). Um jedoch den Ressourcenverbrauch so gering wie möglich zu halten, werden keine Proxyobjekte für die Kontextobjekte des fremden ORB bereitgestellt. Stattdessen wird der komplette Pfad durch den Namensraum beginnend vom Bridge-Kontext bis zum tatsächlichen Objekt in die FIMO-Objektreferenz kodiert, welche zur Anfrage der FNS Dienste der Bridge dient.

Bild 49 verdeutlicht an einem Beispiel das Konzept. Dabei versucht ein FIMO-Objekt ein Objekt des fremden ORB über dessen Namen aufzulösen *(resolve)*. Diese Anforderung wird via den lokalen Namensdienst des Feldgeräts zum Namensdienst der Bridge weitergeleitet. Die Objektreferenz auf welcher diese Anfrage basiert enthält den kompletten Pfad, soweit er bereits im fremden ORB traversiert wurde ("/slave control/ property handler"). Tatsächlich weist diese Objektreferenz auf ein Pseudoobjekt der Bridge, das den Pfad zusammen mit dem als Parameter des *resolve* Aufrufs übergebenen Namen unter Zuhilfenahme des Namensdienstes des fremden ORB auflöst. Falls das Ergebnis der Namensauflösung ein weiteres Kontextobjekt darstellt, d.h. einen Knoten im Namensbaum des fremden ORB, wird eine FIMO-Objektreferenz an das aufrufende Objekt zurückgeliefert, das den nun aktuell traversierten Pfad kodiert und wiederum auf ein Pseudoobjekt der Bridge verweist. Falls das Ergebnis der Namensauflösung ein echtes Objekt darstellt, so erzeugt die Bridge ein Proxyobjekt, das es in seine interne Lookup-Tabelle übernimmt und überträgt eine Objektreferenz auf dieses Proxyobjekt an den Aufrufer. In diesem Zusammenhang bleibt anzumerken, daß gemäß der CORBA-Spezifikation *resolve* Aufrufe jeweils an das Kontextobjekt des fremden Namensbaums gerichtet sind, die die Knoten repräsentieren. Im vorliegenden Beispiel ist dies das zum Knoten "property handler" gehörende Kontextobjekt.

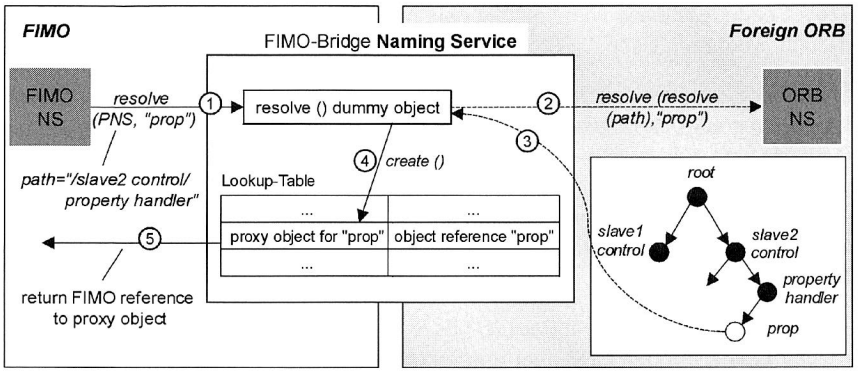


Bild 49: *Auflösung von Objekten des fremden ORB*

6.4.7 Das Objektmodell und der IDL-Compiler

Die FIMO-Architektur wurde insbesondere für Feldgeräte, die unter "C" programmiert werden, konzipiert. Daher muß für diese Programmiersprache ein adäquates Objektmodell zur Verfügung gestellt werden, das in der Lage ist, die in IDL definierten Schnittstellen abzubilden. Ein Objektmodell wird in der CORBA-Spezifikation vorgeschlagen, das u.a. opaque Objektreferenzen und Aufrufkonventionen, als Bestandteil eines erweiterbaren Objektmodells definiert. Das FIMO-Objektmodell verfügt darüber hinaus über *type-check* und *type-cast* zur Laufzeit und spezifiziert für FIMO-Objekte die Möglichkeit eigene Instanzvariablen zu beinhalten.

In Zusammenhang mit dem Objektmodell existieren spezielle FIMO-Schnittstellen, die es auf einfache Weise erlauben, Objekt-Kontainer für einfache oder abgeleitete Objekte zu erzeugen. Letzteres auf der Grundlage eines Ableitungsbaumes, so daß die entsprechende Ableitungshierarchie automatisch im Objekt kodiert wird. Auf Einzelheiten soll an dieser Stelle nicht weiter eingegangen werden. Bild 50 gibt einen grundlegenden Überblick über das Speicherlayout einer Objektinstanz des FIMO-Objekts *class3*, das von *class1* mittels einfacher Vererbung abgeleitet ist. Das Konzept des FIMO-Objektmodells erlaubt es nun insbesondere einen *type-cast* von Unterklassen auf beliebige Oberklassen in einer einzigen *cast*-Operation durchzuführen.

Für die Generierung der Stubs und Skeletons aus der IDL-Definition der Objektschnittstellen ist der FIMO IDL-Compiler verantwortlich. Darüber hinaus erzeugt er die Objektmethoden und die zusätzlich benötigten Methoden für die Objektinstantiierung und -vernichtung, den *type-cast (widen/narrow)* und den *type-check*. Dazu wird eine dem Speicheraufbau in Bild 51 ähnliche "C"-Struktur erzeugt.

Die IDL-Spezifikation wird für die Möglichkeit der Erzeugung eigener Instanzvariablen um zwei Bereichsdeklarationen *'public:'* und *'private:'* erweitert. Für IDL-definierte Va-

riablen mit diesen Deklarationen werden automatisch *get_attribute* und *set_attribute* Methoden erzeugt. Private Variablen eines Objekts werden ebenfalls mit instantiiert, jedoch nicht exportiert. Ein FIMO-Objekt kann damit kreiert werden durch einen Aufruf seiner *new*-Methode bzw. gelöscht werden durch die komplementäre *delete*-Methode. Variablen, die lokal zum Objekt gehören, also die sogenannten CORBA-Attribute, werden automatisch in die neue Instanz aufgenommen. Type-cast Operationen werden dynamisch typgeprüft in den zu dem Objekt gehörenden Methoden *widen* bzw. *narrow.* Ein vereinfachtes Beispiel, das die Anwendungserstellung auf Basis des FIMO-IDL-Compilers zeigt, wird im folgenden Kapitel besprochen.

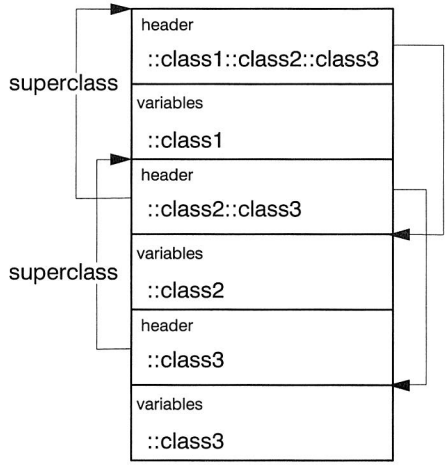


Bild 51: *Speicheraufbau der Objektinstanzen des FIMO-Objektmodells*

6.5 FIMO-Implementierung für das Feldbussystem PROFIBUS-FMS

6.5.1 Charakteristik des PROFIBUS-FMS-Protokolls

Der Austausch von Informationen zwischen verschiedenen Feldgeräten basiert bei PROFIBUS-FMS auf *Kommunikationsobjekten*. Diese besitzen Attribute wie zum Beispiel den Datentyp, Zugriffsrechte oder einen Zugriffsschlüssel. Zu jeder Objektart werden entsprechende Operationen festgelegt, die auf diese angewandt werden können [9]. Die Kommunikationsobjekte werden durch einen Eintrag im *Objektverzeichnis* (OD, Object Dictionary) dem Kommunikationssystem bekannt gegeben. Unterschieden werden statische und dynamische Objektarten. Erstere werden projektiert und können während der Betriebsphase nicht mehr geändert werden. Letztere können im laufenden Betrieb durch einen Eintrag im Objektverzeichnis definiert und auch wieder gelöscht werden.

Das Objektverzeichnis kann bei PROFIBUS-FMS aus bis zu sechs Unterverzeichnissen bestehen. Jedes Unterverzeichnis enthält Objektbeschreibungen ganz bestimmter Kommunikationsobjekte (vgl. [9]). Jeder Eintrag besitzt als eindeutige Adresse einen Index und eventuell zusätzlich einen Namen über den er referenziert werden kann. In Tabelle 3 ist der allgemeine Aufbau eines Objektverzeichnisses dargestellt.

Index	Inhalt	*Beschreibung*
0	allgemeine Informationen	*Struktur- und Verwaltungsinformationen des Objektverzeichnisses*
1... 14	Standarddatentypen: Bool Integer (8-,16-,32-Bit) Unsigned (8-,16-,32-Bit) Floating-Point ...	*statisches Typverzeichnis*
k... m	spezielle Datentypen und Strukturvariablen	
p... r	Kommunikationsobjekte	*statisches Objektverzeichnis*
t... v	Variablenlisten	*dynamisches Objektverzeichnis*

Tab. 3: *Allgemeiner Aufbau des FMS-Objektverzeichnisses (nach [97])*

Das statische Typverzeichnis enthält zum einen die in der FMS-Norm festgelegten Standarddatentypen. Zusätzlich können in einem weiteren Abschnitt selbstdefinierte Datentypen eingetragen werden. Dies können einfache Variablen sein oder Arrays, deren Elemente aus den Standarddatentypen aufgebaut sind, oder Strukturvariablen, deren Elemente aus dem statischen Typverzeichnis stammen müssen. Das dynamische Objektverzeichnis beinhaltet Variablenlisten. Diese stellen eine Gruppierung von im statischen Objektverzeichnis definierten Typen dar und können mit einem eigenen Index und Namen versehen werden. Die in der Tabelle grau hinterlegten Bereiche können von der Anwendung modifiziert werden.

Der eigentliche Datenaustausch findet bei PROFIBUS-FMS zwischen einem Client und einem Server statt. Letzterer wird durch ein *Virtuelles Feldgerät* (VFD) beschrieben. Jedes Feldgerät kann eines oder mehrere VFDs implementieren, die dann wie einzelne Geräte angesprochen werden können. Ein VFD kommuniziert mit anderen Feldgeräten über *projektierte Kommunikationskanäle*, die durch *Kommunikationsreferenzen* (CR, Communication Reference) repräsentiert werden. Die Objekte, die über einen Kommunikationskanal ausgetauscht werden sollen, sind im Objektverzeichnis hinterlegt. Bild 52 gibt einen Überblick über den Zusammenhang zwischen VFD, OD und den Kommunikationsreferenzen.

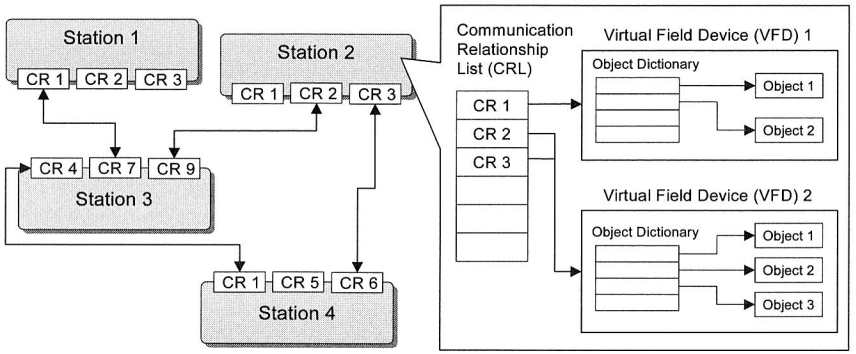


Bild 52: *Zuordnung von CRL, OD und VFD in einem PROFIBUS-FMS Netz*

6.5.2 Realisierung der FIMO-Kommunikation

Die Datenübertragung im Rahmen der FIMO-Kommunikation kann nun auf zwei Arten erfolgen:

- Nutzung der *Variablendienste*: Diese erlauben es einem Feldgerät die zu einem VFD gehörenden Variablen eines anderen Feldgeräts zu lesen. Die Dienstausführung erfolgt bestätigt.
- Nutzung des Dienstes *Information Reports*: Bei diesem Dienst wird der Inhalt einer Variablen geschickt. Der Empfänger muß dann den im Datenpaket erhaltenen Wert durch sein Wissen über den OD des Senders wieder dekodieren. Die Dienstausführung erfolgt unbestätigt und kann im Multi- oder Broadcast-Verfahren verwendet werden.

Da der RPC-Mechanismus der Feldbusschnittstelle im wesentlichen aus dem Verschicken und Empfangen von Datenpaketen, nämlich der Aufruf- und Rückgabeparameter, besteht, ist der Dienst Information Report für die Implementierung der Kommunikation besser geeignet. Zudem ist zu beachten, daß für die Datenübertragung pro Nachricht nur ca. 240 Byte an Nutzdaten zur Verfügung stehen und bei größeren Nachrichten somit eine Segmentierung stattfinden muß. Ein unbestätigter Dienst ist hier auch aus Effizienzgründen vorzuziehen. Der Dienst Information Report wird in der Punkt-zu-Punkt Verbindungsart genutzt, d.h. es wird auf die vorprojektierten Kommunikationskanäle eines Feldgeräts zurückgegriffen. Dieser Ansatz ist zwar weniger flexibel, er deckt sich jedoch mit der üblicherweise statischen Konfiguration eines Feldbussystems. Insbesondere unterscheiden sich FIMO-Feldgeräte in der Projektierungsphase dann auch nicht von anderen Feldgeräten.

6.5.3 Marshaling und Unmarshaling der FIMO-Datentypen

In den statischen Stubs und Skeletons, die der IDL-Compiler aus der Schnittstellenbeschreibung generiert, ist für jeden Methodenaufruf das entsprechende Marshaling und Unmarshaling der Parameter hinterlegt. Intern wird die zu versendende Nachricht zuerst mit den dazugehörigen Parametern in einer verketteten Liste aufgebaut. Diese ist hierarchisch organisiert, d.h. an komplexe Parameter schließen sich rekursiv die Unterparameter an. Beim Versenden über das Feldbussystem wird die hierarchische Strukur, wie in Bild 53 angedeutet, durch das Einfügen spezieller Trennzeichen in eine flache Struktur überführt.

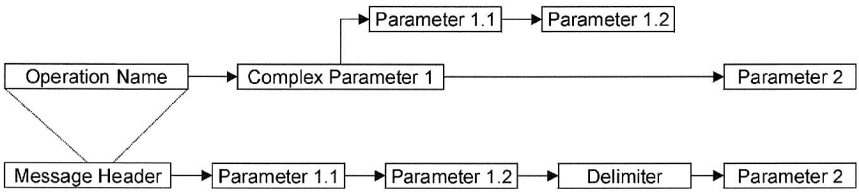


Bild 53: *Aufbau einer FIMO-Nachricht im Speicher bzw. beim Verschicken*

PROFIBUS-FMS bietet standardmäßig eine Reihe von Datentypen an (vgl. dazu das statische Typverzeichnis), die elegant für das Versenden der meisten FIMO-Datentypen eingesetzt werden können. Dies erspart das Implementieren eigener Funktionen für das Marshaling bzw. Unmarshaling.

Bei der Initialisierung der Feldbusschnittstelle müssen zuerst die für die Übertragung benötigten Basisdatentypen im Objektverzeichnis angelegt werden. Dazu wird eine spezielle Strukturvariable als "Schablone" verwendet, deren grundlegender Aufbau für alle Basisdatentypen identisch ist, die im Strukturelement "Datentyp" jedoch den konkreten Datentyp aufnimmt (siehe Tabelle 4). Ein spezieller Eintrag im Objektverzeichnis ist für den Nachrichtenkopf und die Objektreferenz notwendig. Das Versenden der Strukturvariablen geschieht in einzelnen Information Reports[6], der Nachrichtenkopf muß aufgrund seiner Größe auf mehrere Einzelnachrichten aufgeteilt werden.

6. Anm.: Variablenlisten können an dieser Stelle nicht verwendet werden.

Strukturelement	*Beschreibung*
Request_Id	*dient der Identifizierung und Zuordnung von Requests/Replies*
Sequence	*Numerierung der zu einer Nachricht gehörenden Teilnachrichten*
msg_type	*Kennung für den Datentyp, um diesen auf der Empfangsseite auch dynamisch auslesen zu können*
<Datentyp>	*der eigentliche Wert des Datentyps*

Tab. 4: *Aufbau der für die Übertragung verwendeten Strukturvariable*

6.5.4 Erstellung von Anwendungen

Der prinzipielle Prozeß zur Erstellung einer FIMO-Anwendung ist in Bild 54 dargestellt. Den Ausgangspunkt bildet die Definition der Objekte und der zugehörenden Schnittstellen durch die Schnittstellenbeschreibungssprache IDL. Dabei werden die Objekttypen festgelegt, die Objektattribute, die Methoden und ihre Parameter. Wie bereits angesprochen, stellt die FIMO-IDL aufgrund einiger nicht implementierten Datentypen

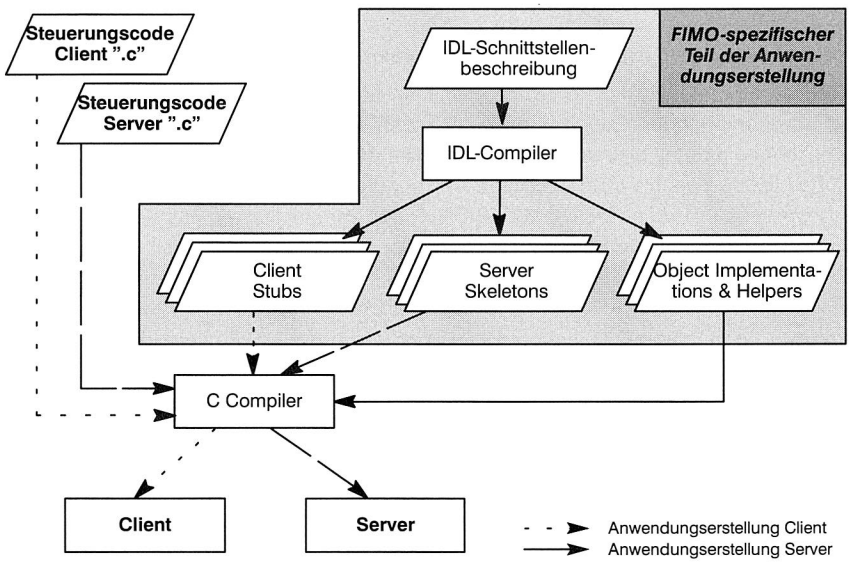


Bild 54: *Vorgang der Anwendungserstellung*

lediglich eine Teilmenge der CORBA-IDL dar. Der Compiler generiert daraus die Stubs und Skeletons, die den Marshaling und Unmarshaling Code enthalten und zudem zahlreiche Hilfsfunktionen, u.a. für die Objekterzeugung und -vernichtung (vgl. Kapitel

6.4.7 für Details). Der Steuerungscode für den Client bzw. den Server wird dann als normaler "C"-Code zu den Stubs bzw. Skeletons sowie den Hilfsfunktionen gelinkt.

6.6 Zusammenfassung und Bewertung

FIMO stellt eine an die speziellen Bedürfnisse der Prozeßebene und der Feldbussysteme angepaßte CORBA-Architektur für den Einsatz in Feldgeräten dar. Damit deckt FIMO nur Anforderungen der Implementierungsebene ab. Als wesentlichen Vorteil bietet es ein durchgängiges CORBA-basiertes Objektmodell, das eine schnelle Anwendungserstellung erlaubt. Dafür sind jedoch größere Änderungen in den Feldgeräten selbst vorzunehmen, die sich bis auf eine eigene Verwaltung der eintreffenden Feldbusnachrichten erstrecken. Für einfachere Feldgeräte scheidet diese Lösung aus Gründen der zu knappen Ressourcen und verfügbaren Rechenleistung aus. Ebenso sind diejenigen Bussysteme für eine FIMO-Implementierung ungeeignet, die nicht über asynchrone Datenübertragungsdienste verfügen, da sich die Nutzung synchroner Dienste sicher auf die normale Prozeßdatenkommunikation auswirken würde.

Im folgenden wird eine detaillierte Bewertung hinsichtlich der in Kapitel 2 formulierten Anforderungen vorgenommen.

Anforderung K1: Unterstützung der vertikalen Kommunikation 

Über die FIMO-ORB-Bridge wird die vertikale Kommunikation der Prozeßebene mit der darüberliegenden Ebene unterstützt. Dabei ist die Integration nicht auf CORBA-Objekte beschränkt, vielmehr können mit Hilfe der in Kapitel 5 beschriebenen COM/CORBA-Bridge auch CORBA-basierte Anwendungen wie native COM-Anwendungen angesprochen werden. Die COM/CORBA-Bridge wird in diesem Fall der FIMO-ORB-Bridge am Übergang von der Prozeßebene zur Zellenebene nachgeschaltet.

Anforderung K2: Unterstützung der horizontalen Kommunikation 

Aufrufe an den FNS *(list/resolve)*, die nicht innerhalb des Feldbussystems selbst auflösbar sind, werden automatisch über die FIMO-ORB-Bridge an die darüberliegende Ebene weitergeleitet. Der dort verfügbare, eventuell kommerzielle ORB wird nun seinerseits versuchen, den ihm übergebenen Objektnamen aufzulösen. Dazu benutzt er seinen eigenen Namensdienst. Implementiert nun beispielsweise das Feldbussystem einer benachbarten Fertigungszelle ebenfalls FIMO, so sind die dort definierten Objekte an der zugehörigen FIMO-ORB-Bridge registriert und können damit von jedem CORBA-konformen Namensdienst lokalisiert werden. Die horizontale Kommunikation über verschiedene Feldbussysteme hinweg wird somit unterstützt.

Anforderung K3: Unterstützung für ein verteiltes Client/Server-Modell 

Das Client/Server-Modell von FIMO basiert auf dem in CORBA spezifizierten verteilten Objektmodell.

7 Konzeption eines Middleware-Framework zur Integration der Prozeßebene

Die in den vorherigen Kapiteln konzipierten und realisierten Erweiterungen zur Integration der Prozeßebene waren für sich genommen nicht in der Lage, alle gestellten Anforderungen befriedigend abzudecken. Dies kann prinzipiell nur im Zuge einer umfassenderen Gesamtlösung durch ein Middleware-Framework gelingen, das *per se* die Implementierungsebene und die Ebene der Anwendungslogik abzudecken vermag. Dabei ist die Konzeption und vollständige Spezifikation eines Middleware-Frameworks eine derart komplexe Aufgabenstellung, daß im Rahmen dieser Arbeit lediglich die wichtigsten Teile davon herausgegriffen und detaillierter vorgestellt werden können. Für die folgenden Ausführungen sind Vorkenntnisse im Bereich der Metainformationssysteme hilfreich, um insbesondere die grundlegenden Begriffe "Metadaten" und "Metasprache" zu verstehen, die hier nicht explizit eingeführt werden können. Ein guter erster Überblick über diese Thematik wird in *Ortner [98]* gegeben.

7.1 Vorbetrachtungen

Sieht man die in den Kapiteln vier bis sechs vorgeschlagenen Konzepte nicht allein aus dem Blickwinkel der Anforderungen sondern aus einer allgemeineren Sicht, so ergibt sich das folgende Bild: Allen Konzepten ist eine *spezifische Middleware-Lösung* gemein. Dies bedeutet, daß eine ganz bestimmte Systemumgebung vorausgesetzt wird. Bei den universellen Geräteprofilen ist es der Einsatz von ActiveX-Komponenten, die technologisch auf DCOM basieren und so ein bestimmtes Betriebssystem bzw. einen bestimmten Web-Browser erfordern und damit auch indirekt eine bestimmte Hardwareplattform, auf der die Software lauffähig ist. Beim Jini-Basiskonzept ist zwar prinzipiell das Kommunikationsprotokoll zwischen Client und Server unspezifiziert, um jedoch nicht für jede Aufgabe eine eigene aufwendige Protokollentwicklung durchführen zu müssen, ist es sinnvoller, eine Middleware von vornherein festzulegen. Dieser Aspekt wurde beim Jini-basierten Framework mit der Wahl von CORBA auch umgesetzt. Der in der Jini-Spezifikation auf der Grundlage von Java und Java-RMI definierte Lookup-Service, konnte hingegen Web-Server-basiert und damit unabhängig von Java konzipiert und implementiert werden. Bei FIMO jedoch, der einen weitgehend CORBA-konformen ORB darstellt, sind die potentiellen Clients in erster Linie CORBA-Clients.

Die Wahl einer geeigneten Middleware ist gegenwärtig zwischen CORBA und DCOM zu treffen, da Java-RMI und die damit verbundenen Festlegung auf die Programmiersprache Java im Umfeld der Fertigungsautomatisierung derzeit inakzeptabel ist. Theoretisch hat die Entscheidung für eine der beiden Middleware-Implementierungen auch keine Auswirkungen auf potentielle Clients, wenn man an den Einsatz einer DCOM/CORBA-Bridge denkt. Diese vermittelt im Prinzip[7] weitgehend transparent zwischen den beiden Technologien.

Alle bisherigen Konzepte boten zudem *kein einheitliches Daten- und Schnittstellenmodell*. Im Rahmen der universellen Geräteprofile wurde ausgehend von der Automatisierungsaufgabe versucht, die notwendigen Daten und Schnittstellen zu identifizieren, die eine Anwendung in der Leitebene benötigen könnte. Die Integration erfolgte durch ActiveX-Komponenten verbunden mit der Festlegung auf die Middleware COM. Bei Jini kann der Client zwar leicht Funktionen des Feldgeräts aufrufen und nutzen, um jedoch die Methoden selbst oder deren Rückgabedaten in eine bestehende Client-Anwendung zu integrieren, müssen die Service-Objekte in Form von plattformspezifischer Komponentensoftware implementiert werden. Das Java-Komponentenmodell scheidet wiederum in der Praxis aus, da die bestehende Software in der Zellen- und Leitebene nicht in Java geschrieben ist.

Bei der Definition eines Middleware-Frameworks gilt es somit neben den in Kapitel 3 beschriebenen Anforderungen darauf zu achten,

- möglichst keine spezifische Middleware festzuschreiben und
- eine neutrale Beschreibung der Daten und Schnittstellen zu erreichen.

Bei den folgenden Betrachtungen wird von einem Szenario ausgegangen, bei dem das Feldgerät als Server agiert und sich der Client außerhalb der Prozeßebene befindet. Das Middleware-Framework wird dabei beim Client zur Anwendung kommen, während das Feldgerät aufgrund der eingeschränkten Ressourcen lediglich eine Middleware-Schnittstelle zur Verfügung hat. Dieser Ansatz ist deshalb sinnvoll, weil eine in der Prozeßebene laufende Anwendung nicht über derartig flexible und komplexe Mechanismen verfügen muß, wie sie das Framework anbietet. Bei der horizontalen Kommunikation gemäß der Anforderung K2 aus Kapitel 3.2 gilt dies in gleicher Weise – sie kann vollständig über die Middleware abgewickelt werden. Insofern ist das Ausgangsszenario keine Einschränkung, sondern der allgemeinere Fall, da es eine Betrachtung des kompletten Middleware-Frameworks erfordert.

7.2 Grundkonzeption des Frameworks

7.2.1 Implementierungsebene

Um in einer verteilten Kommunikationsarchitektur eine Client/Server-Kommunikation aufzusetzen, müssen zumindest folgende Mechanismen und Dienste verfügbar sein:

- Es muß einen Mechanismus geben, mit dem ein Client die Server suchen kann. Dazu muß der Client entweder die Schnittstellen des Servers bereits kennen bzw. zur Laufzeit erfragen können.

7. In der Praxis ergeben sich jedoch oft Schwierigkeiten. Beginnen z.B. in einer DCOM-Anwendung die Indizes eines Arrays nicht mit 0 sondern mit 1, so ist die Bridge von *VisualEdge* nicht in der Lage dies zu erkennen. Auf weitere Probleme im Zusammenhang mit Java kann hier nicht genauer eingegangen werden.

- Es muß eine Middleware-Schicht existieren, die die Anfragen des Client an den Server und in umgekehrter Richtung die Rückantwort übermittelt.

Nachfolgend werden die Grundkonzepte für die einzelnen Punkte entwickelt.

Suchen eines Servers

Um einen Überblick über die verschiedenen Methoden zu bekommen, die bei der Lokalisierung eines Servers eingesetzt werden können, empfiehlt sich ein Blick auf CORBA. Dort werden verschiedene Mechanismen angeboten, mit deren Hilfe Clients einen Server finden können. Im allgemeinsten Fall besitzt der Client keinerlei Wissen über den Namen und den Ort des Servers, sondern lediglich eine Vorstellung, welche Schnittstellen dieser anbieten muß. Mittels des *CORBA Trading Service* [99], der dem Discovery/Lookup-Service von Jini ähnlich ist, fragen Clients dann einen bestimmten *Service Type* nach. Dieser wird beschrieben durch ein IDL-Interface und eine Menge von Attributen. Falls der *Name* eines Servers bereits bekannt ist, kann dieser auch über den *CORBA Naming Service* lokalisiert werden. Eine weitere Alternative besteht darin, an einer bekannten Stelle in einem Netzwerk eine *Objektreferenz* in Form einer Zeichenkette *(Object Reference String)* zu hinterlegen, die der Client zum Zugriff auf den Server benutzen kann. Darüber hinaus existieren noch weitere spezielle Mechanismen, um Server zu lokalisieren, die in diesem Zusammenhang jedoch keine Rolle spielen.

Je mehr Vorwissen man also über den Server zum Zeitpunkt der Suche besitzt, desto einfacher lassen sich die Suchmechanismen konzipieren. Im Rahmen dieser Arbeit kann man leicht folgende Einschränkungen treffen: Da die Feldgeräte über ein Feldbussystem vernetzt sind und dieses nur über ein Gateway angesprochen werden kann, bietet es sich an, die Information welche Feldgeräte-Server verfügbar sind, am Gateway zu hinterlegen[8]. Um einen *konkreten* Server auszuwählen, kann die Beschreibung der Daten und Schnittstellen, die im Rahmen des Frameworks sowieso erfolgen muß, herangezogen werden. Damit bleibt nur noch die Frage des geeigneten Protokolls offen, um sich mit dem Gateway zu verbinden, die Liste der Server zu durchsuchen und den gewünschten auszuwählen. Dieser Vorgang muß insbesondere automatisiert möglich sein.

Vermittlung eines Methodenaufrufs

Geht man davon aus, daß das Feldgerät den Server darstellt, so kann bei der Client/Server-Kommunikation die Verbindung außerhalb der Prozeßebene unterschieden werden von der Verbindung zwischen Gateway und Feldgerät. Um keine spezifische

8. Der Einfachheit halber wird von "Gateway" gesprochen, wenn der Rechner bezeichnet werden soll, der den Zugriff auf das Feldbussystem realisiert. Es wird auch stets von einer 3-Schichten-Architektur bestehend aus Feldgeräte-Server, Gateway und Client ausgegangen, was keine prinzipielle Einschränkung darstellt.

Middleware-Lösung für einen der Bereiche vorzuschreiben, gibt es derzeit nur eine Lösung: die Verwendung eines einfachen und neutralen Punkt-zu-Punkt Verbindungsprotokolls zum Austausch strukturierter und typisierter Informationen. Die diese Informationen verarbeitenden Stellen können selbst wieder auf speziellen Middleware-Implementierungen aufsetzen. Das *Simple Object Access Protokoll* (SOAP, [52]) stellt eine solche Lösung dar, dessen Unterstützung innerhalb des Frameworks gefordert wird.

Ein Anwendungsszenario könnte dann wie folgt aussehen: Der Client bekommt auf Ebene der Anwendungslogik die Daten und Schnittstellen der Server mitgeteilt. Dabei wird auch die Information über die verfügbaren Kommunikationswege, insbesondere der unterstützten Middleware-Implementierungen des Gateways mitgeliefert. Je nach Plattform und Verfügbarkeit der Middleware beim Client und beim Server kann dann der Aufbau der Kommunikationsverbindung zum Gateway erfolgen. Mit der in jedem Fall geforderten Unterstützung für SOAP, kann ein Client außerhalb der Prozeßebene stets eine SOAP-Verbindung zum Gateway aufbauen. In Bild 55 ist dieses Grundprinzip dargestellt.

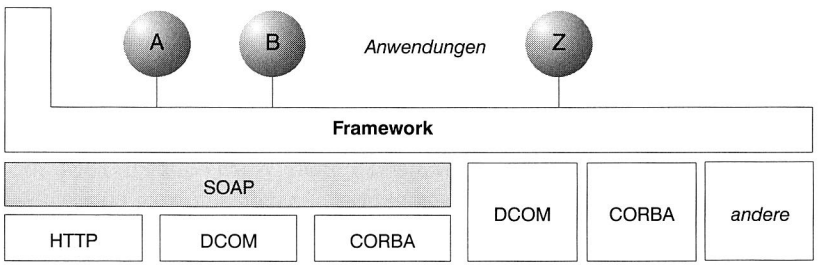


Bild 55: *Die Middleware-Struktur des Frameworks mit der geforderten Unterstützung für SOAP*

SOAP stellt ein einfaches, XML-basiertes Protokoll dar, das aus drei Teilen besteht (vgl. auch Kapitel 2.5.2 "BizTalk"):

- Der *SOAP-Envelope* beschreibt, welche Daten die Nachricht beinhaltet und wie diese verarbeitet werden müssen.
- Der *SOAP-Header* enthält u.a. die Kodierregeln für den Austausch anwendungsspezifischer Datentypen.
- Der *SOAP-Body* definiert ein Format, um RPC-ähnliche Aufrufe und Antworten abbilden zu können.

SOAP kann prinzipiell mit jeder Middleware-Plattform zusammenarbeiten. In der "W3C-Note" [52] ist festgelegt, wie SOAP-Nachrichten über HTTP verschickt werden

können. Für die weitere Ausführung spielt der konkrete Aufbau einer SOAP-Nachricht eine untergeordnete Rolle. Zum besseren Verständnis wie die Abbildung eines RPCs auf SOAP realisiert wird, dient das nachfolgende einfache Beispiel:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 521

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
    <m:SetPosition
      xmlns:m="http://www.faps.de/soap-encoding/Robot">
      <m:XPosition>
        32342
      </m:XPosition>
      <m:YPosition>
        45600
      </m:YPosition>
    </m:SetPosition>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Der URI des Servers wird implizit über das Ziel des HTTP-Request spezifiziert. Nach dem HTTP-üblichen Header folgt die eigentliche SOAP-Nachricht. Der Methodenaufruf wird auf einen Strukturtyp abgebildet, dessen Name den Methodenname darstellt *(SetPosition)*. Durch die Angabe einer URI kann auf ein Schema für die Kodierung verwiesen werden, so daß die Typen der nachfolgend übergebenen Parameter nicht explizit in der Nachricht gesetzt werden müssen. Als Parameter werden im Beispiel zwei Integer-Werte übergeben für die X-Position und die Y-Position. Für die Kodierung der Datentypen dient bei SOAP die Spezifikation der *XML-Datentypen* [100] als Basis.

7.2.2 Ebene der Anwendungslogik

Modellierung der Daten und Schnittstellen

Ähnlich wie bei MMS, soll ein objektorientiertes Modell bei der Abbildung eines Feldgeräts zugrunde gelegt werden. Die Rolle des Objektmodells wird erst in Kapitel 7.4 ausführlicher behandelt. An dieser Stelle soll nur ein grober Überblick gegeben werden, wie die Daten und wie die Funktionen eines Feldgeräts abgebildet werden.

Im Rahmen eines Objektmodells ist zunächst die Klärung des Begriffs "Datum" notwendig. Damit sind typisierte Objekte gemeint, wie sie aus einer objektorientierten

Programmiersprache bekannt sind[9]. Zugleich können sie – ähnlich wie bei MMS – Kommunikationsobjekte darstellen, d.h. diese Objekte können über eine Kommunikationsverbindung ausgetauscht werden. Bei den zu modellierenden Schnittstellen handelt es sich damit um die Objektmethoden.

Die Vorteile einer objektorientierten Modellierung sollten hinreichend bekannt sein, so daß sie hier nicht mehr aufgezählt werden müssen. Auf einen wesentlichen Punkt sei dennoch hingewiesen. Mit der Einführung verschiedener anwendungsbezogener Klassen, beispielsweise für die Gerätediagnose oder die Gerätekonfiguration, ergibt sich automatisch eine Strukturierung der Methoden über ihre Klassenzugehörigkeit. Dies ist bei einer funktional orientierten Modellierung des Feldgeräts, wie sie im Rahmen der EDD erfolgt, nicht möglich. Dort werden die unterschiedlichen ausführbaren Gerätefunktionen im Rahmen der "COMMAND"-Struktur gleichermaßen "nebeneinander" abgebildet.

Als Beispiel für ein Datum soll eine einfache Prozeßvariable dienen, die in einem Objektmodell den in Tabelle 5 gezeigten Aufbau aufweisen kann. Wie bei einem objektorientierten System üblich, besitzt eine Klasse Attribute und Methoden und kann von anderen Klassen abgeleitet werden (Vererbung, hier nicht dargestellt).

Klasse	*ProcessVariable*
Attribute	*Value* *Unit* *Representation*
Methoden	*getValue ( )* *setValue ( )*

Tab. 5: *Aufbau einer Klasse "Prozeßvariable"*

Um auf Ebene der Anwendungslogik die geforderte neutrale Repräsentation der Daten zu erreichen, ist es notwendig, ein von einer konkreten Systemumgebung unabhängiges Datenformat als Ausgangspunkt zu nehmen. Darüber hinaus ist es sinnvoll, nicht einfach nur eine Sprache zu verwenden, in der ein Datum auf eine fest vorgegebene Weise *abgebildet* werden kann, sondern eine Sprache, in der das komplette *Datenmodell* formuliert werden kann – also eine Modellierungssprache oder Metasprache. Daraus ergeben sich eine Reihe von Vorteilen:

- Die Daten sind selbstbeschreibend, d.h. sie tragen die notwendigen Informationen, die für ihre Interpretation benötigt werden in sich (Metadaten).
- Die Daten können problemlos von einer Anwendung an die nächste weitergereicht werden, ohne daß weitere Kontextinformationen notwendig sind.

9. Die Begriffe Datum und Objekt werden weitgehend synonym gebraucht. Es wird der Begriff "Datum" jedoch vorgezogen, wenn im Kontext die damit intuitiv verbundene Vorstellung eines "Prozeßwertes" ausreichend oder anschaulicher erscheint. Entsprechend ist mit Datenmodell eigentlich das gesamte Objektmodell gemeint.

- Anwendungen, die diese Daten nutzen, brauchen lediglich *eine* unveränderliche Schnittstelle implementieren und interpretieren die Information zur Laufzeit.
- Neue Datenformate und komplexe Datentypen können auf einfache Weise in das Konzept integriert werden, da nur ein entsprechend angepaßtes Modell mit übertragen werden muß.

Ein Anwendungsszenario

In Bild 56 ist in einer Prinzipdarstellung gezeigt, wie das Zusammenspiel zwischen einem Client außerhalb der Prozeßebene und einem Feldgeräte-Server abläuft.

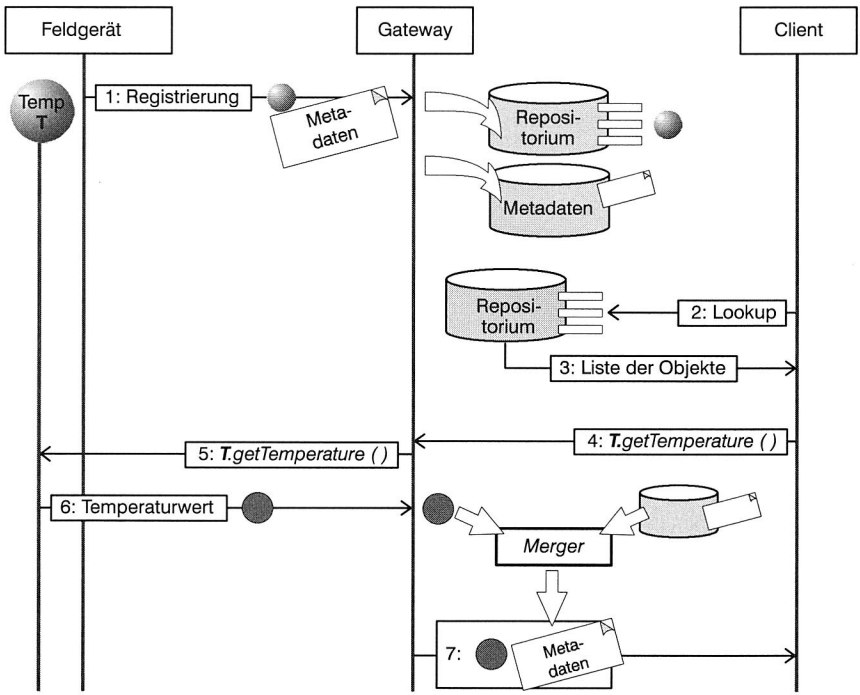


Bild 56: *Prinzipdarstellung – Übertragung der Metadaten*

In einem ersten Schritt erfolgt die Registrierung der Objekte mit den Metadaten am Gateway. Dabei kann unterschieden werden zwischen der Registrierung der Metadaten eines Objekts und der Registrierung der bloßen Objektmethoden in einem speziellen Repositorium. Ein Client verbindet sich über ein festgelegtes Protokoll mit dem Gateway und durchsucht zunächst das Repositorium nach einem bestimmten Objekt. Mit der Objektbeschreibung erhält er zugleich die Methodensignatur, also den Metho-

dennamen, die Parameter und die Datentypen der Parameter. Um zur Laufzeit über die Middleware einen Methodenaufruf an das gewünschte Objekt absetzten zu können, muß der Client eine Art Referenz auf das Objekt besitzen. Diese kann entweder statisch zusammen mit der Objektbeschreibung versendet werden, oder der Client erlangt diese über eine dynamische Namensauflösung. In jedem Fall wird der Methodenaufruf über das Gateway an das betreffende Feldgerät weitervermittelt.

Aus Effizienzgründen ist es nicht sinnvoll, bei der Abfrage des aktuellen Wertes einer Prozeßvariablen, zwischen Feldgerät und Gateway jedesmal auch die Metadaten auszutauschen. Daher erfolgt hier nur eine Übertragung der Produktivdaten, d.h. des "Wertes" des Datenobjekts. Am Gateway werden die Produktivdaten mit den Metadaten dann im sogenannten *Merger* zusammengeführt und dem Aufrufer als Antwort zurückgeliefert.

Gegenüber einem herkömmlichen Methodenaufruf ergibt sich somit folgender wichtiger Unterschied: Die Rückgabedaten stellen nicht einfach Zahlenwerte dar, die mittels eines vorher bekannt gegebenen Typs interpretiert werden können, sondern sind zur Laufzeit interpretierbare Objekte, die *zusammen* mit ihrem Typ, den Modellierungsinformationen und (wahlweise auch) den Methoden übertragen werden.

Während in der Darstellung explizit von einem Methodenaufruf des Client ausgegangen wird, kann auch eine spontane Datenübertragung seitens des Feldgeräte-Servers im Zuge eines Events angestoßen werden. Das Gateway vermittelt die Daten dann an diejenigen Clients weiter, die zuvor genau diesen Event abonniert haben.

Konzeption des Mergers

Das Grundkonzept des Mergers ist prinzipiell auf jede Middleware anwendbar und kann auch im Rahmen von OPC eingesetzt werden. Dazu muß der Merger lediglich allgemein genug gehalten werden. In Bild 57 ist die Funktionsweise dargestellt:

- Die Produktivdaten werden über die Middleware oder eine allgemeine Kommunikationsschnittstelle geliefert.
- Anhand des physikalischen Orts, von dem die Daten stammen und zusätzlicher Informationen, die über die Middleware verfügbar sind, wie den Name des Datums, führt der Merger eine Suche in seiner Metadaten-Datenbank durch und holt die zugehörigen Metadaten.
- Der Merger führt das Datum mit seinen Metadaten zusammen, so daß sie anschließend gemeinsam übertragen werden können.

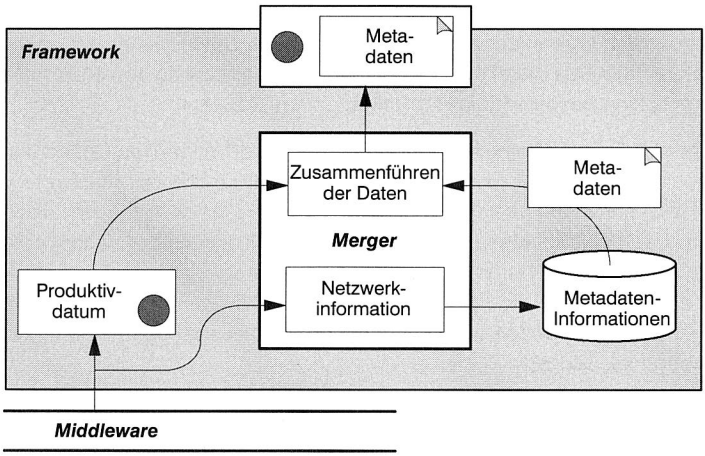


Bild 57: *Funktionsweise des "Mergers"*

7.2.3 Zusammenfassung

In diesem Kapitel wurde lediglich das Grundkonzept des Middleware-Frameworks vorgestellt. Vor einer detaillierteren Ausarbeitung müssen noch folgende Punkte geklärt werden:

1. Das Framework muß einen Mechanismus bereitstellen, mit dem ein Client einen Server suchen kann.
2. Es muß auf Ebene der Middleware festgelegt werden, wie ein Methodenaufruf an ein bestimmtes Feldgerät vermittelt wird und wie ein Feldgerät selbst einen Methodenaufruf absetzen kann.
3. Es wird eine Modellierungssprache benötigt, in der die Feldgeräte und die Objekte inklusive der Objektmethoden modelliert werden.
4. Es wird ein Format benötigt, in dem die Metadaten am Gateway hinterlegt werden.
5. Es wird ein Format benötigt, in dem die Objekte einem Client verfügbar gemacht werden.

7.3 Das Resource Description Framework

7.3.1 Einführung

Die über das World Wide Web am meisten verbreiteten Inhalte stellen HTML-Dokumente dar. Diese sind sowohl vom Menschen als auch maschinell lesbar aber nicht für

die maschinelle *Weiterverarbeitung* geeignet. Dies liegt daran, daß die in den Dokumenten gespeicherten Informationen nicht semantisch aufbereitet vorliegen. Als Lösung dieses Problems bietet sich die zusätzliche Speicherung von *Metadaten* an, die eine semantische Beschreibung der Dokumente mitliefert.

Das *Resource Description Framework* (RDF) des World Wide Web Consortium (W3C) stellt hierzu eine mögliche Methode dar, die speziell für die automatisierte Verarbeitung von Web-Inhalten konzipiert wurde. In der *RDF Model and Syntax Specification* [101] wird das RDF-Metadatenmodell spezifiziert. Dieses stellt zunächst eine anwendungsneutrale Methode bereit, um beliebige Ressourcen beschreiben zu können. Auf Basis des RDF-Modells und der *RDF Schema Specification* [102] wird dann in einem *RDF-Schema* für ein spezielles Anwendungsgebiet das zugehörige Metadatenmodell beschrieben (vgl. Bild 58).

Für die syntaktische Repräsentation von RDF wird im Zuge der W3C-Spezifikation XML verwendet. Damit schließt sich der Kreis, da nun das Metadatenmodell in XML-Dokumenten hinterlegt werden kann. Die RDF "Model and Syntax Specification" besitzt derzeit den Status einer W3C Empfehlung, die RDF "Schema Specification" ist hierfür vorgeschlagen.

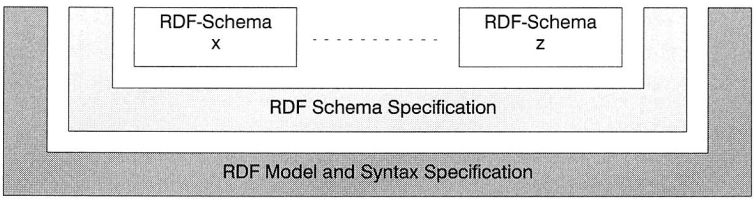


Bild 58: *Einordnung des RDF-Modells und der RDF Schema Spezifikation*

7.3.2 Das RDF Basismodell

Das RDF-Modell repräsentiert im wesentlichen ein klassisches Entity-Relationship Diagramm. Im Basismodell unterscheidet RDF drei Objekttypen:

- *Resources:* Alle in einem RDF-Modell beschriebenen "Dinge" werden als *Resources*[10] bezeichnet. Resources werden über ihren URI eindeutig bezeichnet.
- *Properties:* Properties beschreiben eine Resource, indem sie beispielsweise einen Aspekt, ein Attribut oder eine Relation darstellen. Die Bedeutung einzelner Properties, ihre Zuordnung zu Resource-Typen, die möglichen Werte und Beziehungen zu anderen Properties werden in einem RDF-Schema festgelegt.

10. Bei der Beschreibung des RDF-Frameworks sollen in konsistenter Weise die englischen Bezeichnungen verwendet werden. Daher auch die englische Schreibweise für "Resource(s)".

Properties besitzen einen Namen und einen Wert, wobei der Wert entweder ein Literal (bei Abbildung des RDF-Modells auf XML i.d.R. ein primitiver XML Datentyp) oder selbst wieder eine Resource darstellen kann.

- *Statements:* Eine spezifische Resource zusammen mit einem Property und dessen Wert ("Property Value") bildet ein RDF-Statement.

Folgendes Beispiel soll die Verwendung von RDF verdeutlichen:

Siemens ist der Hersteller der Steuerung S7.

Das RDF-Statement besteht dabei aus drei Teilen: "Steuerung 7" ist die Resource, "Hersteller" das Property und "Siemens" der Wert des Property und in diesem Beispiel ein Literal. Daraus resultiert die in Bild 59 gezeigte graphische Repräsentation.

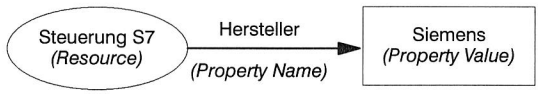


Bild 59: *Graphische Repräsentation eines einfachen RDF-Statements*

Für weiterführende Betrachtungen ist eine formale Notation des RDF-Basismodells vorzuziehen:

Definition 6: *RDF-Statements*
Ausgehend von einer Menge Resources (R), einer Menge an Literalen (L) und einer Menge von Properties (P), die eine Teilmenge von R darstellt, kann man Statements s definieren als ein Tripel (p, r, o), wobei $p \in P$, $r \in R$ und $o \in R \cup L$.

Wie bereits angesprochen, kann das RDF-Modell in XML syntaktisch repräsentiert werden. Die entsprechende Grammatik ist in [101] definiert und muß für die weiteren Ausführungen ebenso wie XML und *XML Namespaces* [50] als bekannt vorausgesetzt werden. Das obige Beispiel könnte als XML Dokument dann folgendermaßen aussehen:

```
<?xml version="1.0">
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:s="http://www.faps.de/schema/">
      <rdf:Description about="http://www.ad.siemens.de/S7">
         <s:Hersteller>Siemens</s:Hersteller>
      </rdf:Description>
</rdf:RDF>
```

Um eine Sammlung von Resources oder Literalen leichter modellieren zu können, stellt RDF im Basismodell drei Container-Objekte *(rdfs:Container)* zur Verfügung *Bag (rdf:Bag), Sequence (rdf:Seq)* und *Alternative (rdf:Alt)*. Der Unterschied zwischen Bag und Sequence besteht in der Signifikanz der Reihenfolge der beinhalteten Resources

oder Literalen, die nur bei der Sequence von Bedeutung ist. Wird eine Menge von Resources oder Literalen als Alternative modelliert, so besitzt eine Resource, die diese Alternative als Wert eines Property definiert, jeweils nur einen der Werte der Alternative.

Um RDF-Statements über RDF-Statements definieren zu können (auf der Meta-Metaebene), können sogenannte *Reified Statements* formuliert werden. Dazu werden "normale" RDF-Statements mit Hilfe der speziellen Properties *rdf:predicate*, *rdf:subject*, *rdf:object* und *rdf:type* modelliert. Dies soll am nachfolgenden Beispiel eines Reified Statement verdeutlicht werden (vgl. auch Bild 60)

Stöckel behauptet "Siemens ist der Hersteller der Steuerung S7".

Mittels des in Bild 60 dargestellten ausgezeichneten Properties *rdf:type* wird dabei ein einfaches Typkonzept in RDF eingeführt. Überträgt man obiges Beispiel auf ein klassisches Objektmodell, so bedeutet dies, daß ein instantiiertes Objekt

- in der ausgezeichneten Resource, die über das rdf:type-Property angebunden ist, die Objektklasse bezeichnet (bei einem Reified Statement ist es *rdf:statement*),
- die zu der Klasse gehörenden Instanzvariablen setzen muß (bei einem Reified Statement sind dies *rdf:predicate*, *rdf:subject* und *rdf:object*).

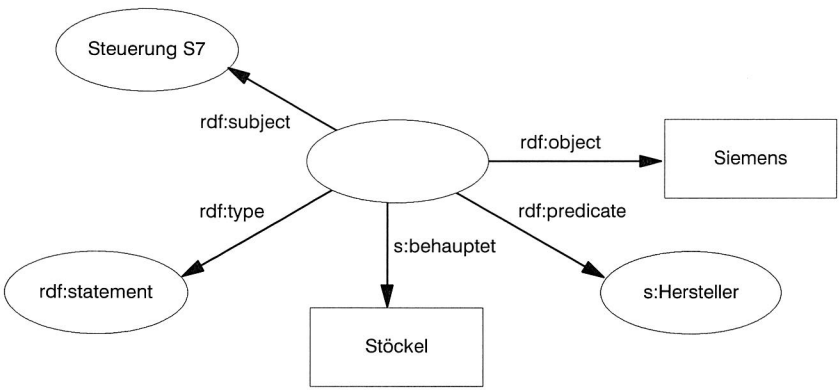


Bild 60: *Graphische Repräsentation eines Reified Statements*

Für verschiedene Typen (rdf:type) können also verschiedene Properties zwingend vorgeschrieben werden. Überhaupt kann man, falls man eine Analogie zu einem Objektmodell aufbauen möchte, Resources allgemein als Objekte betrachten, deren Instanzvariablen in Form von Properties modelliert werden. Der Vollständigkeit halber soll auch dieses Beispiel in XML wiedergegeben werden:

```
<?xml version="1.0">
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:s="http://www.faps.de/schema/">
   <rdf:Description>
      <rdf:subject resource="http://www.ad.siemens.de/S7"/>
      <rdf:predicate resource="http://www.faps.de/schema/Hersteller"/>
      <rdf:object>Siemens</rdf:object>
      <rdf:type resource=
           "http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement/>
      <s:behauptet>Stöckel</s:behauptet>
   </rdf:Description>
</rdf:RDF>
```

Während das RDF-Modell lediglich binäre Relationen unterstützt, da ein Statement nur eine Relation zwischen *zwei* Resources definiert, müssen in der Praxis oft höherstufige Relationen formuliert werden können. Ein Temperaturwert besteht beispielsweise aus dem bloßen Zahlenwert, der die Temperaturhöhe repräsentiert, und der Einheit, in der die Temperaturhöhe angegeben ist. Im nachfolgenden Beispiel sind die beiden Werte des Property "beträgt" doppelt unterstrichen.

Die Temperatur des Kessels beträgt 200 Kelvin.

Zur Darstellung höherstufiger Relationen werden deshalb Resources zwischengeschaltet, denen die "fehlenden" Werte des Property zugeordnet werden. Ein ausgezeichnetes Property namens *value* bezeichnet den "Hauptwert" der Relation (Bild 61).

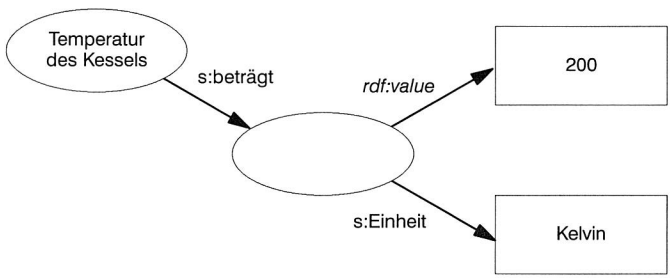


Bild 61: *Repräsentation einer ternären Relation und das ausgezeichnete Property rdf:value*

7.3.3 Die RDF Schema Spezifikation

Die Semantik wird in RDF durch ein anwendungsspezifisches Schema ausgedrückt. Dort werden die in Verbindung mit bestimmten Resources möglichen Properties definiert, Wertebereiche für die Properties festgelegt usw. Im Gegensatz zu einer XML DTD (vgl. Kapitel 2.5.2), die lediglich die Struktur eines XML Dokuments vorgibt, kann mit Hilfe eines RDF-Schemas beschrieben werden, wie die Statements eines RDF-Mo-

dells interpretiert werden müssen. Durch die Verwendung von XML-Namensräumen können in einem RDF/XML-Dokument verschiedene Schemata referenziert werden.

In der RDF-Schema Spezifikation wird eine Sprache definiert, um RDF-Schemata formulieren zu können. Präziser ausgedrückt, handelt es sich um ein Typsystem in welchem spezielle Resources und Properties, wie *rdfs:Class* oder *rdfs:subClassOf*, und deren Bedeutung bei der Definition konkreter anwendungsspezifischer Schemata festgelegt wird. Das Typsystem wird dabei selbst wieder im RDF-Basismodell und damit in Form von Resources, Properties und Statements spezifiziert. Folglich ist auch die RDF Schema Spezifikation in XML darstellbar.

Die Entwicklung des RDF-Schemas wurde von Sprachen zur Wissensrepräsentation und Sprachen zur Schema-Definition in Datenbanksystemen wie *NIAM* [103] beeinflußt. Weiterhin flossen Elemente des Prädikatenkalküls erster Stufe mit ein v.a. der Sprachen *CycL* [104] und *KIF* [105]. Die W3C-Initiativen *PICS* (Platform for Internet Content Selection), *Dublin Core Metadata* und *P3P* (Platform for Privacy Preferences) stellen konkrete Anwendungsgebiete der RDF Schema Spezifikationen dar.

Einen Überblick über die im RDF-Schema definierten Klassen und Resources gibt Bild 62. Mit einem abgerundeten Rechteck werden die Klassen dargestellt. Abgeleitete Klassen werden von der Basisklasse umschlossen. Mit einem Punkt sind Resources bezeichnet. Der Typ *(rdf:type)* einer Resource ist die sie umschließende Resource bzw. die Klasse, auf die der Pfeil deutet. Resources können damit Instanzen einer oder mehrerer Klassen sein, die über das *rdf:type* Property spezifiziert werden. Das RDF-Schema beschreibt die zugehörige Ableitungshierarchie, indem beispielsweise das ausgezeichnete Property *rdfs:subClassOf* verwendet wird. Interessant ist in diesem Zusammenhang, daß nicht nur auf Klassenebene eine Vererbung möglich ist, sondern auch auf Ebene der Properties mittels *rdfs:subPropertyOf.*

Nachfolgend werden die noch nicht im Rahmen des RDF-Basismodells vorgestellten Klassen und Resources beschrieben:

- *rdfs:Resource*
  Alle in RDF modellierten Dinge repräsentieren Resources und sind damit automatisch Instanzen von *rdfs:Resource.*
- *rdf:Property*
  Die Teilmenge der Resources, die Properties darstellen, sind Instanzen von *rdf:Property* (vgl. hierzu auch die Definition 6).

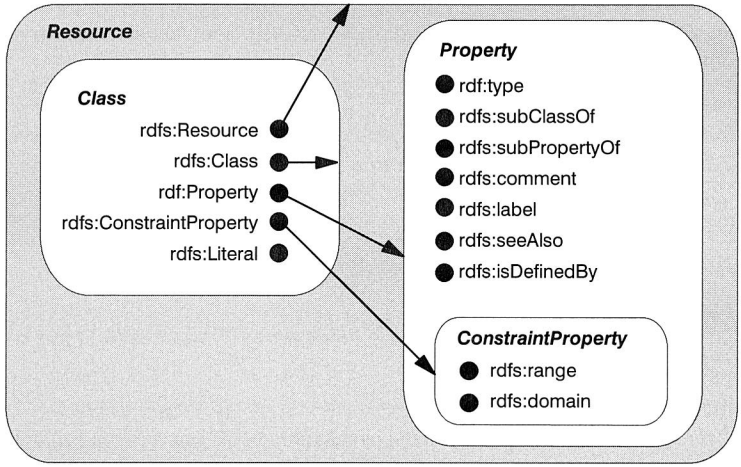


Bild 62: *Klassen und Resources in RDF (nach [102])*[11]

- *rdfs:Class*
  Eine Resource (das "Objekt" in einer objektorientierten Sprache) gehört einer bestimmten Klasse an, die wiederum als Resource modelliert wird. Der Typ der *Klasse (rdf:type)* ist dann *rdfs:Class.*
- *rdf:type*
  Zeigt die Zugehörigkeit zu einer Klasse an ("is a"). Ist der Wert des *rdf:type* Property eine Klasse, so wird die Resource als Instanz der Klasse bezeichnet. Eine Resource kann eine Instanz einer oder mehrerer Klassen sein.
- *rdf:subClassOf*
  Spezifiziert eine transitive Ableitungshierarchie zwischen Klassen.
- *rdfs:subPropertyOf*
  Spezifiziert, daß eine Property eine Spezialisierung einer anderen Property darstellt.
- *rdfs:seeAlso*
  Stellt eine Verweismöglichkeit auf eine beschreibende Resource vom Typ *rdfs:Resource* dar.

11. In dieser Darstellungsweise ist nicht abbildbar, daß *rdfs:ConstraintProperty* sowohl eine Unterklasse von *rdf:property* als auch von *rdfs:ConstraintResource* (nicht abgebildet) ist, die selbst eine Unterklasse von *rdfs:Resource* ist. Ein vollständiges Vererbungsdiagramm oder eine Darstellung im RDF-Basismodell wäre aufgrund der Unterscheidung von Unterklassenbeziehungen und Typbeziehungen zwar präziser, jedoch wesentlich komplexer und im Zuge der ohnehin äußerst knapp gehaltenen Einführung deshalb schwerer verständlich. Für Details sei deshalb nochmals auf die ausführliche Spezifikation verwiesen.

- *rdfs:isDefinedBy*
  Ist definiert als Spezialisierung von *rdfs:seeAlso* und verweist auf die eine Resource definierende Resource.
- *rdfs:ConstraintResource*
  Instanzen dieser Klasse stellen "restringierte" Resources dar, d.h. es werden im Zuge der Beschreibung der Resource *rdfs:ConstraintProperties* verwendet.
- *rdfs:ConstraintProperty*
  Instanzen dieser Klasse sind Properties, die bestimmte Restriktionen beschreiben.
- *rdfs:range*
  Definiert diejenigen Klassen, die als *Werte* eines Properties in Frage kommen ("Wertebereich").
- *rdfs:domain*
  Definiert diejenigen Klassen, für die ein bestimmtes Property anwendbar ist ("Definitionsbereich").

RDF benutzt den aus den XML-Namensräumen bekannten Mechanismus, um Schemata zu identifizieren. Für jedes neue Schema ist eine eigene URI zu verwenden, um jederzeit ein Schema ein-eindeutig referenzieren zu können. Mittels *rdfs:sbClassOf* und *rdfs:subPropertyOf* ist es möglich, Beziehungen zwischen verschiedenen Schemata zu definieren.

Ein grundsätzliches Problem ergibt sich derzeit noch mit den verfügbaren Datentypen. In der RDF-Spezifikationen wird hier explizit auf die XML-Spezifikation von Datentypen [100] verwiesen, die sich jedoch noch im Zustand eines "Working Draft" befindet. Generell ist jedoch mit einer Unterstützung aller in Programmiersprachen üblichen Datentypen zu rechnen.

7.3.4 Zusammenfassung

In der Informatik existiert eine Vielzahl von Metasprachen, Metadatenmodellen und Modellierungssprachen. Das Resource Description Framework eignet sich jedoch aufgrund einer Reihe von Vorteilen besonders als Format für die in Kapitel 7.2.3 noch offen gelassenen Punkte 3 bis 5:

- Bei RDF handelt es sich handelt es sich um eine *rekonstruierte "two-level-language"*, d.h. eine Sprache, in der "sowohl Objekt- als auch Metasprache gesprochen wird" [98]. Indem man in der Ausgangssprache, dem RDF-Basismodell, auch das RDF-Schema ausdrücken kann, ist es überhaupt erst möglich, eine komplette Selbstbeschreibung zu erreichen. Dieser äußerst wichtige Aspekt kann hier leider nicht vertieft werden, es muß dazu auf *Lorenzen [106]* verwiesen werden.

- XML etabliert sich zunehmend als Austauschformat für "semantische" Informationen im WWW, so daß eine schrittweise Ergänzung von HTML um XML für die Zukunft zu erwarten ist. RDF bietet aber gerade die Abbildung auf XML "standardmäßig" an.
- Ein wichtiges Argument, das für den Einsatz von XML und damit für RDF spricht, ist die Vielzahl an Werkzeugen, um XML-Dateien zu parsen und damit die semantischen Informationen zu extrahieren.

7.4 Die Rolle des Objektmodells

Das Objektmodell des Frameworks wird in RDF beschrieben, genauso wie die einzelnen Objekte. Somit ist das Objektmodell auf einfache Weise erweiterbar aber auch austauschbar und stellt nicht – wie sonst eher üblich – den eigentlichen Kern des Frameworks dar. Es ist nicht Ziel dieses Kapitels ein neues und vollständiges Objektmodell zu spezifizieren, das in der Praxis sowieso keine Relevanz haben würde. Vielmehr muß es das Ziel sein, ein vorhandenes und auf breiter Basis akzeptiertes Modell in RDF abzubilden und als Objektmodell festzulegen. Die bei der Umsetzung wichtigsten Punkte sollen hier herausgegriffen werden, ohne auf ein *konkretes* Objektmodell einzugehen.

Um überhaupt einen Eindruck zu bekommen, wie die Umsetzung in RDF aussieht, wurde exemplarisch ein kleiner Ausschnitt der MMS-Spezifikation in leicht vereinfachter Form in Bild 63 abgebildet. Dabei wurde der Schwerpunkt auf die "normalen" Properties einer Resource gelegt, also die Benutzung von RDF zum "Reden in der Objektsprache". Eine vollständige Spezifikation muß zu jeder Resource aber nicht nur die spezifischen Properties sondern auch die Klassenbeziehung (rdfs:subClassOf) und den Typ (rdf:type) angeben, um die Spezifikation selbst auszudrücken ("Reden in der Metasprache"). Beide Angaben wurden aus Gründen der Übersichtlichkeit bei fast allen Resources weggelassen. So ist der Typ der Resource "Variable" wieder "Variable" und der Wert des Properties *rdfs:subClassOf* der Resource "Variable" ist "Object". Würde man die vollständige Abbildung eines Objektmodells wie MMS oder OPC in RDF erwägen, so wäre eine weitestgehende Trennung der Spezifikation in die Teile der Objektsprache und der Metasprache sinnvoller, um nicht die verschiedenen Sprachebenen miteinander zu vermischen.

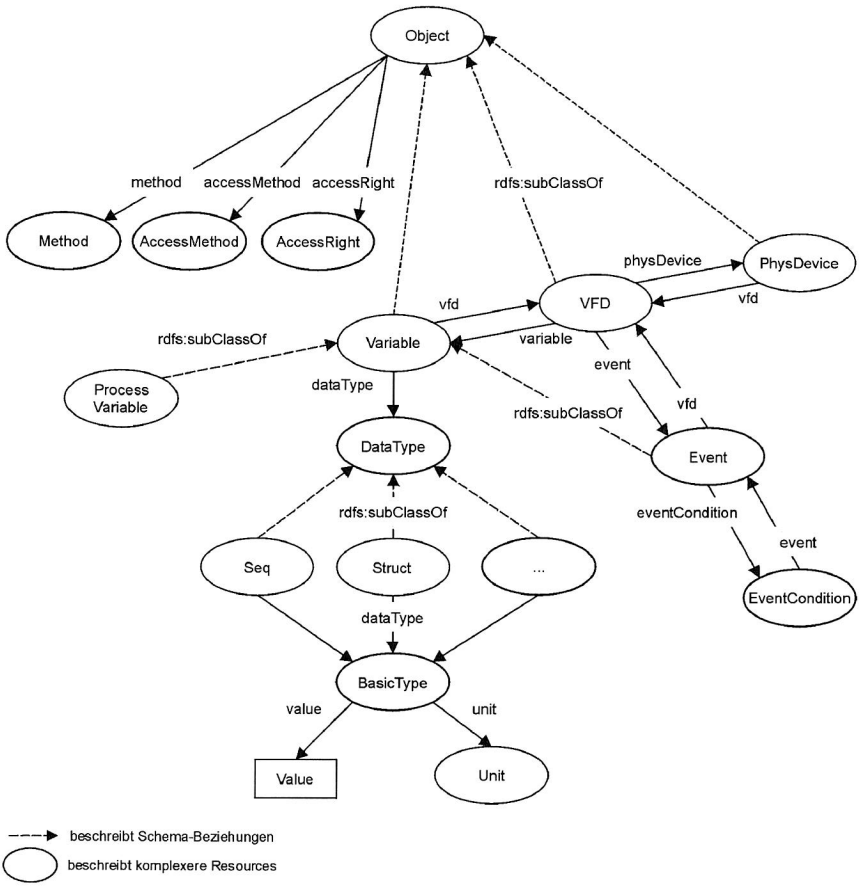


Bild 63: *Auszug aus einem von MMS abgeleiteten Objektmodell*

Die wichtigste Festlegung betreffend das Objektmodell, die auch in den noch folgenden Ausführungen vorausgesetzt wird, ist die Modellierung sämtlicher "Teile" des Feldgeräts als Objekte mit entsprechenden Methoden. Aufbauend auf dieser minimalen Anforderung können – wie in Bild 63 gezeigt – beispielsweise einfache und zusammengesetzte Prozeßvariablen modelliert werden. Ein konkretes Objektmodell unterscheidet sich dann lediglich in einer eventuell etwas veränderten Vererbungshierarchie und zusätzlichen oder anders benannten Properties. Insofern stellt die Art und Weise der Abbildung der Objektattribute und der Objektmethoden auf RDF den eigentlichen Kern des Frameworks dar. Formal wird diese Abbildung in Kapitel 7.5 beschrieben. Zum besseren Verständnis soll jedoch vorab an Hand eines Beispiels dar-

gestellt werden, wie ein Objekt "Prozeßvariable" in RDF/XML beschrieben werden kann. Ausgangspunkt ist die in Tabelle 5 gezeigte Klasse mit den Properties *value, unit, representation* und den Methoden *getValue ( )* und *setValue ( )*:

```
<rdf:RDF xml:lang="en"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:i="http://www.faps.de/schema/basic-type#Integer"
   xmlns:b="http://www.faps.de/schema/basic-type"
   xmlns="http://www.faps.de/schema#">
   <!-- Default Namespace ist das RDF Schema des Objektmodells -->

   <rdf:Description about="soap://gw.faps.de/FB/Device1/VFD2/Temp">

      <!-- Verweis auf das zugehörige VFD-->
      <vfd>
         <rdf:Description about="soap://gw.faps.de/FB/Device1/VFD2">
         </rdf:Description>
      </vfd>

      <!-- Objekttyp -->
      <rdf:type rdf:resource=
           "http://www.faps.de/schema/classes#ProcessVariable"/>

      <!-- Objektattribute -->
      <dataType>
         <rdf:Description about="http://www.faps.de/schema/basic-type">
            <b:unit>Kelvin</b:unit>
            <b:basicType>
               <rdf:Description about=
                    "http://www.faps.de/schema/basic-type#Integer">
                  <i:value>2000</i:value>
               </rdf:Description>
            </b:basicType>
         </rdf:Description>
      </dataType>

      <!-- alternative Zugriffsmethoden -->
      <accessMethod
           resource="iiop://gw.faps.de/FB/Device1/VFD2/Temp"/>
      <accessMethod
            resource="dcom:guid:C2F41010-65B3-11d1-A29F-00AA00C14882"/>
      <!-- Objekt-Methoden -->
      <method>
         <rdf:Description about=
            "soap://gw.faps.de/FB/Device1/VFD2/Temp/setValue">
            <in>
               <rdf:Description
                    about="http://www.faps.de/schema/basic-type">
                  <b:unit>Kelvin</b:unit>
                  <b:basicType>
                     <rdf:Description about=
                          "http://www.faps.de/schema/basic-type#Integer">
                        <i:value></i:value>
                     </rdf:Description>
                  </b:basicType>
```

```
            </rdf:Description>
          </in>
        <rdf:Description>
      </method>

      <method>
        <rdf:Description about=
          "soap://gw.faps.de/FB/Device1/VFD2/Temp/getValue">
          <out>
            <rdf:Description
                about="http://www.faps.de/schema/basic-type">
              <b:unit>Kelvin</b:unit>
              <b:basicType>
                <rdf:Description about=
                    "http://www.faps.de/schema/basic-type#Integer">
                  <i:value></i:value>
                </rdf:Description>
              </b:basicType>
            </rdf:Description>
          </out>
        <rdf:Description>
      </method>
    </rdf:Description>
  </rdf:RDF>
```

Als URI für die Prozeßvariable *Temp* bietet sich die relative Adressierung des Feldbusses ab dem Gateway (Rechnername "gw") an. Dabei erfolgt die Adressierung eines bestimmten Feldgeräts feldbusneutral über dessen eindeutige Nummer am Bus ("Device1"). Die weitere Adressierung ist objektmodellspezifisch – im Beispiel ist die Temperaturvariable dem Virtuellen Feldgerät mit der Nummer zwei ("VFD2") zugeordnet, d.h. es wird das an MMS angelehnte Objektmodell aus Bild 63 verwendet. Der erste Teil einer URI beschreibt stets die Zugriffsmethode auf eine Ressource. Da das Framework SOAP in jedem Fall unterstützen muß, steht hier die entsprechende Kennung.

Eine Besonderheit ist das Property *accessMethod*, das mehrfach vorkommen kann und über das weitere Zugriffsmethoden auf das Objekt definiert werden können. Im Beispiel ist zum einen CORBA angegeben, wobei das Objekt über IIOP und seinen hierarchischen Namen angesprochen wird, und zum anderen DCOM, wobei der GUID (Globally Unique Identifier) das Objekt referenziert.

Die Objektmethoden werden wie das Objekt selbst über eine entsprechend aufgebaute URI referenziert. Die Angabe alternativer Zugriffsmethoden *pro Objektmethode* ist nicht vorgesehen. Über *in*, *out* und *inout* Properties können die Parameter und die Rückgabewerte einer Methode spezifiziert werden. Man erkennt bereits das verfolgte Prinzip als Rückgabewert nicht einfach einen Zahlenwert, sondern eine komplexere Beschreibung im Sinne eines zurückgelieferten Objekts zu verwenden. In Bild 64 ist die Beschreibung der Variable graphisch dargestellt.

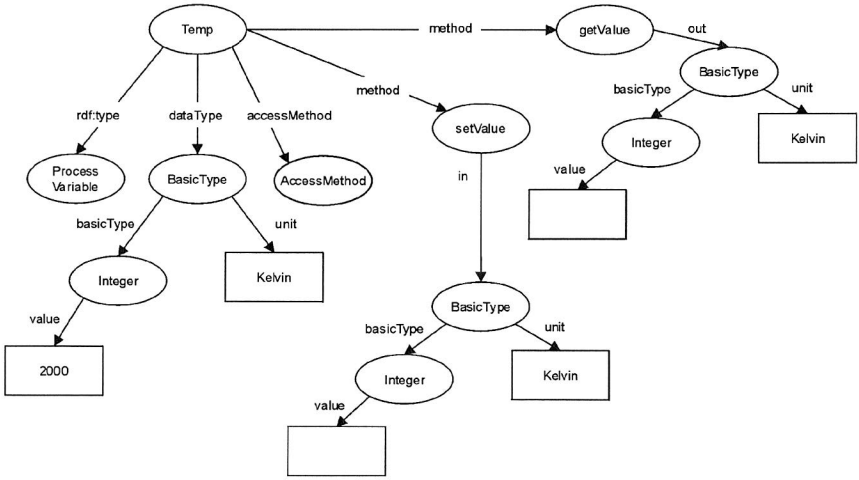


Bild 64: *Graphische Repräsentation der Variable "Temp" im Objektmodell*

7.5 Konkretisierung des Frameworks

In Kapitel 7.2.1 wurden bereits die Grundlagen des Frameworks vorgestellt. Da nun die Modellierungssprache für das Feldgerät und die Objekte mit RDF/XML eingeführt wurde, können einzelne Teile detaillierter besprochen werden.

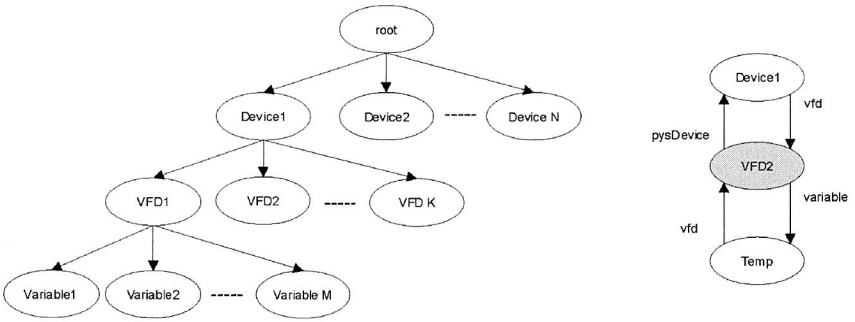
7.5.1 Implementierungsebene

Suchen eines Servers

Als zentraler Zugangspunkt für die Suche eines Feldgeräte-Servers wird ein *Web-Server* auf dem Gateway eingesetzt. Dieser bildet auf einer festen Startseite die Netzstruktur des Feldbussystems ab und bietet über die Navigation von Hyperlinks für jedes Feldgerät den Abruf der hierarchischen Objektstruktur an, wie sie im Objektmodell festgelegt ist. Die Hyperlinks zeigen auf ein am Web-Server installiertes Programm, z.B. ein Servlet, das die bei der Registrierung der Objekte in der Metadatenbank und dem Repositorium hinterlegten Informationen als RDF/XML-Dateien aufbereitet und zurücksendet. Dieses Vorgehen hat einen doppelten Zweck: Zum einen kann eine benutzergeführte Navigation im Web-Browser durch das Objektmodell implementiert werden, indem die RDF/XML-Datei über ein XSL-Style-Sheet für die Anzeige im Web-Browser formatiert wird. Andererseits kann die RDF/XML-Datei geparst werden und somit eine automatisierte Navigation durch das Objektmodell erfolgen. Wiederum er-

weist sich die Flexibilität von RDF/XML als wesentlicher Vorteil, da beispielsweise Automatisierungsaufgaben, für die noch keine Anwendungsklasse im Objektmodell existiert, im RDF-Modell beschrieben werden und diese Beschreibung bei der Navigation des Objektmodells immer zur Verfügung steht.

In Bild 65, das an die vorangegangenen Beispiele anknüpft, wird ein Ausschnitt aus der Navigation durch das Objektmodell gezeigt. Aus Gründen der Vereinfachung sind nur die zu einem VFD gehörenden Variablen dargestellt. Der Client ruft dabei die Information des "VFD2" des physikalischen Feldgeräts mit der Busnummer "1" ab (vgl. Bild 65, (b)).



(a) schematische Darstellung des Objektmodells

(b) Navigation des Objektmodells

Bild 65: *Navigation des Objektmodells am Beispiel der Abfrage der zu einem bestimmten VFD gehörenden Informationen*

Als Rückgabe erhält der Client eine RDF/XML-Beschreibung des VFD2, die wie folgt aussehen könnte:

```
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="http://www.faps.de/schema#">
  <!-- Default Namespace ist das RDF Schema des Objektmodells -->

  <rdf:Description about="soap://gw.faps.de/FB/Device1/VFD2">

    <!-- Verweis auf das Physikalische Gerät -->
    <physDevice>
      <rdf:Description about="soap://gw.faps.de/FB/Device1">
      </rdf:Description>
    </physDevice>

    <!-- Variablen -->
    <variable>
      <rdf:Description
        about="soap://gw.faps.de/FB/Device1/VFD2/Temp">
```

```
        </rdf:Description>
      </variable>

      <!-- Events und andere Objekte des VFD; hier leer -->
    </rdf:Description>
  </rdf:RDF>
```

Die URI-Links, in der RDF/XML-Beschreibung mit der Zugriffsmethode SOAP versehen, kann der Client zur Navigation des Objektmodells benutzen. Dabei ändert der Client in der betreffenden URI die Zugriffsmethode auf HTTP und sendet eine HTTP-Anfrage an den Web-Server. Bei der benutzergesteuerten Navigation erfolgt die Umsetzung der URIs in "href"-Tags – also normale Hyperlinks – durch XSL. Der Web-Server liefert dann die RDF/XML-Beschreibung des betreffenden Objekts. Beispielsweise antwortet der Web-Server auf eine HTTP-Anfrage der Form "http://gw.faps.de/FB/Device1/VFD2/Temp" mit einer Beschreibung des Objekts *Temp*, wie dies in Kapitel 7.4 dargestellt ist.

Eine wichtiger Punkt, der bei der Navigation des Objektmodells und auch bei der später beschriebenen Implementierung des Mergers mit in die Überlegungen einbezogen werden muß, ist die Frage, wer das Property *accessMethod* setzt. Ordnet man diese Aufgabe allein dem Bereich der Objektmodellierung zu, so würde man unnötigerweise auf die Implementierung des Gateways zurückwirken. Denn der Zugriff auf die Feldgeräte-Server wird letztlich über Proxy-Objekte am Gateway realisiert (vgl. weiter unten) und diese können prinzipiell über eine beliebige Middleware mit den Clients kommunizieren. Da somit das Gateway bestimmt, welche Kommunikationsmöglichkeiten zu den darüberliegenden Ebenen verfügbar sind, wird das zugehörige Property *accessMethod* auch hier gesetzt.

Vermittlung eines Methodenaufrufs

Eine Besonderheit stellt der Methodenaufruf dar. Dieser kann zwar prinzipiell nach dem gleichen Schema erfolgen, also über den Web-Server vermittelt werden. Aus Gründen der Effizienz ist es jedoch sinnvoller, daß der Client die URI und die genaue Beschreibung der Methodensignatur nutzt, um unter Umgehung des Web-Servers eine "direkte" Verbindung mit dem Objekt herzustellen, dessen Methode er aufrufen will.

Im Fall von SOAP kann der Client das HTTP-Protokoll benutzen, wobei er seine Anfrage wiederum an das Gateway richtet. Bei CORBA muß der Client zunächst eine Objektreferenz auf das Objekt besitzen. Diese bekommt er durch eine Namensauflösung der URI. Über die in RDF/XML-dargestellte Methodensignatur und die Beschreibung der Datentypen im RDF-Schema kann der Methodenaufruf über das Dynamic Invocation Interface zur Laufzeit zusammengesetzt und gesendet werden.

Im Fall von DCOM gibt es zwar prinzipiell die Möglichkeit einen Methodenaufruf zur Laufzeit zu erzeugen und zu versenden, doch diese im Rahmen des *Automation Inter-*

face realisierte Technik funktioniert gegenwärtig nicht über ein Netzwerk. Hier muß die vom Server gelieferte RDF/XML-Information in einen fest kodierten Methodenaufruf übersetzt werden.

Aufgrund der in den Kapiteln 4 bis 6 gewonnenen Erfahrungen hat man beim Übergang auf das Feldbussystem nur eine Alternative, wenn man Methodenaufrufe in beiden Richtungen und insbesondere direkt vom Feldgerät aus ermöglichen will – die Implementierung einer Middleware auf dem Feldbussystem. Am Gateway vermitteln dann Proxyobjekte und eine Objekt-Bridge die Aufrufe auf den Feldbus bzw. von diesem nach außen. Sucht ein Feldgerät einen Server außerhalb der Prozeßebene, so muß er die ihm zur Verfügung stehenden, Middleware-spezifischen Dienste nutzen. Damit wendet es sich an das Gateway, das nun seinerseits alle ihm verfügbaren Dienste nutzen kann. Es ist davon auszugehen, daß bestehende Anwendungen außerhalb der Prozeßebene das Framework lediglich nutzen, aber nicht selbst implementieren. Somit kann das Gateway die Framework-Mechanismen zum Finden eines Servers nicht einsetzen. Eine Ausnahme ist die horizontale Kommunikation (vgl. Anforderung K2 aus Kapitel 3.2), wenn sich das Gateway selbst wieder an ein anderes Gateway wenden kann.

Wie bereits angesprochen, soll im Rahmen des Middleware-Frameworks keine spezielle Middleware vorgegeben werden, dies soll erst recht für die Prozeßebene gelten. Nachdem jedoch explizit eine Unterstützung für SOAP *außerhalb* der Prozeßebene gefordert wird, könnte man bei durchgängiger Verwendung von SOAP auch auf dem Feldbus auf eine Umsetzung der Middleware-spezifischen Adressierung der Objekte am Gateway verzichten.

Einen Sonderfall des Methodenaufrufs bilden die Ereignisse, also spontane Nachrichten eines Feldgeräts an interessierte Clients. Über die im Objektmodell vorgesehene Klasse "Event" besteht dabei die Möglichkeit, nicht nur die von einem VFD angebotenen Ereignisse einzusehen, sondern auch sich für jedes einzelne zu registrieren. Die Registrierung ist ein spezieller Methodenaufruf, der auf zweierlei Weise implementiert werden kann. Zum einen kann das Feldgerät selbst eine Liste der Abonnenten für bestimmte Ereignisse führen, zum anderen kann es aus Gründen der ohnehin knappen Ressourcen sinnvoll sein, die Registrierung am Gateway selbst vorzunehmen. Das Feldgerät übermittelt im letzteren Fall *jedes* Ereignis an das Gateway, das nun seinerseits die jeweiligen Clients informiert.

7.5.2 Ebene der Anwendungslogik

Modellierung der Objekte in RDF

Die Modellierung der Objekte stellt in RDF kein weiteres Problem dar. Es genügt hier auf das in Kapitel 7.4 dargestellte Beispiel zu verweisen. Eine Besonderheit ist jedoch die Repräsentation der Datentypen. Die Resource *BasicType* (vgl. Bild 63 und 64)

stellt nämlich eine künstliche Oberklasse zu den tatsächlichen Basisdatentypen dar, wie sie im Rahmen der Spezifikation der XML-Datentypen in RDF verfügbar sind. Der Vorteil dieser Vorgehensweise ist eine Trennung des bloßen Wertes eines Datentyps von zusätzlichen Attributen, wie physikalischer Einheit oder Repräsentation bzgl. einer bestimmten Normierungsgröße. Dies erlaubt eine Trennung des Wertes eines Datums von seinen Metadaten, wie es für den Austausch der Produktivdaten zwischen Feldgerät und Gateway sinnvoll ist (vgl. Bild 56). Dieser Punkt wird bei einer genaueren Betrachtung der Arbeitsweise des Mergers weiter unten wieder aufgegriffen.

Modellierung der Methoden in RDF

Methoden werden in RDF als Resources, die die Properties *in, inout* und *out* besitzen können, abgebildet. Das Property *in* bezeichnet dabei die Übergabeparameter eines Methodenaufrufs, das Property *out* die Rückgabewerte und das Property *inout* Werte, die sowohl übergeben werden als auch in der Rückgabe vorkommen. Ein Methodenaufruf erfolgt stets durch Wertübergabe ("call by value").

In den bisher gezeigten Beispielen wurde lediglich der Datentyp "Integer" verwendet und für dessen Definition auf das interne Schema verwiesen. Mit der Verfügbarkeit einer stabilen Version der Spezifikation der XML-Datentypen (vgl. [100]) können alle aus Programmiersprachen bekannten Datentypen bis auf Zeiger in XML direkt dargestellt werden. Die untenstehende RDF/XML-Deklaration stellt einen Auszug aus dem RDF-Schema des bisher verwendeten Objektmodells dar ("http://www.faps.de/schema"), der die Abbildung der Methoden beschreibt. Dabei werden die aus der RDF Schema Spezifikation bekannten Properties *range* und *domain* eingesetzt, um für die neu definierten Klassen bzw. Properties Wertebereiche und Definitionsbereiche festzulegen.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

   <rdfs:Class rdf:ID="Variable">
      <rdfs:comment>Die Klasse Variable</rdfs:comment>
      <rdf:subClassOf rdf:resource="#Object"/>
   </rdfs:Class>

   <rdf:Property rdf:ID="method">
      <rdfs:domain rdf:resource="#Object"/>
      <rdfs:range rdf:resource="#Method"/>
   </rdf:Property>

   <rdfs:Class rdf:ID="Method"/>

   <rdf:Property rdf:ID="in">
      <rdfs:domain rdf:resource="#Method"/>
      <rdfs:range rdf:resource=
           "http://www.faps.de/schema/datatypes#DataType"/>
   </rdf:Property>
```

```
    <rdf:Property rdf:ID="inout">
      <rdfs:domain rdf:resource="#Method"/>
      <rdfs:range rdf:resource=
           "http://www.faps.de/schema/datatypes#DataType"/>
    </rdf:Property>

    <rdf:Property rdf:ID="out">
      <rdfs:domain rdf:resource="#Method"/>
      <rdfs:range rdf:resource=
           "http://www.faps.de/schema/datatypes#DataType"/>
    </rdf:Property>
  </rdf:RDF>
```

Registrierung der Objekte am Gateway

Für die Registrierung der Objekte eines Feldgeräts am Gateway bieten sich zwei Lösungen an. Zum einen kann der Hersteller die RDF/XML-Beschreibungen zusammen mit dem Gerät auf einer Diskette ausliefern, die am Gateway eingespielt wird. Die elegantere Möglichkeit ist es, diese Beschreibungen direkt auf dem Feldgerät zu hinterlegen und, wie in Kapitel 5 beschrieben, entsprechende Übertragungsdienste über den Feldbus und Registrierungsdienste am Gateway vorzusehen.

Wie die RDF/XML-Beschreibung letztlich am Gateway gespeichert wird ist Teil der Implementierung. Der Dienst, der in Zusammenarbeit mit dem Web-Server die Navigation des Objektmodells und damit die Suche eines Servers ermöglicht, muß diese Informationen lediglich parsen und in Form eines *Document Object Model* (DOM) aufbereiten, um die bei einer Anfrage benötigten Teile der Objektmodellbeschreibung extrahieren und zurückliefern zu können.

Implementierung des Mergers

Der nachfolgende Ausschnitt aus der RDF/XML-Beschreibung der Methode *getValue* der Prozeßvariable *Temp* zeigt noch einmal, die bei der Modellierung der Daten realisierte Trennung des Datenwerts (kursiv dargestellt) von den Metadaten.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="http://www.faps.de/schema#">
  <!-- Default Namespace ist das RDF Schema des Objektmodells -->

  <rdf:Description about=
    "soap://gw.faps.de/FB/Device1/VFD2/Temp/getValue">
    <out>
      <rdf:Description
           about="http://www.faps.de/schema/basic-type">
        <b:unit>Kelvin</b:unit>
          <b:basicType>
            <rdf:Description about=
                 "http://www.faps.de/schema/basic-type#Integer">
              <i:value> <!-- hier wäre der Datenwert --> </i:value>
```

```
            </rdf:Description>
          </b:basicType>
        </rdf:Description>
      </out>
    </rdf:Description>
  </rdf:RDF>
```

Damit der Merger zur Laufzeit die Produktivdaten mit den Metadaten zusammenführen kann, benötigt er im Fall eines Methodenaufrufs bzw. eines Events die zugehörige URI. Damit kann er nicht nur den Objektnamen, das VFD und das physikalische Gerät ablesen, sondern auch weitere Informationen aus der RDF/XML-Beschreibung des zugehörigen Feldgeräts nachschlagen. Prinzipiell genügt es dann, die bei der Methodenbeschreibung mit *inout* bzw. *out* bezeichneten Parameterbeschreibungen zu extrahieren, damit eine neue RDF/XML-Beschreibung zu erstellen und an Stelle des offen gelassenen Datenwertes den aktuellen Wert einzutragen. Die Daten sind dann über die Attribute *unit* und *representation* (hier nicht dargestellt) beschrieben. Eine XML-Namensraum Deklaration, die auf das verwendete RDF-Schema verweist, ist Bestandteil jeder RDF/XML-Kodierung, so daß der Client immer auch die Informationen über das verwendete Objektmodell vorliegen hat.

Reicht nun der Client seinerseits die Daten an einen Dritten weiter, so kennt der Dritte nicht deren Herkunft. Dem kann dadurch abgeholfen werden, daß ein zusätzliches Property, beispielsweise *rdfs:seeAlso,* mitgeführt wird, das im Fall eines Methodenaufrufs auf die zugehörige Methode verweist, im Fall eines Events auf das Event-Objekt. Denkbar wäre es auch, direkt über das Property *accessMethod* auf die Methode oder das Event-Objekt zu verweisen. Wie in Kapitel 7.5.1 bei der Suche eines Servers beschrieben, würde dann der Merger das entsprechende Property erzeugen.

Vergleicht man abschließend den Informationsgehalt, der durch den Merger übertragen werden muß, so ist dieser identisch zu der oben gezeigten Methodenbeschreibung, wobei lediglich der Wert des Properties *value* mit dem tatsächlichen Prozeßwert gesetzt ist.

7.6 Zusammenfassung und Bewertung

Gegenstand eines Middleware-Frameworks ist es zu definieren, wie Anwendungsobjekte allgemein repräsentiert werden, welche Schnittstellen und Dienste das Framework bzw. die darunterliegende Schicht der Middleware anbietet und wie die Abbildung des Frameworks auf die Middleware aussieht. Im Gegensatz zu einem Anwendungs-Framework, das die Unterstützung konkreter Anwendungen verfolgt, und für die auszutauschenden Daten gezielt Dienste bereitstellt, bleibt in einem Middleware-Framework die Frage offen, welche Art von Informationen letztlich ausgetauscht werden.

Das in diesem Kapitel vorgestellte Middleware-Framework wurde in Hinblick auf die Erfüllung aller in Kapitel 3 formulierten Anforderungen entwickelt. Dieses Ziel wurde

mit der Einführung eines generischen, zur Selbstbeschreibung fähigen Modellkonzepts auf Basis von RDF und XML erreicht. Bei der unterlagerten Middleware wurde mit der geforderten Unterstützung für SOAP und der Möglichkeit zur Angabe weiterer Middleware versucht, dem Dilemma auszuweichen, sich auf eine bestimmte Technologie von vornherein festlegen zu müssen.

Anforderung K1: Unterstützung der vertikalen Kommunikation 

Die vertikale Kommunikation wird unterstützt. Dabei kann ein Feldgerät sowohl in der Rolle des Servers agieren, aber auch selbst als Client die Dienste eines Servers außerhalb der Prozeßebene in Anspruch nehmen. Innerhalb der Prozeßebene wird dem Feldgerät in der Regel jedoch nicht das komplette Framework, sondern lediglich die Middleware-Schicht zur Verfügung stehen. Für die Kommunikationsverbindung außerhalb der Prozeßebene bis zum Gateway zeichnet die am Gateway implementierte Architektur verantwortlich. Diese kann neben SOAP weitere Middleware-Technologien unterstützen und dem Client beim Kommunikationsaufbau (vgl. "Suchen eines Servers", Kapitel 7.5.1) entsprechend mitteilen.

Anforderung K2: Unterstützung der horizontalen Kommunikation 

Die geforderte horizontale Kommunikation innerhalb der Prozeßebene wird unterstützt. Dabei wird vom Client und vom Server, die sich an einem Feldbussystem befinden, jeweils nur die Middleware-Schicht genutzt. Die notwendige Kooperation der beiden Gateways, die außerhalb der Prozeßebene erfolgt, kann unter Zuhilfenahme der Framework-Dienste wesentlich vereinfacht werden.

Anforderung K3: Unterstützung für ein verteiltes Client/Server-Modell 

Ein verteiltes Client/Server-Modell wird durch die Middleware bereitgestellt und in Form der im Objektmodell definierten Dienste den Clients in der Zellen- bzw. Leitebene angeboten.

Anforderung M1: Umfassendes, generisches Modellkonzept 

Mit RDF bietet das Middleware-Framework ein allgemeines, um Vererbungs- und Klassenkonzepte erweitertes Entity-Relationship Modell. Dieses erlaubt die Abbildung beliebig komplexer Geräte- und Anwendungsmodelle.

Anforderung M2: Erweiterbarkeit 

Das in RDF spezifizierte Modell kann beliebig erweitert werden. Jedes in RDF/XML modellierte Objekt trägt zudem einen Verweis auf sein Objektmodell in Form eines Verweises auf sein RDF-Schema. Damit können Clients das Objektmodell zur Laufzeit interpretieren und so auf Erweiterungen flexibel reagieren.

Anforderung M3: Leichte Integrierbarkeit in bestehende Anwendungen 

Die technische Integrierbarkeit hängt wesentlich von der Verfügbarkeit von XML/RDF-Parsern ab, die die Interpretation der Modelle in der jeweiligen Anwendung ermögli-

chen. Hier bestehen im Bereich RDF noch Defizite, die letztlich zu einem Abschlag in der Bewertung führen. Auf semantischer Ebene ist aufgrund der Selbstbeschreibung die Integrierbarkeit in bestehende Anwendungen vollständig erfüllt.

Anforderung M4: Selbstbeschreibung 

Das RDF-Basismodell und die RDF Schema Spezifikation stellen die notwendigen Sprachkonzepte zur Verfügung, die es ermöglichen gleichermaßen "Objektsprache" und "Metasprache" zu sprechen. Durch den oben beschriebenen Verweis eines in RDF/XML modellierten Objekts auf sein RDF-Schema, trägt jedes Objekt die Beschreibung seines Objektmodells bei sich.

Jedoch sind die Mechanismen, die RDF anbietet, für eine vollständige Selbstbeschreibung nicht in jedem Fall ausreichend. Beispielsweise können die folgenden in der RDF Schema Spezifikation beschriebenen Semantiken nicht in RDF formuliert werden:

- Die Forderung, daß die Properties *rdfs:subClassOf* und *rdfs:subPropertyOf* keine Zyklen bilden dürfen, kann in RDF nicht ausgedrückt werden.
- Die mit der Benutzung eines bestimmten *Constraints* verbundene Bedeutung, kann in RDF ebenfalls nicht ausgedrückt werden.

Der erste Punkt ist weniger kritisch zu betrachten, da potentielle Clients sich dieser grundlegenden Eigenschaft eines Objektmodells einfach "bewußt" sein müssen. Hingegen kann es für einen Client bei der automatisierten Bearbeitung eines ihm unbekannten Objektmodells durchaus wichtig sein, die Semantik einer "ConstraintResource" zu erkennen. Entsprechende Erweiterungen der RDF Schema Spezifikation sind in diesem Bereich deshalb vorgesehen.

8 Zusammenfassung und Ausblick

Die Anforderungen an die Flexibilität und Offenheit der Informations- und Kommunikationssysteme innerhalb eines Unternehmens haben sich in den vergangenen Jahren stark verändert. Zum einen bedingen neue zwischenbetriebliche Kooperationsformen eine engere Verknüpfung von Anwendungssystemen über Unternehmensgrenzen hinweg. Zum anderen stellen neue Anwendungsfelder, wie die betriebsbegleitende Simulation oder die Ferndiagnose und Fernwartung technischer Anlagen über das Internet, immer höhere Anforderungen an die Integration der innerbetrieblichen Informationsverarbeitung. Dabei spielt die Verfügbarkeit und der Zugriff auf aktuelle Betriebsdaten eine sehr wichtige Rolle, um Entscheidungen schnell und auf der Grundlage fundierter Daten treffen zu können.

Die fehlende Integration der Prozeßebene in eine unternehmensweite Informationsverarbeitung erweist sich jedoch in zunehmendem Maße als Hindernis. Mit dem Einsatz von Feldbussystemen und der Ersetzung der bis dahin analog erfolgten Signalübertragung sollte dieses Problem ursprünglich gelöst werden. Tatsächlich erschwerte aber die Einführung einer Vielzahl unterschiedlicher und zueinander inkompatibler Bussysteme den Übergang von der Prozeßebene zu der darüberliegenden Zellenebene. Die gegenwärtigen Bemühungen der Systemhersteller, Feldbushersteller und Anwender konzentrieren sich deshalb darauf, an dieser Stelle eine feldbusneutrale Zugriffstechnologie zu etablieren. In der Praxis ist ein solcher Ansatz hinsichtlich der Funktionalität auf die Schnittmenge der von den unterlagerten Feldbussystemen angebotenen Dienste und damit der Produktivdatendienste beschränkt. Eine direkte Nutzung der von einem Feldgerät angebotenen Daten und Funktionen ist damit nicht möglich.

Wie im Verlauf der Arbeit gezeigt wurde, basiert eine umfassende Lösung des Integrationsproblems auf der Beseitigung der sie verursachenden Heterogenität. Dabei kann man unterscheiden zwischen der Heterogenität auf Ebene der Implementierung und auf Ebene der Anwendungslogik. Dies wurde insbesondere bei der Formulierung von Anforderungen berücksichtigt, die qualitativ beschreiben, wie ein geeignetes Lösungskonzept gestaltet werden muß. Indem eine Bewertung bestehender Systeme an Hand der Anforderungen vorgenommen wurde, konnten dort die wesentlichen Defizite deutlich herausgearbeitet werden.

In einem ersten Schritt wurde die Erweiterung der vorhandenen Systeme mit dem Ziel einer weitergehenden Erfüllung der Anforderungen betrieben. Zunächst wurde eine Architektur konzipiert, die auf der Basis von Geräteprofilen funktional äquivalente Softwarekomponenten anbietet, welche in der Zellen- und Leitebene eingesetzt werden können. Der Grundgedanke dabei ist, daß die in den Geräteprofilen getroffenen Festlegungen über die in einem bestimmten Anwendungsfall mit einem Feldgerät auszutauschenden Informationen und die möglichen Interaktionen direkt in entsprechende Softwarekomponenten übersetzt werden können. Diese realisieren zudem eine di-

rekte Kommunikationsverbindung zu dem betreffenden Feldgerät. Als wesentliche Vorteile der Architektur sind die Verkürzung der Zeiten zur Anwendungserstellung und zur Anwendungsintegration bei jenen Automatisierungsaufgaben zu nennen, die in Form von Profilen bereits standardisiert vorliegen.

Eine Forderung, die die Anwender im Zusammenhang mit dem Einsatz von Feldbussystemen immer wieder erhoben haben, ist die "Plug and Play"-Fähigkeit der Feldgeräte. Ziel ist es dabei, ein am Feldbus neu angeschlossenes Gerät unverzüglich betreiben und von der Zellen- bzw. Leitebene aus ansprechen zu können, idealerweise ohne vorher manuell Software-Installationen durchführen zu müssen. Dieser Gedanke wurde in einer weiteren Architektur aufgegriffen, bei der die Feldgeräte mit gerätespezifischen Softwarekomponenten in Form von Java-Applets ausgestattet sind und die alle zur Konfiguration und zum Betrieb notwendigen Funktionen umfassen. Durch die Nutzung entsprechend konzipierter Übertragungsmechanismen können Clients aus darüberliegenden Ebenen oder über das Internet diese Komponenten laden und lokal ausführen. Dabei erfolgt automatisch der Aufbau einer virtuellen Kommunikationsverbindung zu dem betreffenden Feldgerät, über die anschließend anwendungsspezifische Dienste zur Verfügung stehen. Der wichtigste Vorteil der Architektur liegt in der schnelleren Inbetriebnahme der Geräte, da aufwendige Installationen entfallen. Damit kommt sie gerade dem "Plug and Play"-Gedanken einen wesentlichen Schritt näher.

Ein drittes Konzept beinhaltet die Integration der Prozeßebene über eine einheitliche Middleware-Schicht. Mit FIMO wurde eine spezielle, zu CORBA konforme, jedoch für die besonderen Belange der Prozeßebene adaptierte, Middleware-Architektur entwickelt. Sie orientiert sich dabei an der minimumCORBA-Spezifikation, die speziell auf Embedded Systeme ausgerichtet ist. Als Besonderheiten verfügt FIMO über einen verteilten Namensdienst und eine Bridge zur Anbindung an die darüberliegende Zellenebene. An Hand einer Implementierung über dem Feldbussystem PROFIBUS-FMS konnten die aufgestellten Konzepte erfolgreich verifiziert werden. Der wesentliche Vorteil dieses Ansatzes liegt in der aus Sicht eines Anwendungsprogrammierers vollzogenen, vollständigen Aufhebung der Grenzen zwischen den verschiedenen Kommunikationsebenen.

Es wurde bei der Erweiterung der vorhandenen Systeme deutlich, daß nur ein umfassender Ansatz, wie er im Zuge eines Middleware-Frameworks erarbeitet werden kann, in der Lage ist, die geforderten Anforderungen auf Ebene der Implementierung und der Anwendungslogik gleichermaßen zu erfüllen. Zwei wichtige Entwurfskriterien konnten in diesem Zusammenhang identifiziert werden: Die Vermeidung der Festlegung auf eine bestimmte Middleware-Implementierung sowie die Bereitstellung neutraler Beschreibungsmethoden zur Abbildung der Daten und Funktionen eines Feldgeräts. Dem ersten Punkt wurde dadurch Rechnung getragen, daß verschiedene Zugriffsmethoden auf einzelne Objekte explizit angegeben werden können. Dennoch muß gewissermaßen als "kleinster gemeinsamer Nenner" eine bestimmte Zugriffsmethode immer möglich sein. Hierzu wurde SOAP vorgeschlagen, ein einfaches und

neutrales Punkt-zu-Punkt Verbindungsprotokoll, das über HTTP implementiert werden kann. Der zweite Punkt konnte mit dem Einsatz von RDF sowohl zur Gerätemodellierung als auch zur Anwendungsmodellierung erfüllt werden. RDF zeichnet sich als Metasprache aus, die es ermöglicht, neben den reinen Objektbeschreibungen auch Informationen über das eingesetzte Modell zu liefern. Diese Eigenschaft wurde u.a. dazu benutzt, um die von einem Feldgerät im Zuge eines Methodenaufrufs an einen Client zurückgesendeten Daten mit entsprechenden Metadaten zu versehen. Auf diese Weise wird eine vereinfachte Daten- und Funktionsintegration in bestehende Anwendungen erreicht.

Die in dieser Arbeit vorgestellten Konzepte und Implementierungen zeigen deutlich die Potentiale der Feldbussysteme auf. Sie vermögen nicht nur den Bereich der Echtzeitdatenübertragung in der Prozeßebene abzudecken, sondern mittels geeigneter Erweiterungen auch die effektive Integration in eine unternehmensweite Informationsverarbeitung zu unterstützen. Im Gegensatz zu den seit kurzem wieder intensiv diskutierten Ethernet-basierten Übertragungsverfahren, kann die Feldbustechnik dabei auf eine im Verlauf von mehr als 15 Jahren nunmehr erprobte Technologie verweisen.

In Verbindung mit dem zu erwartenden Vordringen von Ethernet in Verbindung mit TCP/IP in die nicht zeitkritischen Bereiche der Zellenebene und eventuell der Prozeßebene können schnell die gleichen Probleme wie bei den heutigen Feldbussystemen entstehen, da noch keine einheitliche Anwendungsschicht für die Automatisierungstechnik existiert. Auch hier können die in dieser Arbeit vorgeschlagenen Konzepte als wegweisend für zukünftig anstehende Integrationsbemühungen betrachtet werden.

Literaturverzeichnis

1. Weule, H.: "Information als Produktionsfaktor". In: Görke, W.; Rinisland. H.; Syrbe, M. (Hrsg.): *Information als Produktionsfaktor. (GI Jahrestagung Karlsruhe 1992).* Berlin u.a.: Springer Verlag, 1992, (Reihe Informatik aktuell), S. 3–19

2. Schmid, B.: "Nutzen der Ressource Information für das Unternehmen". In: *Erfolgsfaktor Information – 5. Fachinformationskongreß für Unternehmer und Führungskräfte (München 1993).* München: Bayerisches Staatsministerium für Wirtschaft und Verkehr, 1993

3. Picot A.; Reichwald, R.; Wigand R.T.: *Die grenzenlose Unternehmung.* 2., aktualisierte Auflage. Wiesbaden: Gabler Verlag, 1996

4. Mertens, P.: "Virtuelle Unternehmen". In: *Wirtschaftsinformatik* (1994) 2, S. 169–172

5. Arnold, O.; Faisst, W.; Härtling, M.; Sieber, P.: "Virtuelle Unternehmen als Unternehmenstyp der Zukunft?". In: *Handbuch der maschinellen Datenverarbeitung* 185 (1995), S. 8–23

6. Feldmann, K.; Rottbauer, H.; Stöckel, T.: "Information Systems Architecture for Collaborative Manufacturing in Virtual Enterprises". In: *Proceedings of the 10th PROLAMAT Conference.* Trento, Italy, 1998

7. Göhringer, J.: "Nutzen und Realisierung von Telediagnosesysteme zur Anlagebetreuung". In: VDI-Bildungswerk (Hrsg.): *Auslegung und Betrieb flexibler Montagesysteme*. Düsseldorf: VDI-Verlag, 1999

8. N.N.: "PROFIBUS – Der internationale, offene Feldbusstandard". Foliensatz zu PROFIBUS, PROFIBUS International, 1997

9. Bender, K. (Hrsg.): *PROFIBUS: Der Feldbus für die Kommunikation.* 2., überarbeitete Auflage. München, Wien: Carl Hanser Verlag, 1992

10. Wedekind, H.: "HEDAS: Heterogene, durchgängige Anwendungssysteme". In: *Von der Informatik zu Computational Science und Computational Engineering.* Arbeitsberichte des IMMD der Friedrich-Alexander-Universität Erlangen-Nürnberg anläßlich des Abschlußkolloquiums des SFB 182. Band 31. Nummer 6. Erlangen, Oktober 1998, S. 36–59

11. Hempen, U.; Niemann, K.-H.: "Kleine Revolution in der Prozeßautomation". In: *Elektronik* (1999) H. 21, S. 66–72

12. Diedrich, C.: "Etappen zur Integration intelligenter Feldgeräte in Leitsysteme". In: *atp – Automatisierungstechnische Praxis* 41 (1999) 10, S. 14–23

13. ISO 7498: *Information technology – Open Systems Interconnection – Basic Reference Model.* Genf: International Organization for Standardization, 1989

14. Badach, A.; Hoffmann, E.; Knauer, O.: *High Speed Internetworking – Grundlagen und Konzepte für den Einsatz von FDDI und ATM.* Bonn u.a.: Addision-Wesley, 1995

15. Blumann, W.; Horstmann, A. (Hrsg.): *Fertigungsautomatisierung mit MMS.* Düsseldorf: VDI-Verlag, 1993

16. REFA – Verband für Arbeitsstudien und Betriebsorganisation e.V. (Hrsg.): *Methodenlehre der Planung und Steuerung.* Teil 3, München: Hanser Verlag: 1985

17. Mertens, P.; Faisst, W.: "Virtuelle Unternehmen, eine Organisationsstruktur für die Zukunft?". In: *technologie & management* 44 (1995) 2, S. 61–68

18. Schneider, H.-J.: "Licht und Schatten digitaler Feldgeräte (Teil 1)". In: *atp – Automatisierungstechnische Praxis* 40 (1998) 2, S. 13–21

19. Schneider, H.-J.: "Licht und Schatten digitaler Feldgeräte (Teil 2)". In: *atp – Automatisierungstechnische Praxis* 40 (1998) 3, S. 44–52

20. Pfleger, J.A.H.: "Verteilung der Automatisierungsaufgaben bei Feldbuseinsatz". In: *atp – Automatisierungstechnische Praxis* 40 (1998) 3, S. 40–43

21. Feldmann, K.; Solvie, M.; Stöckel, T.: "Dezentralisierung von Steuerungsaufgaben durch Einsatz von intelligenten Feldgeräten am Beispiel einer Lageregelung". In: *Tagungsband zur Inet'96 (Karlsruhe 1996).* Feldkirchen: Franzis-Verlag, 1996, S. 178–185

22. Feldmann, K.; Solvie M.; Stöckel, T.: "Decentralisation of control tasks using intelligent field devices illustrated by the example of a NC–axis controller based on PROFIBUS-FMS". In: *Conference Proceedings FieldComms EUROPE 1996.* Brüssel: 1996.

23. Ergebnisprotokoll über das Fachgespräch "Modulare prozeßnahe Automatisierungs- und Steuerungssysteme", Forschungszentrum Karlsruhe: 26. April 1996

24. Solvie, M.: *Zeitbehandlung und Multimedia-Unterstützung in Feldkommunikationssystemen.* München, Wien: Carl Hanser Verlag, 1996 (Dissertation)

25. Kriesel, W.; Lippik, D.; Heimbold, T.: "Universelle Feldbuskopplung für Sensoren und Aktoren". In: *atp – Automatisierungstechnische Praxis* 39 (1997) 8, S. 43–50

26. DIN EN 50170: *General purpose field communication system.* 1996

27. N.N.: "Der Markt für Feldbussysteme 1998". CONSULTIC Marketing & Industrieberatung GmbH, Großostheim: Mai 1999

28. DIN EN 50254: *High Efficiency Communication Subsystem for Small Data Packages.* 1998

29. prEN 50325: *Industrial communication subsystem based on ISO 11898 (CAN) for Controller-device Interfaces.* 1999

30. Balzert, H.: *Lehrbuch der Software-Technik: Software-Entwicklung.* Heidelberg u.a.: Spektrum Akademischer Verlag, 1996

31. Schwarz, M.: "Moment, ich verbinde...". In: *c't – Magazin für Computertechnik,* 1997, H. 3, S. 256 ff

32. Orfali, R.; Harkey, D.; Edwards, J.: *The Essential Client/Server Survival Guide – Second Edition.* New York u.a.: John Wiley, 1996

33. ISO: *Manufacturing Message Specification.* Part I: Service Definition; Part II: Protocol Specification. IS 9506. Genf: International Organization for Standardization, 1990

34. Bernstein, P.A.: "Middleware: A Model for Distributed System Services". In: *Communications of the ACM,* Vol. 39, Nr. 2, Feb. 1996, S.86–98

35. Object Management Group: *A Discussion of The Object Management Architecture.* Januar 1997, Object Management Group (http://www.omg.org)

36. Object Management Group: *The Common Object Request Broker: Architecture and Specification.* Revision 2.3.1, Oktober 1999, Dokument Nr. 99-10-07, Object Management Group (http://www.omg.org)

37. Object Management Group: *CORBAservices: Common Objects Services Specification.* Dezember 1998, Dokument Nr. 98-12-9, Object Management Group (http://www.omg.org)

38. Object Management Group: *Common Facilites Architecture.* Revision 4.0, Juli 1998, Dokument Nr. 98–07–10, Object Management Group (http://www.omg.org)

39. Redmond, F. E. III: *DCOM – Microsofts Distributed Component Object Model.* Foster City, CA u.a.: IDG Books Worldwide, 1997

40. Orfali, R.; Harkey, D.; Edwards, J.: *The Essential Distributed Objects Survival Guide*. New York u.a.: John Wiley, 1996

41. Telcordia Technologies: Internet Growth Reports (http://www.netsizer.com/)

42. Berners-Lee, T.; Connolly, D.: *Hypertext Markup Language – 2.0*. RFC 1866, Internet Engineering Task Force (IETF), 1995

43. N.N.: *Recommendation Hypertext Markup Language – HTML 4.0.* W3C Recommendation, 1998 (http://www.w3.org/TR/REC–html40)

44. Bush, V.: "As We May Think". The Atlantic Monthly, July 1945 (verfügbar über http://www.isg.sfu.ca/~duchier/misc/vbush/)

45. Berners-Lee, T.: "Information Management: A Proposal". CERN, March 1989, May 1990 (verfügbar über http://www.w3.org/History/1989/proposal.html)

46. Berners-Lee, T.; Fielding, R.; Frystyk H.: *Hypertext Transfer Protocol – HTTP/1.0.* RFC 1945, Internet Engineering Task Force (IETF), 1996

47. Fielding, R.; Gettys, J.; Mogul, J.; Frystyk H.; Masinter, L.; Leach, P.; Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2068, Internet Engineering Task Force (IETF), 1998

48. Berners-Lee, T., Fielding, R.T. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, Internet Engineering Task Force (IETF), August 1998.

49. Orfali, R.; Harkey, D.: *Client/Server Programming with Java and Corba.* New York u.a.: John Wiley, 1997

50. Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.: "Extensible Markup Language (XML) 1.0". W3C Recommendation, Feb. 1998

51. N.N.: *BizTalk Framework 2.0 Draft: Document and Message Specification.* Microsoft Corporation, June 2000

52. Box, D.; Ehnebuske, D.; Kakivaya, G.; Layman, A.; Mendelsohn, N.; Nielsen, H.F.; Thatte, S.; Winer, D.: *Simple Object Access Protocol (SOAP) 1.1.* W3C Note, May 2000 (verfügbar über http://www.w3.org/TR/SOAP/)

53. Flanagan, D.: *Java in a Nutshell – 2nd Edition.* Bonn: O'Reilly/International Thomson Verlag, 1997

54. Cornell, G.; Horstmann, C. S.: *Core Java – Second Edition.* Mountain View CA: SunSoft Press, 1997

55. Summers, R.C.: *Secure Computing.* New York u.a.: Mc Graw-Hill, 1997

56. Raepple, M.: *Sicherheitskonzepte für das Internet.* Heidelberg: dpunkt-Verlag, 1998

57. Stallings, W.: *Protect your Privacy: The PGP user's Guide by William Stallings.* New Jersey: Prentice Hall, 1995

58. Singh, N.: "Unifying Heterogeneous Information Models". In: *Communications of the ACM.* Vol. 41 (1998) 5, p. 37–44

59. N.N.: "PROFIBUS Technische Kurzbeschreibung". PROFIBUS Nutzerorganisation, 1999

60. N.N.: "PROFIBUS Extensions to EN 50170 (DPV1). Version 2.0". PROFIBUS Nutzerorganisation, 1998

61. N.N.: "Open Integration – vertikale Integration von IT und Feldtechnik". PROFIBUS-Nutzer-Organisation, Version 1.1, 1999

62. Bruns, H.; Hempen, U.; Ott, W.; Vahldieck, R.: "Ein Konzept für eine COM/DCOM-basierte herstellerunabhängige Integration von Feldgeräten in Engineeringsysteme". In: *atp – Automatisierungstechnische Praxis* 41 (1999) 10, S. 34–40

63. Polzer, K.: "Einheitliches Bedienen, Beobachten und Parametrieren von Prozeßgeräten". In: *atp – Automatisierungstechnische Praxis* 40 (1998) 3, S. 57–64

64. Augustin, M; Polzer, K.; Ott, W.: "Electronic Device Description Language – Basis für eine einheitliche und plattformunabhängige Gerätebedienung". In: *atp – Automatisierungstechnische Praxis* 41 (1999) 10, S. 24–32

65. N.N.: "GSD-Spezifikation für PROFIBUS-FMS. Version 1.0, Juli 1998". PROFIBUS Nutzerorganisation, 1998

66. N.N.: "Electronic Device Description Language. Version 1.0, September 1999". PROFIBUS Nutzerorganisation, 1999

67. N.N.: "InterOperable Systems Project. Fieldbus Specification. Device Description Language". ISP Foundation, 1993

68. N.N.: *Highway Adressable Remote Transducer (HART).* Informationen verfügbar über http://www.hartcomm.org

69. N.N.: "FDT-Spezifikation Version 98–1". Gemeinschaftsarbeitskreis ZVEI-Leitsysteme und PROFIBUS Nutzerorganisation Gerätebeschreibung, Frankfurt und Karlsruhe, 1999

70. OPC Task Force: *OLE for Process Control – OPC Overview*. Version 1.0. 1998

71. OPC Task Force: *OPC Common Definitions and Interfaces*. Version 1.0. 1998

72. OPC Task Force: *Data Access Custom Interface Standard*. Version 2.03. 1999

73. OPC Task Force: *Data Access Automation Interface Standard*. Version 2.02. 1999

74. Montenegro, S.: "Sicherheit und die Mensch-Maschine-Kooperation". In: *Elektronik*, (1998) H. 15, S. 88 ff

75. N.N.: "PROFIBUS–Profil für Drehzahlveränderliche Antriebe, PROFIDRIVE". PROFIBUS Nutzerorganisation, 1997

76. Stieler, S.: "Einheitliche Anzeige-Bedienoberfläche für digitale Feldgeräte: Was tut sich bei Herstellern und Feldbusorganisationen?". In: *atp – Automatisierungstechnische Praxis* 40 (1998) 3, S. 65–66

77. N.N.: "PROFIBUS–Profil für Roboter und Numerische Steuerungen". PROFIBUS Nutzerorganisation, 1996

78. Feldmann, K.; Stöckel, T.: "Einsatz von Java–Anwendungen zum Bedienen und Beobachten". In: *Tagungsband zur Echtzeit und Inet'97* (Wiesbaden 1997). Feldkirchen: Franzis–Verlag, 1997, S. 199–205

79. Feldmann, K.; Stöckel, T.: "Utilisation of Java-applets for Building Device-Specific Man-Machine Interfaces". In: *Conference Proceedings FieldComms UK 1997.* Hinckley, UK1997

80. Feldmann, K.; Stöckel, T.: "A Jini-based Framework for Accessing Process Level Data in Manufacturing Systems". In: *Production Engineering* Vol. VII/1 (2000), Berlin 2000

81. Waldo, J.: *Jini Architecture Overview*. Sun Microsystems, Palo Alto 1998

82. N.N.: *Jini Architecture Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

83. N.N.: *Jini Discovery and Join Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

84. N.N.: *Jini Discovery Utilities Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

85. N.N.: *Jini Lookup Service Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

86. N.N.: *Jini Lookup Attribute Schema Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

87. N.N.: *Jini Distributed Event Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

88. N.N.: *Jini Distributed Leasing Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

89. N.N.: *Jini Entry Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

90. N.N.: *Jini Entry Utilities Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

91. N.N.: *Jini Spaces Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

92. N.N.: *Jini Transaction Specification*. Revision 1.0. Sun Microsystems, Palo Alto 1999

93. N.N.: *Object Bridge COM/CORBA Enterprise Client – Users Guide*. Version 1.1. Visual Edge, Quèbec, 1998

94. Object Management Group: *minimumCORBA.* April 1998, OMG TC Document orbos/98-08-04, Object Management Group (http://www.omg.org)

95. Object Management Group: *Real-Time CORBA.* March 1999, OMG TC Document orbos/99-02-12, Object Management Group (http://www.omg.org)

96. Object Management Group: *The Common Object Request Broker: Architecture and Specification.* Revision 2.1, August 1997, Document 97-08, Object Management Group (http://www.omg.org)

97. N.N.: *SimaticNet – FMS Programmierschnittstelle, Ausgabe 5.* Siemens AG, 1996

98. Ortner, E.: "Repository Systems. Teil 1: Mehrstufigkeit und Entwicklungsumgebung". In: *Informatik Spektrum.* Bd. 22, Heft 4, Aug. 1999. S. 235–251

99. Object Management Group: *Trading Object Service, Version 1.0.* December 1998, OMG Document formal/97-12-23 bzw. formal/2000-06-27, Object Management Group (http://www.omg.org)

100. Biron, P. V.; Malhotra, A.: "XML Schema Part 2: Datatypes". W3C Working Draft, April 2000 (verfügbar über http://www.w3.org/TR/xmlschema–2/)

101. Lassila, O.; Swick, R.R.: "Resource Description Framework (RDF) Model and Syntax Specification". W3C Recommendation, Feb. 1999 (verfügbar über http://www.w3.org/TR/REC–rdf–syntax/)

102. Brickley, D.; Guha, R.V.: "Resource Description Framework (RDF) Schema Specification 1.0". W3C Candidate Recommendation, March 2000 (verfügbar über http://www.w3.org/TR/rdf–schema/)

103. Nijssen, G.M.; Halpin, T.: *Conceptual Schema and Relational Database Design*. Sydney: Prentice Hall,1989

104. Lenat, D. B.: "Cyc: A Large–Scale Investment in Knowledge Infrastructure." *Communications of the ACM,* Vol. 38, Nr. 11, Nov. 1995

105. N.N.: *Knowledge Interchange Format (KIF)*. Draft proposed American National Standard (NCITS.T2/98–004), 1998 (verfügbar über: http://logic.stanford.edu/kif/dpans.html)

106. Lorenzen, P.: *Lehrbuch der konstruktiven Wissenschaftstheorie.* Mannheim: B.I.-Wissenschaftsverlag, 1987

Lebenslauf

Thomas Stöckel
geb. am 17.02.1971 in Erlangen

1977 – 1981	Grundschule Bruck-West
1981 – 1990	Emmy-Noether-Gymnasium in Erlangen
10/90 – 02/96	Studium der Informatik an der Friedrich-Alexander-Universität Erlangen-Nürnberg Abschluß: Diplom-Informatiker (Univ.)
02/96 – 12/00	Wissenschaftlicher Assistent am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Friedrich-Alexander-Universität Erlangen-Nürnberg

Reihe Fertigungstechnik Erlangen

Band 1
Andreas Hemberger
Innovationspotentiale in der rechnerintegrierten Produktion durch wissensbasierte Systeme
208 Seiten, 107 Bilder. 1988. Kartoniert.

Band 2
Detlef Classe
Beitrag zur Steigerung der Flexibilität automatisierter Montage–systeme durch Sensorintegration und erweiterte Steuerungskonzepte
194 Seiten, 70 Bilder. 1988. Kartoniert.

Band 3
Friedrich–Wilhelm Nolting
Projektierung von Montagesystemen
201 Seiten, 107 Bilder, 1 Tabelle. 1989.
Kartoniert.

Band 4
Karsten Schlüter
Nutzungsgradsteigerung von Montagesystemen durch den Einsatz der Simulationstechnik
177 Seiten, 97 Bilder. 1989. Kartoniert.

Band 5
Shir–Kuan Lin
Aufbau von Modellen zur Lageregelung von Industrierobotern
168 Seiten, 46 Bilder. 1989. Kartoniert.

Band 6
Rudolf Nuss
Untersuchungen zur Bearbeitungsqualität im Fertigungssystem Laserstrahlschneiden
206 Seiten, 115 Bilder, 6 Tabellen. 1989. Kartoniert.

Band 7
Wolfgang Scholz
Modell zur datenbankgestützten Planung automatisierter Montageanlagen
194 Seiten, 89 Bilder. 1989. Kartoniert.

Band 8
Hans–Jürgen Wißmeier
Beitrag zur Beurteilung des Bruchverhaltens von Hartmetall–Fließpreßmatrizen
179 Seiten, 99 Bilder, 9 Tabellen. 1989. Kartoniert.

Band 9
Rainer Eisele
Konzeption und Wirtschaftlichkeit von Planungssystemen in der Produktion
183 Seiten, 86 Bilder. 1990. Kartoniert.

Band 10
Rolf Pfeiffer
Technologisch orientierte Montageplanung am Beispiel der Schraubtechnik
216 Seiten, 102 Bilder, 16 Tabellen. 1990. Kartoniert.

Band 11
Herbert Fischer
Verteilte Planungssysteme zur Flexibilitätssteigerung der rechnerintegrierten Teilefertigung
201 Seiten, 82 Bilder. 1990. Kartoniert.

Band 12
Gerhard Kleineidam
CAD/CAP: Rechnergestützte Montagefeinplanung
203 Seiten, 107 Bilder. 1990. Kartoniert.

Band 13
Frank Vollertsen
Pulvermetallurgische Verarbeitung eines übereutektoiden verschleißfesten Stahls
XIII + 217 Seiten, 67 Bilder, 34 Tabellen. 1990. Kartoniert.

Band 14
Stephan Biermann
Untersuchungen zur Anlagen– und Prozeßdiagnostik für das Schneiden mit CO_2–Hochleistungslasern
VIII + 170 Seiten, 93 Bilder, 4 Tabellen. 1991. Kartoniert.

Band 15
Uwe Geißler
Material– und Datenfluß in einer flexiblen Blechbearbeitungszelle
124 Seiten, 41 Bilder, 7 Tabellen. 1991. Kartoniert.

Band 16
Frank Oswald Hake
Entwicklung eines rechnergestützten Diagnosesystems für automatisierte Montagezellen
XIV + 166 Seiten, 77 Bilder. 1991. Kartoniert.

Band 17
Herbert Reichel
Optimierung der Werkzeugbereitstellung durch rechnergestützte Arbeitsfolgenbestimmung
198 Seiten, 73 Bilder, 2 Tabellen. 1991. Kartoniert.

Band 18
Josef Scheller
Modellierung und Einsatz von Softwaresystemen für rechnergeführte Montagezellen
198 Seiten, 65 Bilder. 1991. Kartoniert.

Band 19
Arnold vom Ende
Untersuchungen zum Biegeumformen mit elastischer Matrize
166 Seiten, 55 Bilder, 13 Tabellen. 1991. Kartoniert.

Band 20
Joachim Schmid
Beitrag zum automatisierten Bearbeiten von Keramikguß mit Industrierobotern
XIV + 176 Seiten, 111 Bilder, 6 Tabellen. 1991. Kartoniert.

Band 21
Egon Sommer
Multiprozessorsteuerung für kooperierende Industrieroboter in Montagezellen
188 Seiten, 102 Bilder. 1991. Kartoniert.

Band 22
Georg Geyer
Entwicklung problemspezifischer Verfahrensketten in der Montage
192 Seiten, 112 Bilder. 1991. Kartoniert.

Band 23
Rainer Flohr
Beitrag zur optimalen Verbindungstechnik in der Oberflächenmontage (SMT)
186 Seiten, 79 Bilder. 1991. Kartoniert.

Band 24
Alfons Rief
Untersuchungen zur Verfahrensfolge Laserstrahlschneiden und –schweißen in der Rohkarosseriefertigung
VI + 145 Seiten, 58 Bilder, 5 Tabellen. 1991. Kartoniert.

Band 25
Christoph Thim
Rechnerunterstützte Optimierung von Materialflußstrukturen in der Elektronikmontage durch Simulation
188 Seiten, 74 Bilder. 1992. Kartoniert.

Band 26
Roland Müller
CO_2-Laserstrahlschneiden von kurzglasverstärkten Verbundwerkstoffen
141 Seiten, 107 Bilder, 4 Tabellen. 1992. Kartoniert.

Band 27
Günther Schäfer
Integrierte Informationsverarbeitung bei der Montageplanung
195 Seiten, 76 Bilder. 1992. Kartoniert.

Band 28
Martin Hoffmann
Entwicklung einer CAD/CAM–Prozeßkette für die Herstellung von Blechbiegeteilen
149 Seiten, 89 Bilder. 1992. Kartoniert.

Band 29
Peter Hoffmann
Verfahrensfolge Laserstrahlschneiden und –schweißen : Prozeßführung und Systemtechnik in der 3D–Laserstrahlbearbeitung von Blechformteilen
186 Seiten, 92 Bilder, 10 Tabellen. 1992. Kartoniert.

Band 30
Olaf Schrödel
Flexible Werkstattsteuerung mit objektorientierten Softwarestrukturen
180 Seiten, 84 Bilder. 1992. Kartoniert.

Band 31
Hubert Reinisch
Planungs– und Steuerungswerkzeuge zur impliziten Geräteprogrammierung in Roboterzellen
XI + 212 Seiten, 112 Bilder. 1992. Kartoniert.

Band 32
Brigitte Bärnreuther
Ein Beitrag zur Bewertung des Kommunikationsverhaltens von Automatisierungsgeräten in flexiblen Produktionszellen
XI + 179 Seiten, 71 Bilder. 1992. Kartoniert.

Band 33
Joachim Hutfless
Laserstrahlregelung und Optikdiagnostik in der Strahlführung einer CO_2–Hochleistungslaseranlage
175 Seiten, 70 Bilder, 17 Tabellen. 1993. Kartoniert.

Band 34
Uwe Günzel
Entwicklung und Einsatz eines Simulationsverfahrens für operative und strategische Probleme der Produktionsplanung und –steuerung
XIV + 170 Seiten, 66 Bilder, 5 Tabellen. 1993. Kartoniert.

Band 35
Bertram Ehmann
Operatives Fertigungscontrolling durch Optimierung auftragsbezogener Bearbeitungsabläufe in der Elektronikfertigung
XV + 167 Seiten, 114 Bilder. 1993. Kartoniert.

Band 36
Harald Kolléra
Entwicklung eines benutzerorientierten Werkstattprogrammiersystems für das Laserstrahlschneiden
129 Seiten, 66 Bilder, 1 Tabelle. 1993. Kartoniert.

Band 37
Stephanie Abels
Modellierung und Optimierung von Montageanlagen in einem integrierten Simulationssystem
188 Seiten, 88 Bilder. 1993. Kartoniert.

Band 38
Robert Schmidt–Hebbel
Laserstrahlbohren durchflußbestimmender Durchgangslöcher
145 Seiten, 63 Bilder, 11 Tabellen. 1993. Kartoniert.

Band 39
Norbert Lutz
Oberflächenfeinbearbeitung keramischer Werkstoffe mit XeCl–Excimerlaserstrahlung
187 Seiten, 98 Bilder, 29 Tabellen. 1994. Kartoniert.

Band 40
Konrad Grampp
Rechnerunterstützung bei Test und Schulung an Steuerungssoftware von SMD–Bestücklinien
178 Seiten, 88 Bilder. 1995. Kartoniert.

Band 41
Martin Koch
Wissensbasierte Unterstützung der Angebotsbearbeitung in der Investitionsgüterindustrie
169 Seiten, 68 Bilder. 1995. Kartoniert.

Band 42
Armin Gropp
Anlagen– und Prozeßdiagnostik beim Schneiden mit einem gepulsten Nd:YAG–Laser
160 Seiten, 88 Bilder, 7 Tabellen. 1995. Kartoniert.

Band 43
Werner Heckel
Optische 3D–Konturerfassung und on–line Biegewinkelmessung mit dem Lichtschnittverfahren
149 Seiten, 43 Bilder, 11 Tabellen. 1995. Kartoniert.

Band 44
Armin Rothhaupt
Modulares Planungssystem zur Optimierung der Elektronikfertigung
180 Seiten, 101 Bilder. 1995. Kartoniert.

Band 45
Bernd Zöllner
Adaptive Diagnose in der Elektronikproduktion
195 Seiten, 74 Bilder, 3 Tabellen. 1995. Kartoniert.

Band 46
Bodo Vormann
Beitrag zur automatisierten Handhabungsplanung komplexer Blechbiegeteile
126 Seiten, 89 Bilder, 3 Tabellen. 1995. Kartoniert.

Band 47
Peter Schnepf
Zielkostenorientierte Montageplanung
144 Seiten, 75 Bilder. 1995. Kartoniert.

Band 48
Rainer Klotzbücher
Konzept zur rechnerintegrierten Materialversorgung in flexiblen Fertigungssystemen
156 Seiten, 62 Bilder. 1995. Kartoniert.

Band 49
Wolfgang Greska
Wissensbasierte Analyse und Klassifizierung von Blechteilen
144 Seiten, 96 Bilder. 1995. Kartoniert.

Band 50
Jörg Franke
Integrierte Entwicklung neuer Produkt– und Produktionstechnologien für räumliche spritzgegossene Schaltungsträger (3–D MID)
196 Seiten, 86 Bilder, 4 Tabellen. 1995. Kartoniert.

Band 51
Franz–Josef Zeller
Sensorplanung und schnelle Sensorregelung für Industrieroboter
190 Seiten, 102 Bilder, 9 Tabellen. 1995. Kartoniert.

Band 52
Michael Solvie
Zeitbehandlung und Multimedia–Unterstützung in Feldkommunikationssystemen
200 Seiten, 87 Bilder, 35 Tabellen. 1996. Kartoniert.

Band 53
Robert Hopperdietzel
Reengineering in der Elektro– und Elektronikindustrie
180 Seiten, 109 Bilder, 1 Tabelle. 1996. Kartoniert.

Band 54
Thomas Rebhan
Beitrag zur Mikromaterialbearbeitung mit Excimerlasern – Systemkomponenten und Verfahrensoptimierungen
148 Seiten, 61 Bilder, 10 Tabellen. 1996. Kartoniert.

Band 55
Henning Hanebuth
Laserstrahlhartlöten mit Zweistrahltechnik
157 Seiten, 58 Bilder, 11 Tabellen. 1996. Kartoniert.

Band 56
Uwe Schönherr
Steuerung und Sensordatenintegration für flexible Fertigungszellen mit kooperierenden Robotern
188 Seiten, 116 Bilder, 3 Tabellen. 1996. Kartoniert.

Band 57
Stefan Holzer
Berührungslose Formgebung mit Laserstrahlung
162 Seiten, 69 Bilder, 11 Tabellen. 1996. Kartoniert.

Band 58
Markus Schultz
Fertigungsqualität beim 3D–Laserstrahlschweißen von Blechformteilen
165 Seiten, 88 Bilder, 9 Tabellen. 1997. Kartoniert.

Band 59
Thomas Krebs
Integration elektromechanischer CA–Anwendungen über einem STEP–Produktmodell
198 Seiten, 58 Bilder, 8 Tabellen. 1996. Kartoniert.

Band 60
Jürgen Sturm
Prozeßintegrierte Qualitätssicherung in der Elektronikproduktion
167 Seiten, 112 Bilder, 5 Tabellen. 1997. Kartoniert.

Band 61
Andreas Brand
Prozesse und Systeme zur Bestückung räumlicher elektronischer Baugruppen (3–D MID)
182 Seiten, 100 Bilder. 1997. Kartoniert

Band 62
Michael Kauf
Regelung der Laserstrahlleistung und der Fokusparameter einer CO_2–Hochleistungslaseranlage
140 Seiten, 70 Bilder, 5 Tabellen. 1997. Kartoniert

Band 63
Peter Steinwasser
Modulares Informationsmanagement in der integrierten Produkt– und Prozeßplanung
190 Seiten, 87 Bilder. 1997. Kartoniert.

Band 64
Georg Liedl
Integriertes Automatisierungskonzept für den flexiblen Materialfluß in der Elektronikproduktion
196 Seiten, 96 Bilder, 3 Tabellen. 1997. Kartoniert.

Band 65
Andreas Otto
Transiente Prozesse beim Laserstrahlschweißen
132 Seiten, 62 Bilder, 1 Tabelle. 1997. Kartoniert

Band 66
Wolfgang Blöchl
Erweiterte Informationsbereitstellung an offenen CNC–Steuerungen zur Prozeß– und Programmoptimierung
168 Seiten, 96 Bilder. 1997. Kartoniert.

Band 67
Klaus–Uwe Wolf
Verbesserte Prozeßführung und Prozeßplanung zur Leistungs– und Qualitätssteigerung beim Spulenwickeln.
186 Seiten, 125 Bilder. 1997. Kartoniert.

Band 68
Frank Backes
Technologieorientierte Bahnplanung für die 3D–Laserstrahlbearbeitung
138 Seiten, 71 Bilder, 2 Tabellen. 1997. Kartoniert.

Band 69
Jürgen Kraus
Laserstrahlumformen von Profilen
137 Seiten, 72 Bilder, 8 Tabellen. 1997. Kartoniert.

Band 70
Norbert Neubauer
Adaptive Strahlführungen für CO_2–Laseranlagen
120 Seiten, 50 Bilder, 3 Tabellen. 1997. Kartoniert.

Band 71
Michael Steber
Prozeßoptimierter Betrieb flexibler Schraubstationen in der automatisierten Montage
168 Seiten, 78 Bilder, 3 Tabellen. 1997. Kartoniert.

Band 72
Pfestorf, Markus
Funktionale 3D–Oberflächenkenngrößen in der Umformtechnik
162 Seiten, 84 Bilder, 15 Tabellen. 1997. Kartoniert.

Band 73
Volker Franke
Integrierte Planung und Konstruktion von Werkzeugen für die Biegebearbeitung
143 Seiten, 81 Bilder. 1998. Kartoniert.

Band 74
Herbert Scheller
Automatisierte Demontagesysteme und recyclinggerechte Produktgestaltung elektronischer Baugruppen
184 Seiten, 104 Bilder, 17 Tabellen. 1998. Kartoniert.

Band 75
Arthur Meßner
Kaltmassivumformung metallischer Kleinstteile
– Werkstoffverhalten, Wirkflächenreibung, Prozeßauslegung –
164 Seiten, 92 Bilder, 14 Tabellen. 1998. Kartoniert.

Band 76
Mathias Glasmacher
Prozeß– und Systemtechnik zum Laserstrahl–Mikroschweißen
184 Seiten, 104 Bilder, 12 Tabellen. 1998. Kartoniert.

Band 77
Michael Schwind
Zerstörungsfreie Ermittlung mechanischer Eigenschaften von Feinblechen mit dem Wirbelstromverfahren
124 Seiten, 68 Bilder, 8 Tabellen. 1998. Kartoniert.

Band 78
Manfred Gerhard
Qualitätssteigerung in der Elektronikproduktion durch Optimierung der Prozeßführung beim Löten komplexer Baugruppen
179 Seiten, 113 Bilder, 7 Tabellen. 1998. Kartoniert.

Band 79
Elke Rauh
Methodische Einbindung der Simulation in die betrieblichen Planungs– und Entscheidungsabläufe
192 Seiten, 114 Bilder, 4 Tabellen. 1998. Kartoniert.

Band 80
Niederkorn Sorin
Meßeinrichtung zur Untersuchung der Wirkflächenreibung bei umformtechnischen Prozessen
99 Seiten, 46 Bilder, 6 Tabellen. 1998. Kartoniert.

Band 81
Stefan Schuberth
Regelung der Fokuslage beim Schweißen mit CO_2–Hochleistungslasern unter Einsatz von adaptiven Optiken
140 Seiten, 64 Bilder, 3 Tabellen. 1998. Kartoniert.

Band 82
Armando Walter Colombo
Development and Implementation of Hierarchical Control Structures of Flexible Production System Using High–Level Petri Nets
216 Seiten, 86 Bilder. 1998. Kartoniert.

Band 83
Otto Meedt
Effizienzsteigerung bei Demontage und Recycling durch flexible Demontagetechnologien und optimierte Produktgestaltung
186 Seiten, 103 Bilder. 1998. Kartoniert.

Band 84
Knuth Götz
Modelle und effiziente Modellbildung zur Qualitätssicherung in der Elektronikproduktion
212 Seiten, 129 Bilder, 24 Tabellen. 1998. Kartoniert.

Band 85
Ralf Luchs
Einsatzmöglichkeiten leitender Klebstoffe zur zuverlässigen Kontaktierung elektronischer Bauelemente in der SMT
176 Seiten, 126 Bilder, 30 Tabellen. 1998. Kartoniert.

Band 86
Frank Pöhlau
Entscheidungsgrundlagen zur Einführung räumlicher spritzgegossener Schaltungsträger (3–D MID)
144 Seiten, 99 Bilder. 1999. Kartoniert.

Band 87
Roland Kals
Fundamentals of the miniaturization of sheet metal working processes
128 Seiten, 58 Bilder, 11 Tabellen. 1999. Kartoniert.

Band 88
Gerhard Luhn
Implizites Wissen und technisches Handeln am Beispiel der Elektronikproduktion.
253 Seiten, 61 Bilder, 1 Tabelle. 1999. Kartoniert.

Band 89
Axel Sprenger
Adaptives Streckbiegen von Aluminium–Strangpreßprofilen
114 Seiten, 63 Bilder, 4 Tabellen. 1999. Kartoniert.

Band 90
Hans–Jörg Pucher
Untersuchungen zur Prozeßfolge Umformen, Bestücken und Laserstrahllöten von Mikrokontakten
158 Seiten, 69 Bilder, 9 Tabellen. 1999. Kartoniert.

Band 91
Horst Arnet
Profilbiegen mit kinematischer Gestalterzeugung
128 Seiten, 67 Bilder, 7 Tabellen. 1999. Kartoniert.

Band 92
Doris Schubart
Prozeßmodellierung und Technologieentwicklung beim Abtragen mit CO_2-Laserstrahlung
133 Seiten, 57 Bilder, 13 Tabellen. 1999. Kartoniert.

Band 93
Adrianus L. P. Coremans
Laserstrahlsintern von Metallpulver – Prozeßmodellierung, Systemtechnik, Eigenschaften laserstrahlgesinterter Metallkörper
184 Seiten, 108 Bilder, 12 Tabellen. 1999. Kartoniert.

Band 94
Hans-Martin Biehler
Optimierungskonzepte für Qualitätsdatenverarbeitung und Informationsbereitstellung in der Elektronikfertigung
199 Seiten, 105 Bilder. 1999. Kartoniert.

Band 95
Wolfgang Becker
Oberflächenausbildung und tribologische Eigenschaften Excimerlaserstrahlbearbeiteter Hochleistungskeramiken
175 Seiten, 71 Bilder, 3 Tabellen. 1999. Kartoniert.

Band 96
Philipp Hein
Innenhochdruck-Umformen von Blechpaaren: Modellierung, Prozeßauslegung und Prozeßführung
129 Seiten, 57 Bilder, 7 Tabellen. 1999. Kartoniert.

Band 97
Gunter Beitinger
Herstellungs- und Prüfverfahren für thermoplastische Schaltungsträger
176 Seiten, 92 Bilder, 20 Tabellen. 1999. Kartoniert.

Band 98
Jürgen Knoblach
Beitrag zur rechnerunterstützten verursachungsgerechten Angebotskalkulation von Blechteilen mit Hilfe wissensbasierter Methoden
156 Seiten, 53 Bilder, 26 Tabellen. 1999. Kartoniert.

Band 99
Frank Breitenbach
Bildverarbeitungssystem zur Erfassung der Anschlußgeometrie elektronischer SMT-Bauelemente
168 Seiten, 92 Bilder, 12 Tabellen. 2000. Kartoniert.

Band 100
Bernd Falk
Simulationsbasierte Lebensdauervorhersage für Werkzeuge der Kaltmassivumformung
134 Seiten, 44 Bilder, 15 Tabellen. 2000. Kartoniert.

Band 101
Wolfgang Schlögl
Integriertes Simulationsdaten-Management für Maschinenentwicklung und Anlagenplanung
157 Seiten, 101 Bilder, 20 Tabellen. 2000. Kartoniert.

Band 102
Christian Hinsel
Ermüdungsbruchversagen hartstoffbeschichteter Werkzeugstähle in der Kaltmassivumformung
130 Seiten, 80 Bilder, 14 Tabellen. 2000. Kartoniert.

Band 103
Stefan Bobbert
Simulationsgestützte Prozessauslegung für das Innenhochdruck-Umformen von Blechpaaren
123 Seiten, 77 Bilder. 2000. Kartoniert.

Band 104
Harald Rottbauer
Modulares Planungswerkzeug zum Produktionsmanagement in der Elektronikproduktion
176 Seiten, 106 Bilder. 2000. Kartoniert.

Band 105
Thomas Hennige
Flexible Formgebung von Blechen durch Laserstrahlumformung
120 Seiten, 50 Bilder. 2001. Kartoniert.

Band 106
Thomas Menzel
Wissensbasierte Methoden für die rechnergestützte Charakterisierung und Bewertung innovativer Fertigungsprozesse
152 Seiten, 71 Bilder. 2001. Kartoniert.

Band 107
Thomas Stöckel
Kommunikationstechnische Integration der Prozeßebene in Produktionssysteme durch Middleware-Frameworks
162 Seiten, 65 Bilder, 5 Tabellen. 2001. Kartoniert.

Band 108
Frank Pitter
Verfügbarkeitssteigerung von Werkzeugmaschinen durch Einsatz mechatronischer Sensorlösungen
158 Seiten, 131 Bilder, 8 Tabellen. 2001. Kartoniert.

Band 109
Markus Korneli
Integration lokaler CAP-Systeme in einen globalen Fertigungsdatenverbund
125 Seiten, 53 Bilder, 11 Tabellen. 2001. Kartoniert.