IAS-Forschungsberichte

Forschungsbericht
Institut für Automatisierungs- und Softwaretechnik

Hrsg.: Prof. Dr.-Ing. Dr. h. c. P. Göhner

Nasser Jazdi

Universelle Fernservice-Infrastruktur für eingebettete Systeme

Band 3/2003

Universität Stuttgart

Universelle Fernservice-Infrastruktur für eingebettete Systeme

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

Vorgelegt von
Nasser Jazdi
aus Mashhad (Iran)

Hauptberichter:	Prof. Dr.-Ing. Dr. h. c. Peter Göhner
Mitberichter:	Univ.-Prof. Dr.-Ing. Georg Färber
Tag der Einreichung:	16.06.2003
Tag der mündlichen Prüfung:	30.07.2003

Institut für Automatisierungs- und Softwaretechnik
der Universität Stuttgart

2003

IAS-Forschungsberichte

Band 3/2003

Nasser Jazdi

Universelle Fernservice-Infrastruktur für eingebettete Systeme

D 93 (Diss. Universität Stuttgart)

Shaker Verlag
Aachen 2003

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

Zugl.: Stuttgart, Univ., Diss., 2003

ISBN 3-8322-1942-0
ISSN 1610-4781

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • eMail: info@shaker.de

Meinem Vater, Prof. Dr. M. Yazdi und meiner Frau, Karin Gäbel-Jazdi für ihre Unterstützung

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik (IAS) der Universität Stuttgart.

Zu allererst bedanke ich mich bei meinem Doktorvater und Leiter des Instituts, Herrn Prof. Dr.-Ing. Dr. h. c. Peter Göhner sehr herzlich für die Unterstützung, viele wertvolle Anregungen und kritische Diskussionen sowie die Übernahme des Hauptberichts.

Mein herzlicher Dank gilt auch Herrn Prof. Dr.-Ing. Georg Färber, Leiter des Lehrstuhls für Realzeit-Computersysteme der TU München, für das Interesse und die Übernahme des Mitberichts.

Nicht unerwähnt lassen möchte ich alle Kolleginnen und Kollegen bei denen ich mich für die gute Zusammenarbeit, die kollegiale Unterstützung und die konstruktiven Diskussionsbeiträge bedanke. Insbesondere Kirstin Barbey, Susanne Bock, Stephan Eberle, Dr. R. Gäbel und Jan Traumüller danke ich für die kritische Durchsicht des Manuskripts.

In diesem Zusammenhang möchte ich auch die Studierenden erwähnen, deren Diplom- und Studienarbeiten einen wesentlichen Anteil zum Gelingen der Arbeit beitrugen.

Schließlich möchte ich mich bei meiner Familie bedanken, die mir während meiner Arbeit sehr viel Kraft gab, meinem Vater, der das Interesse an der Wissenschaft in mir weckte und bis heute pflegt und meiner Frau, Karin für ihr grenzenloses Verständnis, ihre Unterstützung und Geduld während der Entstehung meiner Arbeit. Nicht vergessen möchte ich meine Kinder, Maral, Kian und Shirin, die in dieser Zeit ihren Vater oft entbehren mussten.

Stuttgart, im Juni 2003

Nasser Jazdi

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

aAk	**a**llgemeine **A**nbindungs**k**omponente
ACPLT/KS	**A**a**c**hener **P**rozess**l**eit**t**echnik / **K**ommunikations**s**ystem
API	**A**pplication **P**rogramming **I**nterface
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
CAL	**C**AN-**A**pplication-**L**ayer
CAN	**C**ontroller **A**rea **N**etwork
CiA	**C**AN-**i**n-**A**utomation
COB	**C**AN-**Ob**jekt
COM	**C**omponent **O**bject **M**odel
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture
Dak	**D**aten**a**ufbereitungs**k**omponente
DCOM	**D**istributed **C**omponent **O**bject **M**odel
DLL	**D**ynamic **L**ink **L**ibrary
DOM	**D**ocument **O**bject **M**odel
DTD	**D**ocument **T**ype **D**efinition
EMIT	**E**mbedded **M**icro **I**nternetworking **T**echnology
ES	**E**mbedded **S**ystem (Eingebettetes System)
FTP	**F**ile **T**ransfer **P**rotocol
FzS	**F**ern**z**ugriff-**S**erver
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper**T**ext **M**arkup **L**anguage
ID	**Id**entifier
IDA	**I**nterface for **D**istributed **A**utomation
IP	**I**nternet **P**rotocol
ISO	**I**nternational **S**tandardization **O**rganisation

IT	**I**nformation **T**echnology
JVM	**J**ava **V**irtual **M**achine
LAN	**L**ocal **A**rea **N**etwork
MVC	**M**odel-**V**iew-**C**ontrol
OECD	**O**rganization for **E**conomic **C**ooperation and **D**evelopment
OLE	**O**bject **L**inking and **E**mbedding
OMG	**O**bject **M**anagement **G**roup
OPC	**O**LE for **P**rocess **C**ontrol
ORB	**O**bject **R**equest **B**roker
OV	**O**bjekt**v**erzeichnis
PC	**P**ersonal **C**omputer
PDA	**P**ersonal **D**igital **A**ssistent
PDO	**P**rozess**d**aten**o**bjekte
RFC	**R**equest **f**or **C**omments
RMI	**R**emote **M**ethod **I**nvocation
RPC	**R**emote **P**rocedure **C**all
RWTH	**R**heinisch-**W**estfälische **T**echnische **H**ochschule
sAk	**s**pezielle **A**nbindungs**k**omponente
SDML	**S**ervice **D**escription **M**arkup **L**anguage
SDO	**S**ervice**d**aten**o**bjekte
SLIO	**S**erial **L**ink **I**nput **O**utput
TCP	**T**ransmission **C**ontrol **P**rotocol
TINI	**T**iny **In**ternet **I**nterface
URI	**U**niform **R**esource **I**dentifier
URL	**U**niform **R**esource **L**ocator
W3C	**W**orld **W**ide **W**eb **C**onsortium
WAN	**W**ide **A**rea **N**etwork
WAP	**W**ireless **A**pplication **P**rotocol

WML	**W**ireless **M**arkup **L**anguage
WWW	**W**orld **W**ide **W**eb
XML	e**X**tensible **M**arkup **L**anguage
XSL	e**X**tensible **S**tylesheet **L**anguage
XSLT	**XSL**-**T**ransformations
XSP	e**X**tensible **S**erver **P**ages

Begriffsverzeichnis

Aktor: Einheit zur Umsetzung von Stellinformation tragenden Signalen geringer Leistung in leistungsbehaftete Signale einer zur Prozessbeeinflussung notwendigen Energieform.

Client: Anwendung, die die Dienste eines Servers in Anspruch nimmt.

CAL: Flexibles, anwendungsunabhängiges Kommunikationsprotokoll, welches sich besonders zur Implementierung von CAN-basierenden Kommunikationssystemen eignet.

CAN: Ursprünglich für den Automobilsektor von Bosch und Intel entwickelter Feldbus.

CANopen: Basiert auf CAL und definiert Kommunikationsprofile und -verfahren für den Datenaustausch zwischen CANopen-Geräten. CANopen-Geräte sind Geräteprofilen zugeteilt und ermöglichen die herstellerunabhängige Konfiguration über den Bus.

CANopen Objekt: Die über den Bus sichtbare Funktionalität eines CANopen Gerätes wird durch Objekte beschrieben. CANopen-Objekte können Daten, Parameter oder Funktionen eines Gerätes sein. Über einen 16-Bit Index und einen 8-Bit Subindex kann das Objekt im Objektverzeichnis identifiziert werden.

COCOON: Java-Servlet für XML-basierte Web-Publishing.

CORBA: Eine Spezifikation für den Aufruf von Methoden in verteilten Systemen. Ähnlich zu RMI, allerdings unabhängig von der Plattform und der Programmiersprache.

Document Type Definition: Logische Struktur für eine Klasse von XML-Dokumenten, welche die zur Verfügung stehenden XML-Elemente, XML-Attribute und ihre zulässige Schachtelung festlegt.

Eingebettetes System: Mikrocontrollersysteme, die mit Hilfe von Aktoren und Sensoren mit der Umgebung interagieren.

Ethernet: Weit verbreitete Nahbereichsnetzwerk-Basisband-Spezifikation, mit welcher verschiedene Rechner innerhalb eines LANs vernetzt werden.

Fernservice-Applikation: Anwendungen zur Ferndiagnose, Fernvisualisierung, Fernwartung und Fernbedienung von eingebetteten Systemen werden unter dem Oberbegriff Fernservice-Applikationen zusammengefasst.

Fernzugriff-Server: Stellt die von einzelnen eingebetteten Systemen angebotenen Dienste des Clients zur Verfügung. Er übernimmt die Darstellung und Konvertierung der Prozessdaten.

Internet: Weltweiter Verbund aller Computernetzwerke, die das TCP/IP Protokoll verwenden. Die Netze sind untereinander durch Gateways verbunden.

Komponente: Vorgefertigte Software-Einheit, die bewusst für die Mehrfachverwendung ausgelegt wurde. Durch Auswahl, Konfiguration und Parametrierung entstehen aus einzelnen Komponenten vollständige Anwendungen.

RMI: Ermöglicht in Java den Aufruf von Methoden auf virtuellen Maschinen, die sich an beliebigen Stellen im Netz befinden.

Sensor: Einheit zur Umsetzung von physikalischen oder chemischen Größen in eine elektronische Form.

Server: Ein Computer in einem Netzwerk, der seine Leistungen und Daten dem Client mittels Client/Server-Software zur Verfügung stellt.

Servlet: Kleine Programme/Anwendungen, die in der Programmiersprache Java geschrieben sind und im Gegensatz zu Java-Applets nicht auf einem Client, sondern auf einem Server ausgeführt werden.

Socket: Ein eindeutiger Endpunkt einer Ethernet-Kommunikation, der mit Hilfe einer IP-Adresse und einer Portnummer beschrieben wird.

Tag: Definierte Folge von Zeichen, die den Beginn oder das Ende eines logischen Abschnitts oder eines logischen Elements in einem XML-Dokument repräsentieren.

TINI-Board: Von Dallas Semiconductor entwickeltes Java-fähiges Mikrocontrollerboard.

URI: Dient der eindeutigen Lokalisierung einer Ressource im Internet. Es gibt zwei Arten von URIs, Uniform Resource Locators (URLs) und Uniform Resource Names (URNs) [BFM97].

URL: Zeiger auf eine bestimmte Ressource im Internet an einer bestimmten Stelle, z. B. http://www.ias.uni-stuttgart.de.

WWW: Auf Hypertext basierendes Informations- und Quellensystem für das Internet. Das WWW ist der am schnellsten wachsende Teil des Internets.

Zusammenfassung

Globalisierung ist ein Vorgang, durch den Märkte und Produktionen in verschiedenen Ländern immer mehr kooperieren, aber auch voneinander abhängig werden. Ursache hierfür ist die Dynamik des Handels mit Gütern und Dienstleistungen sowie die Bewegungen von Kapital und Technologie. Die Differenzierung erfolgt nicht mehr nur durch das Produkt selbst, sondern zunehmend durch dazugehörige Dienstleistungen, wie Wartung und Software-Updates. Die globale Bereitstellung solcher Dienstleistungen ist jedoch ein existentielles Problem für kleine und mittelständische Unternehmen.

Das Internet als ein weltweiter Verbund von Computernetzwerken bietet eine globale Kommunikationsmöglichkeit für den Menschen. Computer mit Internet-Zugang werden zum festen Bestandteil jedes modernen Arbeitsplatzes, auch infolge der immer geringer werdenden Anschaffungskosten. Die mit dem Internet verbundenen Web-Technologien ermöglichen die weltweite, effektive und kostengünstige Bereitstellung von Dienstleistungen. Der Einsatz der Web-Technologien beim Erbringen von Dienstleistungen für Automatisierungsprodukte und -anlagen verlangt jedoch neue, an die spezifischen Randbedingungen der Automatisierungssysteme zugeschnittene Konzepte.

In der vorliegenden Arbeit wird eine flexible und erweiterbare Infrastruktur für den Einsatz von Web-Technologien in eingebetteten Systemen vorgestellt. Sie baut auf der bewährten Drei-Schichten-Architektur, bestehend aus dem eingebetteten System, dem universellen Fernzugriff-Server und dem Client, auf. Dabei wird eine systemübergreifende und allgemeingültige Schnittstelle für die Anbindung unterschiedlicher eingebetteter Systeme und den Zugriff auf deren Prozessdaten geschaffen. Das Verfahren ermöglicht außerdem ein flexibles Weiterverarbeiten der Gerätedaten, sodass sie für unterschiedliche Clients aufbereitet werden können. Um die Flexibilität - sowohl auf der Seite des eingebetteten Systems als auch beim Client - zu gewährleisten, wird eine neue, XML-basierte Beschreibungssprache (SDML) eingeführt. Die SDML-Dokumente beinhalten Informationen über angebundene eingebettete Systeme, abrufbare Gerätedaten und Präsentationsregeln für unterschiedliche Clients. Sie werden gerätespezifisch erstellt.

Das vorgeschlagene Verfahren ermöglicht die Anbindung unterschiedlicher eingebetteter Systeme an das Internet bei minimalem Hardware- und Softwareaufwand. Die einmalig entwickelten Software-Komponenten des Fernzugriff-Servers können für unterschiedliche Geräte verwendet werden und tragen damit zur Senkung der Diagnose- und Wartungskosten bei. Der Anwender kann einen gewöhnlichen Browser für die Kommunikation mit seinem Gerät verwenden und braucht somit keine zusätzliche Software auf seinem lokalen Rechner zu installieren.

Abstract

Globalization is a process which evokes dependence between markets and industrial production in different countries. The reason is the dynamic force of trade with its products and its services combined with the flow of capital and technologies all over the world. The differentiation of products does not occur only because of the product itself but increasingly through additional services such as maintenance and software updates. To master this process companies must be present globally. World wide presence therefore is a fundamental problem of small and medium-sized enterprises.

The Internet as a network of millions of computers makes global communication possible. The computer with an Internet access belongs to a modern work place. The Internet technologies enable us to offer services world wide in a very resourceful and cost effective manner.

To use these technologies in the field of industrial automation we need a new concept specified for the requirements of these systems. This thesis proposes a universal infrastructure-concept to enable remote service applications for embedded systems using standard Internet technologies.

An important aspect of this matter is the development of reusable components accomplishing a modular structure, according to the principle of building blocks. The concept is based on the Three-Tier-Architecture.

The tiers are denominated: embedded system, universal device proxy and client. During this work an interface independent of the system was developed for the connection to different embedded systems and the access to their device information.

For the specification of device information a new XML-based description language (SDML-Service Description Markup Language) is introduced, which allows the presentation of the same information to different clients. To support the developer of XML-documents a tool was developed, which generates proposed documents automatically.

The proposed concept enables the connection to different embedded systems via proxy device with a minimum of extra hardware and software. It is not necessary to change the components of the device. The same components developed for the proxy device could be reused for different embedded systems, so the overall cost will be reduced. By using SDML the same information may be presented to different clients and there is no need for the user to install any piece of software on his system. Only a standard browser is required.

1 Einleitung und Motivation

„Die Suche nach der Wahrheit ist einerseits schwer und andererseits leicht- denn es ist offensichtlich, dass sie keinem von uns völlig gelingt und keinem von uns völlig misslingt. Jeder von uns fügt unserem Wissen über die Natur ein wenig hinzu, und aus all den gesammelten Fakten entsteht eine gewisse Größe.“
(Aristoteles)

1.1 Einsatz von Web-Technologien in eingebetteten Systemen

Mikroelektronische Systeme gewinnen einen zunehmenden Einfluss in unserem Alltag. Eine besonders wichtige Klasse elektronischer Systeme sind die eingebetteten Systeme. Diese kommen in immer mehr Bereichen verstärkt zum Einsatz, wie z. B. in der Telekommunikation und Unterhaltungselektronik, Fahrzeugelektronik und Medizintechnik, bei Haushaltsgeräten, sowie in Fabriksteuerungen und Industrierobotern. Diese Einsatzbereiche lassen einen hohen Marktanteil erkennen, der stark im Wachstum begriffen ist und in den kommenden Jahren voraussichtlich den PC-Markt bezüglich Umsatzvolumen einholen wird.

Auf Grund der Globalisierung und dem damit einhergehenden Wegfall der nationalen Grenzen werden eingebettete Produkte weltweit verkauft. Dadurch gewinnen die globalen Dienstleistungen immer mehr an Bedeutung und werden zum wichtigsten Innovationsfaktor der Unternehmen. Dies bringt eine große Herausforderung mit sich, die nicht ohne Weiteres, insbesondere nicht von kleinen und mittelständischen Firmen, zu bewältigen ist.

Das Internet und die damit verbundenen Web-Technologien ermöglichen das Anbieten von weltweiten, effektiven und kostengünstigen Dienstleistungen. Von entfernten Orten aus können entweder Informationen über ein Gerät ermittelt (Ferndiagnose, Fernvisualisierung) oder Gerätefunktionen fernausgelöst werden (Fernwartung, Fernbedienung). Zur Abwicklung dieser Aufgaben können bewährte und verbreitete Werkzeuge (z. B. Webbrowser) genutzt werden, was die Gerätehandhabung entschieden einfacher und komfortabler gestaltet. Spezialgeräte und -anwendungen, teures Fachpersonal oder lange Einarbeitungs- und Reisezeiten sind nicht mehr erforderlich [EbGö01].

Für die Realisierung der Fernservice-Applikationen (Ferndiagnose, Fernvisualisierung, Fernwartung, Fernbedienung) muss das eingebettete System zwei Randbedingungen erfüllen: Es muss einerseits die Möglichkeit für den Zugriff auf Prozessdaten gegeben (z. B. durch ein integriertes Feldbussystem), andererseits muss ein Internet-Zugang vorhanden sein. Für die Realisierung solcher Fernservice-Applikationen ist es folglich notwendig, das Gerät um bestimmte

Hardware und Software zu erweitern. Der einfachste Weg ist der, einen PC mit dem eingebetteten System zu verbinden. Die Kommunikation erfolgt über einen Feldbus und der Zugang zum Internet wird, z. B. mittels einer Ethernet-Verbindung, ermöglicht. Als Software-Schnittstelle wird eine Anwendung benötigt, die diese Anforderungen realisiert, d. h. über den Feldbus die Prozessdaten aus dem Gerät holt, verarbeitet und sie anschließend im Internet zur Verfügung stellt. Der Anwender, der auf das eingebettete System zugreifen möchte, um Informationen abzufragen oder bestimmte Aktionen auf dem Gerät durchzuführen, benötigt natürlich eine speziell abgestimmte Software.

Es gibt derzeit einige Lösungen auf dem Markt, die ein eingebettetes System auf die soeben beschriebene Weise *Internet-fähig* machen können. Der PC wird in der Regel durch einen herstellerspezifischen eingebetteten PC oder gar eine Mikrocontroller-basierte, ebenfalls herstellerspezifische Gateway-Komponente ersetzt. Wie wird nun eine solche Lösung zusammen mit einem eingebetteten System eingesetzt? Auf der einen Seite muss das eingebettete System an die Hardware des Herstellers angebunden werden. Hierzu muss eine spezielle Anpassung der Schnittstelle zwischen Hardware und eingebettetem System erfolgen. Auf der anderen Seite muss der Anwender für die Kommunikation mit dem Gerät herstellerspezifische Software auf seinem Rechner installieren. Die Einarbeitung in die Bedienung der Software ist oft mit viel Aufwand verbunden. Es werden teilweise Spezialformate für die Übermittlung und die Speicherung der Prozessdaten verwendet, was eine Weiterverarbeitung durch andere Programme erschwert. Für die Darstellung der Daten kommen neben spezieller Software häufig auch individuell gestaltete Anzeige- und Bediengeräte zum Einsatz. Diese sind meist um ein Vielfaches teurer als Standard-Hardware.

Für die Anbindung eines neuen eingebetteten Systems müssen, falls dies überhaupt möglich ist, Hardware und Software mit großem Aufwand adaptiert werden. Ebenso schwierig gestaltet sich die Bereitstellung der Daten für andere Typen von Clients.

Die Konkurrenz unter den Anbietern für Fernservice-Applikationen ist relativ gering, da die Systeme nur komplett angeboten werden. Kleinere Firmen, die nur einen Teil, z. B. einen Client oder die Anbindung eines eingebetteten Systems vertreiben möchten, müssen sich an die Schnittstellen externer Anbieter halten. Eine Standardisierung ist nicht abzusehen.

1.2 Ziel der universellen Fernservice-Infrastruktur

Die Problematik beim Einsatz von Web-Technologien in eingebetteten Systemen bildet den Ausgangspunkt der vorliegenden Arbeit. Die zu entwickelnden Konzepte und Verfahren sollen den Einsatz effizienter und kostengünstiger gestalten. Im Folgenden werden die wichtigsten Aspekte herausgestellt.

Anwendbarkeit in der Praxis

Das oberste Ziel der Arbeit ist die Konzeption einer allgemein gültigen Infrastruktur zur Bereitstellung von Fernservice-Applikationen für eingebettete Systeme. Dabei werden Ferndiagnose und Fernvisualisierung bzw. Fernwartung und Fernbedienung berücksichtigt. Ein Fokus liegt auf der Allgemeingültigkeit und Erweiterbarkeit des Verfahrens. Diese Aspekte sollen die Bedürfnisse der industriellen Entwickler von Fernservice-Applikationen erfüllen und damit soll der Einsatz des Verfahrens in der Praxis erleichtert werden.

Entwicklung mehrfach verwendbarer Komponenten

Ein weiteres Ziel ist die Entwicklung mehrfach verwendbarer Komponenten, sodass ein modularer Aufbau möglich wird. Zukünftige Entwicklungszeiten sollen reduziert und Kosten gesenkt werden.

Aufbau auf Standard-Technologien

Bei der Entwicklung des neuen Verfahrens soll darauf geachtet werden, dass möglichst viele Standard-Technologien eingesetzt werden. Dieses Vorgehen dient einerseits der Robustheit des Verfahrens, da es auf bereits getestete und bewährte Technologien aufbaut und andererseits, auf Grund der Verbreitung dieser Standards, der Akzeptanz.

Keine zusätzliche Software-Installation beim Anwender

Die Aufbereitung der Daten für die Darstellung beim Anwender soll so erfolgen, dass der Anwender für die Präsentation keinerlei Zusatzsoftware benötigt. Die Darstellung soll im Browser, der Bestandteil jedes Rechners ist, erfolgen.

1.3 Sicherheit beim Zugriff auf Automatisierungssysteme

Mit Hilfe des Fernzugriffs auf Automatisierungssysteme können die Benutzer oder das Wartungspersonal aus der Ferne, z. B. über das Internet, mit dem Gerät kommunizieren. Das Internet ist allerdings ein unsicherer Übertragungskanal, da die gesendeten Daten nahezu jeder an die übertragenden Netzen angebundenen Partei zur Verfügung stehen, von ihr gelesen und manipuliert werden können. Dabei ist die Spanne an möglichen Bedrohungsszenarien groß, vom Eindringen in die Privatsphäre (beispielsweise durch Lesen vertraulicher Informationen) bis hin zur lebensgefährlichen Manipulation sicherheitsrelevanter Systeme (z. B. in Zügen, Flugzeugen oder Kernkraftwerken).

Der Trend zum Einsatz von Web-Technologien in der Automatisierungstechnik und zum globalen Zugang zu Daten macht Sicherheit zu einer wichtigen Herausforderung. Der Schutz von sicherheitskritischen Systemen und Infrastruktureinrichtungen wie Produktionsanlagen oder Versorgungsunternehmen, die einen Zugang zum Internet haben, gegen elektronische Angriffe

über Kommunikationsnetzwerke ist ein Thema von wachsender Bedeutung [Naed03]. Sicherheit[1] hat das Verhindern von Gefahren, die während des Betriebes von Menschen oder der Umwelt ausgehen, zum Ziel [LaGö99b]. Im IT-Bereich lässt sich Sicherheit dabei in folgende 3 Teilgebiete untergliedern:

Vertraulichkeit bedeutet, dass Informationen nur von den Personen gelesen werden können, für welche diese Informationen auch bestimmt sind. Eine Schutzmaßnahme gegen unbefugtes Lesen von Informationen ist die kryptographische Verschlüsselung. Ein Beispiel für die Verschlüsselung der Daten ist das **S**ecure **S**ocket **L**ayer-Verfahren (SSL). Bei diesem Verfahren wird der komplette Datenstrom zwischen zwei Computern in beide Richtungen verschlüsselt. Nur wer den richtigen Schlüssel kennt, kann die Datenpakete entschlüsseln.

Authentizität soll sicherstellen, dass eine Information nachweisbar von derjenigen Quelle stammt, aus der sie zu kommen vorgibt. Der Sender einer Information ist also eindeutig identifizierbar und niemand kann vorgeben, dieser Sender zu sein. Eine mögliche Maßnahme zur Zugriffskontrolle und Authentifizierung ist der Einbau einer Passwortabfrage. Dabei wird jedem Benutzer eine eindeutige *Benutzerkennung* und ein (selbst gewähltes) Passwort zugeteilt. Damit kann sich der Benutzer gegenüber dem Server legitimieren.

Integrität garantiert, dass eine übertragene Information nicht durch Dritte abgefangen und verändert worden ist.

Das Thema „Sicherheit im Internet“ hat zwei wesentliche Aspekte, zum einen die Sicherheit gegen Sabotage, also Sicherheit vor Programmen mit Schadensfunktionen, die über E-Mail, oder durch den Aufruf entsprechend präparierter WWW-Seiten unbemerkt auf den Rechner gelangen können, zum anderen die Sicherheit der Daten bzgl. unbefugtem Zugriff bzw. die Sicherheit gegen Ausspionieren innerhalb von Netzwerken [Schn00]. Um den Zugang der unbefugten Personen zu einem Netz zu verhindern kann u. a. eine Firewall eingesetzt werden. Die Firewall schränkt den Zugriff von außen auf das Netz, aber auch von innen aus dem Netz ein [MOV96].

Obwohl das Thema Sicherheit eine zentrale Rolle bei der Realisierung jeglicher webbasierter Lösungen in der Automatisierungstechnik spielt, war die Betrachtung dieses Aspektes kein Bestandteil der vorliegenden Arbeit. Der Grund hierfür ist, dass das Thema „Sicherheit in der Automatisierungstechnik“ die Hauptfragestellung einer weiteren Forschungsarbeit am Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart ist. Ziel dieser Arbeit ist durch Anwendung der o. g. Bereiche ein Konzept zur Steigerung der Sicherheit beim Fernzugriff auf Automatisierungssysteme zu entwickeln. Dazu wird neben der Klassifikation unterschiedlicher Arten von Automatisierungssystemen und der von ihnen übertragenen Informationen auch die

---

[1] engl. security, im Gegensatz zu engl. safety, welche sich mit der Verhinderung von Gefahren, die vom Prozessautomatisierungssystem ausgehen, beschäftigt:

Analyse der jeweils spezifischen Randbedingungen der entsprechenden Systeme sowie die Betrachtung möglicher Angriffspunkte durchgeführt. Aus den so gewonnenen Informationen werden Bewertungsmaßstäbe entwickelt sowie Verfahren ausgewählt und eingesetzt, um die zu übertragenden Prozessdaten und die Automatisierungsgeräte selbst angemessen zu schützen [Gutb03].

Im Rahmen der vorliegenden Arbeit wurden lediglich bei den Implementierungen bestehende Sicherheitstechnologien eingesetzt. Diese sind im Einzelnen:

- Passwortabfrage zur Zugriffskontrolle und Authentifizierung.
- **S**ecure **S**ocket **L**ayer-Verfahren (SSL) zur Verschlüsselung der sicherheitsrelevanten Daten.
- Firewall, um den Zugriff von außen auf das Netz einzuschränken.

1.4 Gliederung der Arbeit

In Kapitel 2 werden die für das Verständnis der Arbeit notwendigen Begriffe aus dem Gebiet der Web-Technologien in der Automatisierungstechnik eingeführt.

Kapitel 3 stellt die bestehenden Konzepte für die systemunabhängige Übertragung von Prozessdaten vor. Hierbei wird im Wesentlichen der Stand der Forschung auf diesem Gebiet wiedergegeben. Anschließend werden die Konzepte anhand festgelegter Kriterien bewertet.

In Kapitel 4 wird die Problematik beim Einsatz von Web-Technologien in eingebetteten Systemen und deren Ursachen behandelt. Daran anschließend werden die Anforderungen an einem Lösungsansatz abgeleitet.

Ausgehend von den im vorherigen Kapitel aufgezeigten Problemen, wird in Kapitel 5 das Konzept der universellen Fernservice-Infrastruktur basierend auf allgemeingültigen Schnittstellen eingeführt und erläutert. Abschnitt 5.3 stellt das Verfahren für den Einsatz von Web-Technologien in eingebetteten Systemen auf Basis einer universellen Infrastruktur vor. Dabei wird zunächst die Drei-Schichten-Architektur des Verfahrens erläutert und anschließend werden die Aufgaben der einzelnen Schichten diskutiert.

In Kapitel 6 wird die im Rahmen dieser Arbeit entwickelte Beschreibungssprache SDML vorgestellt, die zur Beschreibung von Geräteinformationen und zur Konfiguration des Fernzugriff-Servers eingesetzt wird. Dabei werden die wichtigsten Elemente dieser Spezifikationssprache eingeführt und diskutiert.

Kapitel 7 widmet sich dem funktionalen Aufbau der universellen Fernservice-Infrastruktur. Der Fernzugriff-Server wird hierbei detailliert beschrieben.

Um das vorgeschlagene Verfahren zu veranschaulichen werden in Kapitel 8 zwei Fallbeispiele herangezogen. Dabei werden neben eingebetteten Systemen der Fernzugriff-Server und unterschiedliche Clients vorgestellt. Weiterhin werden die Systemarchitektur und Systemkomponente beschrieben. Am Ende werden die gesammelten Erfahrungen und Ergebnisse vorgestellt und diskutiert.

Abschließend wird im Kapitel 9 das Konzept zusammengefasst und ein Ausblick auf mögliche Erweiterungen aufgezeigt.

2 Grundlagen

Zum besseren Verständnis der Arbeit werden im Folgenden die notwendigen Grundlagen vermittelt. Zunächst werden eingebettete Systeme und deren Eigenschaften vorgestellt. Dabei wird eine Unterscheidung zwischen zeit- und ereignisgesteuerten Systemen vorgenommen und diskutiert. Im weiteren Verlauf wird der Zugriff auf Prozessdaten erläutert und eine Einführung in die Web-Technologien gegeben. Hierzu wird zunächst kurz auf die Historie dieser Technologien eingegangen und anschließend ein Überblick über verschiedene Web-Technologien gegeben. Von diesen Technologien werden diejenigen, die im Rahmen dieser Arbeit von tragender Bedeutung sind, vorgestellt und deren Vor- und Nachteile diskutiert. Um eine einheitliche Sprache zu schaffen, werden schließlich einige Begriffsdefinitionen eingeführt.

2.1 Eingebettete Systeme

Als eingebettete Systeme (Embedded Systems) werden im Allgemeinen Elektronik-Systeme bezeichnet, welche in größere Umgebungen integriert sind. Sie werden eigens für spezielle Anwendungen entworfen und führen dezidierte Funktionen innerhalb eines Gesamtsystems aus. Eingebettete Systeme können sowohl Standard-Mikroprozessoren und Standard-Mikrocontroller als auch an die jeweilige Anwendung angepasste Hardware und Software enthalten [ROSE99]. Eingebettete Systeme übernehmen komplexe Steuerungs-, Regelungs- und Datenverarbeitungsaufgaben in technischen Systemen. Ihre Funktionalität wird durch Integration von Spezialhardware mit Prozessoren und Software realisiert.

Die Hauptaufgabe der eingebetteten Systeme besteht darin, Aktionen als Antwort auf bestimmte Eingangswerte aus der Umgebung auszuführen und Daten informationstechnisch zu verarbeiten [LiFä00]. Diese Aufgabe wird in der Regel von Mikrocontrollersystemen durchgeführt, die mit Hilfe von Sensoren und Aktoren mit der Umgebung kommunizieren. Abbildung 2.1 zeigt den schematischen Aufbau eines eingebetteten Systems.

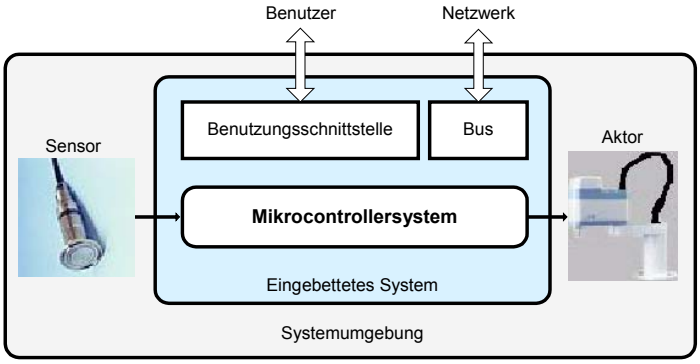

Abbildung 2.1: Schematischer Aufbau eines eingebetteten Systems

Mittlerweile befinden sich bereits in vielen Geräten wie z. B. in Haushaltsgeräten, in Konsumelektronikgütern und in medizinischen Geräten eingebettete Systeme [GöLa99]. In den letzten Jahren haben sich Einsatzbereiche und Anzahl der eingebetteten Systeme sehr stark vergrößert. Der Markt für eingebettete Systeme wird von der kontinuierlichen Verbesserung des Preis-Leistungs-Verhältnisses der Halbleiterindustrie angetrieben. Dadurch werden immer häufiger mechanische, hydraulische oder auch elektronische Steuerungen durch wettbewerbsfähige computerbasierte Systeme ersetzt. Es wird erwartet, dass der Markt für eingebettete Systeme in den nächsten zehn Jahren signifikant wachsen wird [HEAT99]. Auf Grund ihrer Flexibilität und – bei Großserienfertigung – niedrigen Stückkosten ersetzen sie zunehmend herkömmliche Technologien. Beispielsweise lässt sich die Achskonstruktion eines PKWs ohne Sicherheitseinbuße erheblich vereinfachen, wenn eine elektronische Stabilisierungshilfe (ESP) zum Einsatz kommt. Es ist insofern nicht verwunderlich, dass eingebettete Systeme zu den am schnellsten wachsenden Anwendungsbereichen der IT gehören, mit enormem Wertschöpfungspotential in der Telekommunikation, Produktionssteuerung, Verkehrstechnik und bei elektronischen Konsumgütern.

2.1.1 Eigenschaften von eingebetteten Systemen

Eingebettete Systeme besitzen eine Reihe charakteristischer Eigenschaften [Kop97]:

Massenproduktion: Eingebettete Systeme sind in der Regel für einen Massenmarkt bestimmt und werden deshalb für die Massenproduktion auf hochgradig automatisierten Fertigungsanlagen entwickelt. Dies setzt voraus, dass der Produktionsaufwand und die Materialkosten einer einzelnen Einheit so gering wie möglich sein müssen. Die effiziente Auslastung der Prozessoren und Speicher ist daher von hoher Bedeutung.

Statische Struktur: Das Computersystem ist in ein intelligentes Produkt mit vorgegebener Funktion und Struktur eingebettet. Die vorab bekannte statische Umgebung des Systems kann zur Entwurfszeit analysiert werden, um die Robustheit zu erhöhen, die Effizienz zu steigern und die Software des eingebetteten Systems zu vereinfachen.

Mensch-Maschine-Schnittstelle: Wenn ein eingebettetes System eine Mensch-Maschine-Schnittstelle besitzt, muss diese speziell für den vorgesehenen Einsatzzweck entworfen werden und einfach zu bedienen sein. Idealerweise sollte die Benutzung des intelligenten Produkts selbsterklärend sein und weder Schulung noch Nachschlagen in einer Bedienungsanleitung erfordern.

Minimierung des mechanischen Subsystems: Um die Produktionskosten zu reduzieren und die Zuverlässigkeit des intelligenten Produkts zu erhöhen, wird versucht, die Komplexität des mechanischen Subsystems zu minimieren.

Funktionalität in Software: Die Funktionalität eines intelligenten Produkts wird durch die integrierte Software bestimmt, die in einem Festspeicher abgelegt ist. Es gibt kaum eine Möglich-

keit, diese Software nach der Auslieferung zu modifizieren, weshalb sie hohen Qualitätsstandards genügen muss.

Wartbarkeit: Viele intelligente Produkte sind nicht wartbar, da die Modularisierung des Produkts in austauschbare Einheiten zu teuer ist. Wird ein Produkt jedoch so entworfen, dass es während seines Lebenszyklus gewartet werden kann, so ist eine Diagnose-Schnittstelle und eine verständliche Wartungsstrategie von hoher Bedeutung.

Kommunikation: Obwohl die meisten intelligenten Produkte zunächst als eigenständige Einheiten auf den Markt kommen, wird dennoch an viele Produkte die Anforderung gestellt, mit anderen größeren Systemen zu kommunizieren. Der Datentransfer soll dabei über ein einfaches und robustes Protokoll erfolgen. Die Optimierung der Übertragungsgeschwindigkeit spielt dabei selten eine Rolle.

2.1.2 Zeitgesteuerte und ereignisgesteuerte eingebettete Systeme

In Bezug auf die Reaktionen eingebetteter Systeme auf Stimuli aus ihrer Umgebung werden zwei verschiedene Paradigmen betrachtet: die zeitgesteuerten und die ereignisgesteuerten Systeme [GVNG94].

Zeitgesteuerte eingebettete Systeme sind Echtzeitsysteme, die die zeitlich vorgegebene Erfüllung bestimmter Bedingungen zur Folge haben. Bei den zeitgesteuerten Echtzeitsystemen werden alle Aktivitäten zu bestimmten Zeitpunkten angestoßen. Die Zustandsgrößen des technischen Prozesses werden zu festgelegten Zeitpunkten erfasst, wobei alle Teilprogramme periodisch ausgeführt werden. Daher sind zeitgesteuerte Systeme determiniert [LaGö99]. In diesem Zusammenhang wird zwischen harter und weicher Echtzeit unterschieden. Die Zeitspanne, in der ein Echtzeitsystem auf ein bestimmtes Ereignis reagieren muss, wird durch seine Umgebung vorgegeben [MüFä00]. Der Zeitpunkt, zu dem eine bestimmte Reaktion erfolgen muss, wird als Deadline bezeichnet. Ist eine Reaktion auch nach der Deadline noch akzeptabel, dann wird die Deadline als weich kategorisiert. Können durch eine nicht eingehaltene Deadline katastrophale Folgen eintreten, dann ist die Deadline hart. Ein Echtzeitsystem, das mindestens eine harte Deadline einhalten muss, ist ein hartes Echtzeitsystem bzw. ein sicherheitskritisches Echtzeitsystem. Gibt es keine harte Deadline, so spricht man von einem weichen Echtzeitsystem [Kop97].

In einem ereignisgesteuerten System werden alle Aktivitäten von Ereignissen aus der Umgebung ausgelöst, z. B. die Aktivierung von Tasks oder das Senden von Nachrichten. Ein Ereignis löst eine Änderung des Systemzustandes aus und kennzeichnet den momentanen Zustandswechsel bzw. -übergang [Schn99]. Die Organisation der Abläufe erfolgt durch ein Echtzeit-Betriebssystem. Ereignisgesteuerte Architekturen sind zwar sehr flexibel, jedoch im Allgemeinen nicht deterministisch und daher für sicherheitskritische Systeme nicht akzeptabel.

2.1.3 Wirtschaftliche Bedeutung eingebetteter Systeme

In der Regel liegt der bei weitem größte Anteil der Kosten eines intelligenten Produkts – über seinen gesamten Lebenszyklus betrachtet – in der Produktion. Für das eingebettete System spielen vor allem die Materialkosten der Hardware eine entscheidende Rolle. Die Entwicklungskosten und die Kosten der Software machen nur einen kleinen Anteil aus, der je nach Stückzahl des Produkts unter 5% liegen kann [VaGi01]. Ein Faktor, der durch die zunehmende Bedeutung der Dienstleistungen immer stärker ins Gewicht fällt, sind die Wartungskosten. Sie können eine beträchtliche Höhe annehmen, insbesondere wenn ein unentdeckter Softwarefehler einen Rückruf aller ausgelieferten Einheiten und den Austausch einer kompletten Serie nach sich zieht.

Im Zuge der Globalisierung eröffnen sich neue Absatzmärkte und billigere Produktionsstandorte. Die OECD (Organization for Economic Cooperation and Development, Organisation für wirtschaftliche Zusammenarbeit und Entwicklung), der Zusammenschluss der großen Industriestaaten, bezeichnet die Globalisierung als einen „Prozess, durch den Märkte und Produktion in verschiedenen Ländern immer mehr voneinander abhängig werden – dank der Dynamik des Handels mit Gütern und Dienstleistungen und durch die Bewegungen von Kapital und Technologie“.

Dieses ermöglicht den Herstellern von eingebetteten Systemen eine internationale Verflechtung der Wirtschaftsbeziehungen. Parallel dazu entwickeln sich Kommunikationstechnik und Datenverarbeitung sehr schnell. Zugriff und Austausch von Informationen werden durch technische Errungenschaften, wie zum Beispiel das Internet, revolutionär verändert.

Des Weiteren haben mit dem Eintreten der Globalisierung Dienstleistungen eine verstärkte ökonomische Bedeutung gewonnen. Die bereits erwähnten technischen Entwicklungen der Kommunikationstechnik haben die Möglichkeiten auf dem Dienstleistungssektor nachhaltig verändert. Dienstleistungen können heute mit Hilfe des Internets standortunabhängig angeboten werden. Außerdem eröffnet das Internet selbst, mit seinen zahlreichen Diensten, ein gewaltiges Potenzial an Dienstleistungen. Insbesondere E-Commerce hat dazu geführt, dass auch kleine Dienstleistungsunternehmen in der Lage sind, ihre Dienste weltweit anzubieten. Unternehmen aus anderen Branchen haben ebenfalls den ökonomischen und wettbewerbsspezifischen Nutzen von Dienstleistungen erkannt und damit begonnen, neben ihren eigentlichen Produkten, zusätzliche Dienstleistungen anzubieten. Diese zusätzlichen Dienstleistungen geben vielen Unternehmen, ganz besonders mittelständischen Unternehmen, die Möglichkeit, im weltweiten Konkurrenzkampf wettbewerbsfähig zu bleiben.

Dieser Trend beeinflusst auch die Automatisierungstechnik. Bei der technischen Vorbereitung der Produkte für die Erbringung von Dienstleistungen setzen viele Unternehmen auf das Internet als Kommunikationsmedium. Bezüglich der Art der Dienstleistungen spielen Themen wie Prozessdatenüberwachung (Monitoring), Fehlerdiagnose und Wartung eine wichtige Rolle. Diese

Dienste ermöglichen es, auf eventuell auftretende Probleme bei eingebetteten Systemen schnell zu reagieren, diese zu lokalisieren und eventuell über die Distanz zu beheben. Denkbar sind neben Fehlerdiagnose auch Software-Updates und Fernwartung.

Das Internet wird somit mit völlig neuen Diensten und Möglichkeiten versehen. Komplette Maschinen können via Internet zugänglich gemacht werden. Kunden an beliebigen Orten der Welt können diese Maschinen nutzen und damit Produkte in eigener Regie anfertigen. Beispielsweise könnte eine Maschine zur Anfertigung von Serienbriefen im Netz bereitgestellt werden, die eine individuelle Erstellung, Drucken, Eintüten und Versendung von Serienbriefen erlaubt [HWR99].

Die Realisierung dieser Dienste für eingebettete Systeme ist mit großen Herausforderungen verbunden. Eingebettete Systeme gehören zur niedrigen bis mittleren Preisklasse. Diese Tatsache entscheidet hauptsächlich über den Erfolg einer Internet-Lösung für eingebettete Systeme. Sie muss besonders kostengünstig sein.

Der Zugriff auf Prozessdaten bildet die Voraussetzung für die Realisierung von Fernservice-Applikationen für eingebettete Systeme. Dieser wird im folgenden Abschnitt vorgestellt.

2.2 Zugriff auf Prozessdaten in der Automatisierungstechnik

2.2.1 Anforderungen aus der Automatisierungstechnik

Die Anforderungen bezüglich der Kommunikation von Automatisierungsgeräten an einem Netzwerk, ändern sich gravierend, je nachdem welcher funktionalen Ebene einer Anlage oder Fabrik sie angehören. Gerade der Bereich der industriellen Sensorik und Aktorik bildet hier ein sehr spezifisches und eigenständiges Anforderungsprofil aus. Feldbuslösungen, die hier typischerweise zum Einsatz kommen, berücksichtigen diese Anforderungen aufgrund der traditionellen Sichtweise von „oben nach unten“ in den meisten Fällen jedoch nicht. Die Vernetzung von Steuerungen und Rechnern untereinander bzw. die von einfacher Sensorik und Aktorik können nicht als homogenes Feld gesehen werden, sondern stellen völlig unterschiedliche Anforderungen an ein Kommunikationsnetzwerk. Im Bereich der Prozessebene werden SPSen, NCs, Robotercontroller, Rechner usw. miteinander vernetzt. Die Kommunikation findet in der Regel sowohl mit Systemen oberhalb (z. B. Leitrechner) als auch zwischen Geräten innerhalb der Prozessebene statt. Es werden Daten und Programme im Umfang von einigen KByte bis hin zu mehreren MByte pro Teilnehmer bedarfs- bzw. ereignisgesteuert ausgetauscht. Diese Daten stellen eine einmalige Information dar, sodass deren Übertragung über entsprechende Quittierungs- und Wiederholungsmechanismen des Netzwerkprotokolls abgesichert sein muss. Dies entspricht der klassischen „peer to peer“-Kommunikation, die neben der Prozessebene auch in allen höheren Hierarchieschichten üblich ist. Im Bereich der Sensorik und Aktorik werden dagegen zwei zusätzliche Eigenschaften gefordert, denen in allen anderen Hierarchieebenen keinerlei Rechnung getragen werden muss. Zum einen ist dies die Fähigkeit des Netzwerks, eine zeitkonstante Abtastung von Prozessdaten durchzuführen, zum anderen spielt aufgrund der klei-

nen Datenmengen die Effizienz des Netzwerkprotokolls eine entscheidende Rolle. Die Ebene der Sensorik und Aktorik bildet mit ihren Geräten die Schnittstelle zwischen dem physikalischen Prozess und den Steuerungen und Prozessrechnern. Somit umfasst dieser Bereich sowohl einfachste Geräte, wie z. B. Endschalter und Schütze, die der traditionellen E/A-Ebene der Steuerungen zuzuordnen sind, als auch komplexe Endgeräte (z. B. Antriebssteuerungen) Regler und Bediengeräte. Diese „intelligenten“ Feldgeräte erlauben eine immer stärkere Dezentralisierung von Steuerungsfunktionen, die früher zentral realisiert wurden. Ein Netzwerk für die Sensorik und Aktorik muss damit sowohl einfachen als auch komplexen Geräten gerecht werden. Es ist daher zwingend erforderlich, bei der Festlegung der Anforderungen an das Netzwerk nicht nur eine Geräteklasse zu betrachten, wie z. B. Antriebskomponenten oder Sensoren, sondern die Gesamtheit der unterschiedlichen Gerätearten, die in einem solchen Netzwerk kooperieren. Nur ein ganzheitlicher Ansatz verhindert die Bildung proprietärer Einzelnetze innerhalb der Sensorik und Aktorik bzw. der Steuerungstechnik insgesamt. Dieser Ansatz eines einheitlichen Netzwerks, das sich an den Anforderungen aller Geräte der Einsatzumgebung orientiert, soll den Leitfaden für die weiteren Betrachtungen bilden. Analysiert man nun die Kommunikationseigenschaften der im Bereich der Sensorik und Aktorik anzutreffenden Geräte in Bezug auf die zu übertragenden Daten, so ergibt sich eine Aufteilung in zwei grundsätzliche Klassen von Gerätedaten: zum einen die so genannten E/A-Daten oder auch Prozessdaten, zum anderen die Klasse der Parameter. Beide Datenklassen unterscheiden sich grundlegend. Das Verständnis der daraus resultierenden unterschiedlichen Anforderungen an ein Kommunikationssystem stellt die elementare Basis für die Realisierung eines durchgängigen Sensor-/Aktor-Netzwerks dar.

2.2.2 Prozessdaten

Prozessdaten sind durch ihren unmittelbaren Bezug zum physikalischen Prozess charakterisiert. Sie können z. B. Schalterzustände oder Ansteuersignale für Schütze und Ventile oder aber auch Soll- und Istwerte von Antriebsreglern sein. Die Komplexität von Prozessdaten je Endgerät ist äußerst gering und umfasst typischerweise nur wenige Bits. Aufgrund des Dezentralisierungsgedankens ist man bestrebt, Prozessdaten möglichst am Entstehungsort zu erfassen und über das Netzwerk zugänglich zu machen. Hieraus resultiert eine sehr große Anzahl von Netzwerkteilnehmern, die durchaus im Bereich von einigen hundert liegen kann, aber dennoch nur eine äußerst geringe Informationsmenge abgeben. Dabei wird die Informationsmenge pro Teilnehmer stetig geringer. Lag sie noch vor kurzem bei 8 bis 16 Bit, so sind durch neue Techniken zur Vernetzung von Einzelsensoren heute schon Informationsmengen von nur 1 oder 2 Bit wirtschaftlich möglich. Prozessdaten sind zyklische Informationen, die ständig über das Netzwerk aktualisiert werden müssen. Zur Realisierung von Regelungs- und Steuerungsaufgaben sind außerdem konstante und berechenbare Abtastintervalle für Soll- und Istwerte erforderlich. Man spricht in diesem Zusammenhang auch von der Anforderung einer deterministischen und zeitäquidistanten Datenübertragung. Aufgrund der geringen Informationsmenge pro Netzwerkteilnehmer findet im Bereich der Prozessdaten in der Regel keine Datenvorverarbeitung vor Ort

statt. Die Übertragungszeiten müssen sich daher direkt an den Prozesszeitkonstanten orientieren. Unter Berücksichtigung der Leistungsfähigkeit heutiger Steuerungen und der erforderlichen Maschinentaktzeiten, muss die Aktualisierung aller in einem Netzwerk zusammengefassten Prozessdaten in einem Zeitraum von 1 bis 5 ms erfolgen können. Diese Zeitanforderung wird sich in Zukunft sogar noch weiter verringern. Prozessdaten sind in der Regel über ihre Adresse bzw. das Endgerät, von dem sie stammen, eindeutig identifizierbar. Eine zusätzliche Beschreibung zur Übertragung von Prozessdaten ist demnach nicht erforderlich. Die Bereitstellung von Prozessdaten an ein Anwendungsprogramm sollte in Form eines ständig aktualisierten Prozessabbildes erfolgen.

2.2.3 Parameter

Parameter dienen der Einstellung und Programmierung von „intelligenten“ Geräten. Im Gegensatz zu den Prozessdaten haben Parameterinformationen einen azyklischen Charakter. Das heißt, die Information wird nur bei Bedarf übertragen und damit nicht laufend erneuert. Ihre Übertragung bedarf besonderer Sicherungs- und Quittierungsmechanismen. Die Komplexität eines Parameterblocks im Bereich der Sensorik und Aktorik reicht von einigen 10 bis 100 Byte für die Parametrierung von Geräten bis zu einigen 100 KByte für Programminformationen. Im Vergleich zu den hochdynamischen Prozessdaten können die Zeitanforderungen für die Parameterübertragung im Allgemeinen als unkritisch angesehen werden. Sie liegen je nach Geräteart und Ausbau des Netzwerks im Bereich von einigen 100 ms bis zu einigen Minuten. Da intelligente Datenquellen unterschiedlichste Informationen enthalten können, ist ein Parameterblock nicht allein durch die Teilnehmeradresse des betreffenden Gerätes gekennzeichnet. Seine Übertragung erfordert vielmehr zusätzliche, beschreibende Informationen, die mit Hilfe so genannter Geräteprofile festgelegt werden.

Die genannten grundsätzlichen Datenklassen, Prozessdaten und Parameter, lassen sich in allen Geräten der Sensorik und Aktorik wieder finden. Einfache Feldgeräte werden im Allgemeinen nur durch Prozessdaten repräsentiert, „intelligente“ Feldgeräte, wie z. B. Antriebe, liefern bzw. erfordern dagegen beide Datenklassen. So erfolgt die Initialisierung und Parametrierung eines Antriebs über die Parameterinformationen, während z. B. Soll- und Istwert für Frequenz oder Drehzahl typische Prozessdaten darstellen, die zur Bewerkstelligung der Steuerungs- und Regelungsaufgaben benötigt und auf effiziente Weise im Netzwerk übertragen werden müssen. Aber auch im Bereich der „einfachen“ Endgeräte zeichnet sich immer deutlicher ein Trend in Richtung Parametrierbarkeit ab. Moderne Konzepte der seriellen Vernetzung bieten hierzu vielfältige Möglichkeiten. Dies erlaubt eine flexiblere Nutzung der Geräte und führt zu einer verbesserten Wertschöpfung.

Neben den aufgeführten unterschiedlichen Kommunikationsanforderungen der Prozessdaten und der Parameterinformationen lassen sich auch einige generelle Anforderungen an ein Sensor-/Aktor-Netzwerk aufstellen. Ein entscheidender Gesichtspunkt bei der Auslegung eines solchen

Bussystems ist, dass sich die Bedienung des Systems am Kenntnisstand des Personals in der Einsatzumgebung auszurichten hat. In der Praxis hat sich gezeigt, dass Netzwerke im Bereich der Sensorik/Aktorik nicht von EDV-Spezialisten betreut werden. Installation und Wartung müssen daher einfach und durchschaubar sein. Stochastische oder prioritätsgesteuerte Zugriffsverfahren erfordern aufwändige Leistungs- und Auslastungsanalysen des Netzwerks zur Bestimmung der Worst-Case-Zugriffszeit. Sie kommen daher für ein Sensor-/Aktor-Netzwerk nicht in Frage. Eine klare Master/Slave-Struktur gewährleistet deterministische Zugriffszeiten sowie eine einfache Projektierung des Netzwerks. Die Unterstützung von geringen Übertragungsraten ist für die Verwendung einfacher Medien äußerst wichtig. Dies gewährleistet gleichzeitig eine einfache Installation und einen hohen Grad an elektromagnetischer Verträglichkeit. Diese Forderung steht auf den ersten Blick im Widerspruch zu den ebenfalls geforderten geringen Übertragungszeiten. Zur Erfüllung beider Anforderungen muss das Protokoll des Sensor-/Aktor-Busses eine hohe Ausnutzung der verwendeten Übertragungsrate, d. h. eine hohe Effizienz, aufweisen. So gesehen ist es nicht automatisch ein Vorteil, wenn ein Netzwerk mit einer Übertragungsrate von beispielsweise 12 MBit arbeitet. Entscheidend ist nicht die Bussdatenrate, sondern der Nettodatendurchsatz, der mit der Übertragung erzielt wird. Die Anforderungen an die räumliche Ausdehnung des Netzwerks gehen aufgrund der unterschiedlichsten Anwendungen weit auseinander. Sie können im Bereich von einigen 10 m innerhalb einer Maschine bis hin zu einigen 100 m in der Montage-, Förder- und Lagertechnik liegen. Da in einem offenen Netzwerk Geräte verschiedener Hersteller betrieben werden sollen, kommt einem einfachen und vor allem herstellerneutralen Inbetriebnahme- und Diagnosewerkzeug eine besondere Bedeutung zu. Die beiden grundsätzlichen Geräte- bzw. Datenklassen der Sensorik und Aktorik stellen, wie bereits gezeigt, völlig gegensätzliche Anforderungen an das Kommunikationssystem. Zum einen sind E/A-Daten-transportierende Eigenschaften zur Realisierung der Echtzeitanforderungen zwingend erforderlich, zum anderen sind auch Nachrichten-transportierende Eigenschaften zur Realisierung der Parametrierung von Geräten wünschenswert. Diese gegensätzlichen Anforderungen dürfen aber nicht dazu führen, dass man in einem einheitlichen Einsatzgebiet zwei verschiedene Netzwerke für den Echtzeit-E/A-Datentransport und für die Parametrierung von Geräten verwendet. Dies würde bedeuten, dass z. B. Antriebe, da sie beide Datenklassen benötigen, mit zwei Netzwerkschnittstellen ausgestattet werden müssten, was zu einer für den Anwender inakzeptablen Kostensteigerung führen würde.

2.2.4 Datenübertragungsverfahren in der Automatisierungstechnik

In der klassischen Nachrichtentechnik wird die Datenübertragung als logische Punkt-zu-Punkt-Verbindung zwischen zwei Teilnehmern betrachtet. Zur Übertragung von zyklischen Prozessdaten muss ein zentraler Master die einzelnen Sensoren der Reihe nach mit einer Anfragenachricht zum Senden der Sensordaten auffordern. Die Aufforderung wird jeweils durch die Antwort des Sensors quittiert. In der gleichen Form wird die Übertragung von Daten zu Aktoren abgewickelt. Hierbei schickt die Steuerung eine Nachricht mit dem auszugebenden Datenwert an ei-

nen Aktor. Die Nachricht wird dann vom Aktor durch eine Antwort quittiert. Bei diesem Verfahren wird demnach zwischen dem zentralen Master und jeweils einem Slave ein komplettes Übertragungsprotokoll abgewickelt, wobei stets eine Nachricht ohne Nutzdateninhalt übertragen wird. In einem Telegrammrahmen müssen neben den eigentlichen Nutzinformationen auch eine Vielzahl von Protokolldaten (Adressierung, Kommando, Datensicherung usw.) übertragen werden. Da im Bereich der Sensorik und Aktorik typischerweise äußerst wenig Informationen pro Teilnehmer übertragen werden, ergibt sich hier aufgrund des aufwändigen Protokollmechanismus' zwangsläufig eine sehr geringe Effizienz bei der Übertragung von zyklischen Prozessdaten. Anders stellt sich die Situation bei der azyklischen Übertragung von Parameterinformationen dar. Diese entspricht voll und ganz dem Punkt-zu-Punkt-Prinzip der nachrichtentechnischen Übertragung. Des Weiteren ergibt sich durch lange Parameterblöcke eine wesentlich bessere Protokolleffizienz. Das nachrichtentechnische Übertragungsverfahren eignet sich daher sehr gut zur Übertragung von komplexen azyklischen Parameterblöcken, ist aber für die zyklische Übertragung von kurzen Prozessdaten nicht geeignet.

Als zweites grundsätzliches Übertragungsverfahren bietet sich das so genannte Summenrahmenverfahren an. Hier werden die Daten aller Sensoren und Aktoren eines Netzwerks in einer Nachricht zusammengefasst. Diese Nachricht wird an alle Geräte gleichzeitig gesandt. Der Übertragungs-Overhead tritt also nur einmal auf. Die Zusammenfassung der Informationen aller Netzwerkteilnehmer in einem Telegrammrahmen bewirkt eine drastische Vergrößerung des Nutzdatenblocks. Die Effizienz dieses Übertragungsverfahrens steigt mit der Anzahl der Netzwerkteilnehmer, die im Bereich der Sensorik und Aktorik typischerweise sehr hoch ist. Das Summenrahmenverfahren ist somit optimiert für die Übertragung von zyklischen Prozessdaten. Fügt man in ein solches Summenrahmentelegramm jedoch komplexe Parameterinformationen ein, so führt dies zu einer deutlichen Rahmenverlängerung und damit zur Verschlechterung der Übertragungszeiten für die kurzen Prozessdaten. Das Summenrahmenverfahren erfüllt die Forderung nach bestimmbaren und festen Abtastintervallen und schafft durch seine hohe Effizienz den Brückenschlag zwischen niedriger Datenübertragungsrate und kurzer Übertragungszeit.

Die Effizienzunterschiede bei der Prozessdatenübertragung mit dem nachrichtentechnischen Verfahren und dem Summenrahmenverfahren sollen in einem kurzen Beispiel erläutert werden. Zugrunde gelegt wird ein System mit 32 Netzwerkteilnehmern, die jeweils 8 Bit Prozessdaten bereithalten. Bei einem typischen nachrichtentechnischen Übertragungsverfahren erfordert die Aktualisierung von 8 Bit Informationsdaten einen Overhead von bis zu 200 Bits. Hieraus ergibt sich eine Effizienz von rund 4 %. Das heißt, es werden lediglich 4 % der Übertragungsbandbreite für Nutzinformationen ausgenutzt. Bei Verwendung des dem DEVICENET zugrunde liegenden CAN-Protokolls würde in genanntem Beispiel eine Effizienz von 8 %, beim IEC-Feldbus ein Wert von 5 % erreicht. Betrachtet man hingegen ein Summenrahmenverfahren mit einem einmaligen und konstanten Overhead, so lässt sich die Effizienz auf über 60 % steigern. Das heißt, zur Gewährleistung der gleichen Abtastzeiten muss beim nachrichtentechnischen

Verfahren die Übertragungsrate um mehr als eine Zehnerpotenz höher gewählt werden. Um mit diesem Verfahren überhaupt in den Bereich der schon heute benötigten Reaktionszeiten zu kommen, muss man folglich Übertragungsraten von mehreren MBit verwenden. Hierdurch wird sicherlich die Grenze des gerade noch akzeptablen Aufwands erreicht bzw. teilweise bereits überschritten. Gleichzeitig ergibt sich in Bezug auf die zukünftige Erweiterbarkeit eine Sackgasse, da die Anforderungen an die Reaktionszeiten des Netzwerkes im Zuge der fortschreitenden Dezentralisierung weiter ansteigen werden. Es besteht eine Tendenz zu immer mehr Teilnehmern im Netzwerk, die immer weniger Informationen bereithalten. Am Ende steht die Vernetzung von Einzelsensoren [Inter00]. Somit ist absehbar, dass das nachrichtentechnische Übertragungsverfahren schon bald keine probate Lösung mehr darstellen wird.

Moderne Halbleitertechnologien ermöglichen es heute, auch einzelne Sensoren und Aktoren kostengünstig an Bussysteme anzuschließen. Der Feldbus dient dazu, die Feldebene mit der Prozessleitebene zu verbinden. Er stellt damit die Kommunikation zwischen Sensoren, Aktoren, E/A-Multiplexen, prozessnahen Komponenten und Gateways zu anderen Bussystemen her [Bros92].

2.3 Web-Technologien

Der Begriff *Internet* prägt die Welt der modernen Informationstechnik ein. Das Internet ist zunächst einmal nichts anderes als ein weltweites Netzwerk, das lokale Computernetze miteinander verbindet. Dabei können die einzelnen Computernetze direkt oder indirekt miteinander verbunden sein. Dieses weltweite Netzwerk basiert auf einem einheitlichen Übertragungsprotokoll, dem TCP/IP (Transmission Control Protocol/Internet Protocol). Somit stellt das Internet eine Kommunikationsstruktur zum gleichberechtigten Informationsaustausch zur Verfügung [Balz99].

Mit der Entstehung des Internets wurden einige Dienste angeboten, die unterschiedliche Kommunikationsarten ermöglichten. Zu den wichtigsten Diensten gehören dabei die elektronische Post (E-Mail), die Übertragung von Dateien (FTP) und ganz besonders das so genannte WWW (World Wide Web). Das WWW stellt ein globales, interaktives, dynamisches, plattformübergreifendes und verteiltes Hypermedia-Informationssystem dar [Balz99]. Diese Dienste werden im Rahmen dieser Arbeit unter dem Begriff *Internet-Technologien* zusammengefasst.

Um das Internet und seine Dienste vielfältig nutzen zu können, gibt es heute zahlreiche Technologien. Allen ist gemeinsam, dass sie auf Internet-Technologien aufbauen. Sie nutzen das Internet mit seinen Netzwerk-technischen Standards und Protokollen als Kommunikationsmedium. HTTP, HTML, XML, Java-Applets usw. sind Beispiele hierfür. In dieser Arbeit werden alle diese Technologien, die mit der Entwicklung des Internets in Verbindung stehen, als *Web-Technologien* bezeichnet.

Die ständige Preissenkung im Bereich der Mikroprozessoren und Speicher führt dazu, dass der PC immer billiger und leistungsfähiger wird. Inzwischen ist der PC ein Bestandteil von fast jedem modernen Arbeitsplatz bzw. in vielen Haushalten. Zusätzlich bietet die rasante Verbreitung des Internets den Rechnern eine weltweite Vernetzungs- und Kommunikationsmöglichkeit. Damit ist der Mensch in der Lage, sich innerhalb weniger Sekunden von jedem Ort der Welt mit jedem beliebigen Rechner zu verbinden. Nahezu alle Neuerungen aus dem Bereich der Web-Technologien können auch in der Automatisierungstechnik einsetzt werden. Sie ermöglichen es, zeit- und ortsunabhängig eine Ferndiagnose und Fernvisualisierung bzw. Fernwartung und Fernbedienung durchzuführen. Darüber hinaus wird über das Internet eine Realisierung der Produktionsverfolgung, Produktdokumentation und Qualitätssicherung möglich. Hochspezialisierte Stand-Alone-Anwendungen können mit Hilfe der Web-Technologien zu einem heterogenen, auch über mehrere Standorte verteilten Gesamtsystem zusammengefügt werden. Die Web-Technologien bieten für die Automatisierungstechnik eine Reihe von Vorteilen, denen aber auch Nachteile gegenüber stehen. Die Tabelle 2.1 stellt die wichtigsten Vor- und Nachteile gegenüber:

Tabelle 2.1: Vor- und Nachteile von Web-Technologien

Vorteile	**Nachteile**
• System- und Herstellerunabhängigkeit • Offene Standards • Mehrfachnutzung einer Anwendung durch Client/Server Architektur • Homogene und systemübergreifende Strukturen • Beibehaltung der bisher verwendeten Hardware bzw. Software • Keine zusätzliche Software auf der Client-Seite notwendig • Austausch von Informationen durch Hypermedia	• Sicherheitslücken • Kein Determinismus • Abhängigkeit der Übertragungsgeschwindigkeit von der Netzbelastung

Im Folgenden werden diejenigen Web-Technologien, die für diese Arbeit von besonderer Bedeutung sind, kurz umrissen. Alle diese Technologien können zur Übertragung von Prozessdaten eingesetzt werden. Hierbei handelt es sich um die Technologien HyperText Transfer Protocol (HTTP), eXtensible Markup Language (XML), Remote Method Invocation (RMI) und Sokketverbindung.

2.3.1 HyperText Transfer Protocol

Das HyperText Transfer Protocol (HTTP) ist ein verbindungsorientiertes und zustandsloses Protokoll. Der Begriff zustandslos bedeutet in diesem Zusammenhang, dass der Server sich nicht merkt, was eine HTTP-Anfrage bewirkt hat. Es ist dem Client nicht möglich, den Server in einen anderen Zustand zu versetzen und später eine weitere Anfrage abzusenden, die diesen Zu-

stand voraussetzt und darauf zurückgreift. Das HTTP bildet die Basis für die Übertragung von Dokumenten im World Wide Web (WWW). Damit wird die Kommunikation zwischen einem Webserver und einem Webbrowser möglich. In diesem Zusammenhang übernimmt der Webbrowser die Funktion des Clients, der eine Anfrage an den Webserver stellt. Daraufhin sendet der Webserver genau eine Antwort in Form einer Beschreibungssprache oder eines Skriptes an den Webbrowser. Dieser Ablauf kann in folgende vier Schritte unterteilt werden:

1. *Verbindungsaufbau:* Der Webbrowser öffnet eine TCP-Verbindung zum Webserver. TCP steht für Transmission Control Protocol und stellt gesicherte Vollduplex-Verbindungen zur Verfügung [Blac99].
2. *Anfrage (Request):* Der Webbrowser sendet eine HTTP-Anfrage über die bereits aufgebaute Verbindung an den Webserver.
3. *Antwort (Response):* Auf die HTTP-Anfrage des Webbrowsers antwortet der Webserver in der Regel in Form eines HTML[2]-Dokuments. HTML ist eine Beschreibungssprache, wobei Markierungen, die die Struktur des Textes beschreiben, in den Text eingebettet sind.
4. *Verbindungsabbau:* Im Allgemeinen schließt der Webserver die TCP-Verbindung, nachdem er die Antwort gesendet hat.

HTTP bietet verschiedene Methoden für die Kommunikation zwischen Server und Client. Die *GET*-Methode dient zum Abrufen eines HTML-Dokuments [RaRa98]. Die *HEAD*-Methode wird eingesetzt, um bestimmte Informationen über dieses Dokument als Antwort zu erhalten. Damit ist der Webbrowser in der Lage zu überprüfen, ob das Dokument verändert wurde. In diesem Falle holt er das Dokument erneut. Die *POST*-Methode wird zur Übertragung der Daten vom Webbrowser zum Webserver verwendet. Schließlich gibt es die *PUT*- und die *DELETE*-Methode, die dem Upload und Löschen eines HTML-Dokumentes beim Webserver dienen. Tabelle 2.2 fasst die Vor- und Nachteile des HTTP zusammen:

Tabelle 2.2: Vor- und Nachteile von HTTP bei der Übertragung von Daten

Vorteile	**Nachteile**
• De-facto-Standard im Internet	• Zur Übertragung dynamischer Inhalte sind zusätzliche Erweiterungen nötig (z. B. CGI, Servlet-Engine)
• Unabhängigkeit von Programmiersprachen	• Zustandsloses Protokoll
• Authentifizierungsmöglichkeit (eingeschränkt)	• Unidirektionale Kommunikation

[2] HyperText Markup Language

2.3.2 Socketverbindung

Sockets charakterisieren eine Abstraktionsebene für eine einheitliche Prozesskommunikation. Sie stellen eine Programmbibliothek dar, mit deren Hilfe man Daten zwischen zwei Prozessen austauschen kann. Die beiden Prozesse können dabei auf einem einzelnen Rechner arbeiten, sie können aber auch auf zwei verschiedenen Computern ablaufen. Im letzteren Fall werden die Daten über ein Netzwerk übertragen. Dies geschieht für den Programmierer vollkommen transparent, er muss in beiden Fällen nur die Quell- und Zieladressen ändern. Diese Adressierung erfolgt über die Ports. Die Ports stellen Kommunikationspunkte dar, die zur Kommunikation natürlich noch nicht ausreichen. Es wird ein Kommunikationskanal benötigt, der eine Verbindung zweier Ports repräsentiert. Solch ein Kommunikationskanal wird durch ein Socket bereitgestellt [Boge99]. Die Sockets stellen den De-facto-Standard für die verbindungsorientierte Kommunikation über TCP/IP dar [Haas01]. Sie stellen eine abstrakte Schnittstelle zur Verfügung, um TCP-Verbindungen auf- bzw. abzubauen und Daten zu übertragen. Die Endpunkte der Kommunikation liegen dabei innerhalb der einzelnen verteilten Anwendungen. Damit sich Client und Server verständigen können, ist es notwendig, ein geeignetes Anwendungsprotokoll festzulegen. Der Abstraktionsgrad aus Sicht des Programmierers ist also wesentlich niedriger. Daher spricht man auch davon, dass Sockets eine Low-Level-Schicht darstellen.

Der Ablauf einer Socketkommunikation lässt sich in drei klassische Phasen einteilen, die auch aus anderen Bereichen der Telekommunikation bekannt sind [Kühn01], nämlich Verbindungsaufbau, Datenaustausch und Verbindungsabbau.

Auf Seiten des Servers wird zunächst ein neues Objekt der Klasse Server-Socket erzeugt. Der Konstruktor der Klasse erfordert die Angabe eines Ports, der für Verbindungen zum Client reserviert wird. Danach wird die Methode *accept()* des Objekts aufgerufen. Damit ist der Server in Bereitschaft.

Der Ablauf der Verbindung aus der Sicht des Servers kann in folgenden Schritten zusammengefasst werden:

1. *Verbindungsaufbau:* Die Methode `accept()` dient nur zur ersten Kontaktaufnahme und liefert einen weiteren Socket, sobald sich ein Client anmeldet.
2. *Datenaustausch:* Ermitteln des Ein- und Ausgabestroms über die Methoden `getInputStream()` und `getOutputStream()` des neuen Sockets, der mit dem Client verbunden ist. Über die Ein- und Ausgabeströme können nun Daten verschickt werden.
3. *Verbindungsabbau:* Schließen des Sockets mit `close()` bei Beenden der Verbindung zum Client.

Abbildung 2.2 zeigt die Verbindungsanfrage eines Clients beim Server.

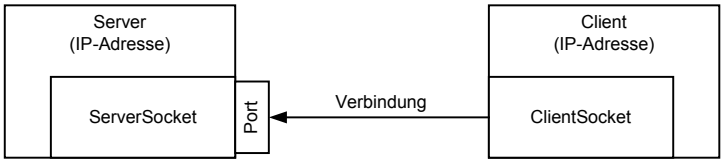

Abbildung 2.2: Verbindungsanfrage beim Server

Aus Sicht des Clients gestaltet sich der Ablauf wie folgt:

1. *Verbindungsaufbau:* Erzeugen eines Objekts der Klasse `Socket`, als Parameter des Konstruktors werden die Adresse des Servers und die dort reservierte Portnummer übergeben.
2. *Datenaustausch:* Lesen und Schreiben von Daten über Ein- und Ausgabeströme.
3. *Verbindungsabbau:* Schließen des Sockets mit `close()` bei Beenden der Verbindung zum Server.

Abbildung 2.3 zeigt die Verbindung, nachdem sie hergestellt ist. Nach dem Verbindungsaufbau wird ein Kommunikationskanal zwischen zwei Knoten aufgebaut.

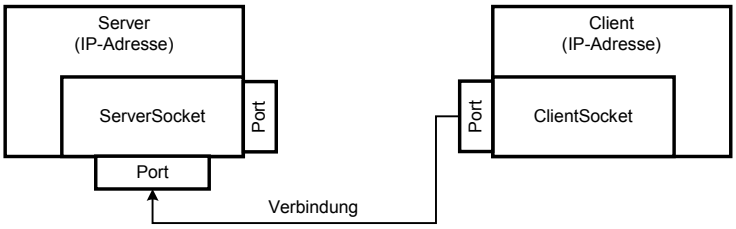

Abbildung 2.3: Herstellung einer Verbindung zwischen Server und Client

Eindeutige Vorteile der Kommunikation über Sockets sind die Plattformunabhängigkeit und die Geschwindigkeit. Die Tatsache, dass der Entwickler selbst für das Anwendungsprotokoll Sorge zu tragen hat, bietet natürlich den Vorteil der vollen Kontrolle über Art und Umfang des Protokolls. Besonders der Umfang wirkt sich direkt auf die Geschwindigkeit aus. Auf der anderen Seite liegt hier eine mögliche zusätzliche Fehlerquelle, und der Zeitaufwand einer eigenen Implementierung sollte auf keinen Fall unterschätzt werden.

Schwierigkeiten bereitet bei der Kommunikation auch die Synchronisation von Aktivitäten des Servers mit denen des Clients. Prinzipiell können beide gleichzeitig senden und empfangen, da zwei voneinander unabhängige Datenströme bestehen. Aus diesem Grund sind Mechanismen zu implementieren, die sicherstellen, dass der Server auf eine bestimmte Nachricht wartet, die der Client sendet, und umgekehrt. Außerdem muss man sich Gedanken über Time-out-Funktionen machen, die sicherstellen, dass der Server nicht unendlich lange auf eine Antwort des Clients wartet, obwohl dieser nicht antworten kann, weil z. B. ein Fehler aufgetreten ist.

Ein weiterer nicht unerheblicher Punkt ist der Einsatz mehrerer Threads auf Seiten des Servers, um mehrere Verbindungen zu Clients herzustellen. Hierbei ist wichtig, dass allen Threads genügend Rechenzeit zur Verfügung steht, um ihre Aufgaben schnell auszuführen. Jeder Thread wartet auf den Eingabestrom von einem Client und sollte nur dann Rechenzeit beanspruchen, wenn er eine Nachricht erhält und diese verarbeiten muss. Tabelle 2.3 stellt die Vor- und Nachteile einer Socketverbindung über TCP/IP gegenüber.

Tabelle 2.3: Vor- und Nachteile einer Socketverbindung über TCP/IP

Vorteile	Nachteile
• Einsatzmöglichkeiten auf unterschiedlichen Plattformen • Unabhängigkeit von Programmiersprachen • Bidirektionale Kommunikation • Minimaler Programmieraufwand und Ressourcenverbrauch • Zustandsorientiertes Protokoll	• Austausch von Methodenaufrufen nicht möglich • Kein Kommunikationsmechanismus • Low-Level-Protokollschicht • Keine Authentifizierungs- und Verschlüsselungsmechanismen

2.3.3 Remote Method Invocation

Remote Method Invocation (RMI) ist ein Java-Mechanismus, um Methoden von Objekten, die nicht auf derselben Java Virtual Machine (JVM) laufen, aufrufen zu können. Dabei bietet der RMI eine weitgehende Transparenz, sodass nach einer anfänglichen Initialisierung ein Aufruf genauso verwendet wird wie im lokalen Fall [Boge99]. Daneben ermöglicht RMI auch die Behandlung von Fehlern (Exceptions), die beim Ausführen einer entfernten Methode auftreten können und an das aufrufende Objekt weitergeleitet werden. Aus Sicht des Clients erfolgen die Aufrufe wie bei lokalen Methoden mit der Ausnahme, dass die beschriebenen Exceptions auftreten können. Die Programmiersprache Java stellt dazu einen Satz von Schnittstellen und Klassen in Form eines Application Programming Interfaces (API) bereit. Die Verwendung von RMI ist auf die Verwendung in einer reinen Java-Umgebung beschränkt [Down98].

RMI lässt sich auch als „Middleware-Ansatz“ [Haas01] der Kommunikation zwischen Server und Client mit Hilfe von Methodenaufrufen beschreiben. Die Aufgaben einer Middleware sind vor allem das Zulassen von Kommunikation zwischen verschiedenen Prozessen sowie die Realisierung möglichst vieler Transparenzen. Das bedeutet, dass beim Entwickeln einer verteilten Anwendung ähnlich vorgegangen werden kann wie bei einer zentralen Anwendung.

Die Übermittlung von Methodenaufrufen zwischen einem Client und einem Server erfolgt in mehrere Schichten [Gros02]. Abbildung 2.4 zeigt den prinzipiellen Aufbau des RMI Systems.

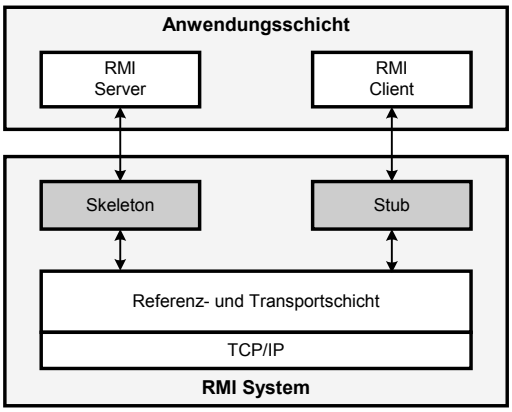

Abbildung 2.4: RMI Schichtenarchitektur

Die untere Referenz- und Transportschicht und die TCP/IP-Schicht stellen die eigentliche Übertragung der Informationen sicher. Die Realisierung des verwendeten Übertragungsprotokolls und des Verbindungsaufbaus bleiben dem Programmierer verborgen und werden daher nicht näher erläutert.

Von Interesse sind dagegen Stub und Skeleton, die die Schnittstelle zwischen der Anwendung und dem Rest des RMI-Systems bilden [Schu00]. Ein Client, der eine Methode eines entfernten Objekts aufrufen möchte, verwendet den Stub, welcher den Aufruf entgegennimmt und weiterleitet. Auf der Seite des Servers erhält der Skeleton diesen Aufruf und gibt ihn an das eigentliche Objekt weiter. Besitzt die Methode einen Rückgabewert, wird dieser auf dem umgekehrten Weg zurück zum Client transportiert.

Kommt es auf dem Server beim Ausführen der Methode zu einem Fehler, so produziert die Methode eine `RemoteException`, die dann ebenso wie ein Rückgabewert zurück an den Client geschickt wird. Diese Art der Fehlerbehandlung ist äußerst bequem, da die Behandlung von Exceptions mittels `try`- und `catch`-Blöcken ein Konzept von Java darstellt. Durch Benutzung des bereits vorhandenen Konzepts ist die Erstellung einer Anwendung „aus einem Guss“ möglich und die Sicherstellung einer entsprechenden Qualität der Software wird wesentlich erleichtert. Den Ablauf des Methodenaufrufs mit RMI wird in Abbildung 2.5 graphisch dargestellt.

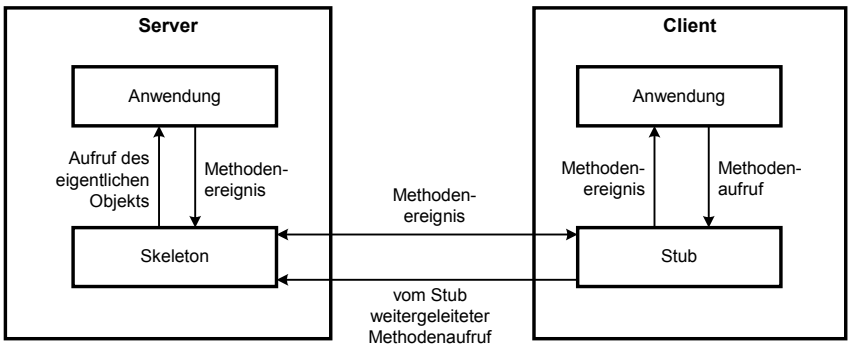

Abbildung 2.5: Methodenaufruf mit RMI

Zum Erstellen von Stubs und Skeletons wird der Compiler `rmic` verwendet, der zur Java- Entwicklungsumgebung gehört. Folgende Schritte müssen beim Entwurf einer verteilten Anwendung mit Hilfe von RMI durchgeführt werden [Haas01]:

- Definition der Schnittstelle(n)
- Implementierung der Schnittstelle(n), Entwicklung von Server und Client
- Erzeugung der Server-Skeletons und Client-Stubs mit `rmic`
- Registrierung der Implementierung in der RMI-Registry

Zuerst muss festgelegt werden, welche Methoden des Servers vom Client aufgerufen werden sollen. Diese Methoden werden dann in einem Interface definiert, das die Schnittstelle zwischen den verteilten Anwendungen beschreibt. Das eigene Interface ist dabei eine Erweiterung des im Paket `java.rmi` vorhandenen Interfaces `Remote`. Jede darin enthaltene Methode muss eine `RemoteException` auslösen, die – wie bereits beschrieben – im Fehlerfall an den aufrufenden Client weitergeleitet wird.

Eine Klasse der Serveranwendung implementiert die in der Schnittstelle enthaltenen Methoden, welche dann vom Client aufgerufen werden können. Um dies zu gewährleisten, muss aus der entsprechenden Klasse des Servers mit Hilfe von `rmic` ein Stub und ein Skeleton generiert werden.

Zusätzlich zur Implementierung des Interfaces muss die Serveranwendung um die Klasse `UnicastRemoteObject` aus dem Paket `java.rmi.server` erweitert werden. Dadurch werden die Funktionalitäten von RMI auf Seiten des Servers zur Verfügung gestellt. Wird ein Objekt dieser Klasse auf dem Server erzeugt, so wird das Objekt in einer Art Datenbank, der so genannten RMI-Registry, registriert. Dem Objekt wird ein Name zugewiesen, über den der Client auf das Objekt zugreifen kann. Zu diesem Zweck wird die Klasse `java.rmi.Naming` verwendet, die zwei Methoden `bind` und `rebind` zur Verfügung stellt. Der Quellcode hierzu könnte wie folgt aussehen:

```
1:    Naming.rebind("//127.0.0.1:4500/Server", object);
2:       //rebind object to rmiregistry running on port 4500
```

Abbildung 2.6: Beschreibung des Objekts in der RMI-Registry

Um RMI in der Praxis einsetzen zu können, ist es notwendig, das Sicherheitsmodell von Java zu berücksichtigen. Das heißt, es muss jedem Client der Zugriff auf den Server durch entsprechende Zugriffsrechte gewährt werden.

Eine weitere sehr nützliche Eigenschaft von RMI ist es, dass Daten einfach zwischen entfernten Objekten ausgetauscht werden können. Ruft der Client eine Methode des Servers auf, können

Daten als Parameter der Methode übergeben werden. Andererseits ist es wie bereits erläutert möglich, dass der Server Daten als Rückgabewert an den Client sendet.

Neben den bisher beschriebenen Mechanismen gibt es noch die Möglichkeit, dass der Server seinerseits Methoden des Clients aufruft. Dies wird mit Callback bezeichnet. Callbacks können notwendig werden, wenn der Server selbst Daten an den Client schicken oder Funktionen ausführen möchte. Um diesen Mechanismus zu nutzen, wird analog zur obigen Vorgehensweise eine Schnittstellenbeschreibung für den Client vorgenommen und es werden ebenfalls ein Stub und ein Skeleton erzeugt. Dadurch erklärt sich auch, dass evtl. mehrere Interfaces für Server und Client definiert werden müssen. Das Anmelden bei einer Registry sowie das Erweitern von `UnicastRemoteObject` entfällt allerdings. Deshalb kann der Server nur auf Methoden des Clients zugreifen, wenn sich dieser vorher beim Server angemeldet hat, daher der Begriff Callback.

2.3.4 Common Object Request Broker Architecture

CORBA (Common Object Request Broker Architecture) ist eine von der OMG aufgestellte Spezifikation für eine objektorientierte verteilte Architektur auf der Anwendungsebene. Die OMG, die von acht Firmen im Jahre 1989 ins Leben gerufen wurde, umfasste Ende 1997 schon mehr als 750 Mitglieder. Sie verfolgt das Ziel, einen allgemeinen Architektur-Rahmen für objektorientierte Anwendungen zu schaffen, der auf einer weltweit verbreiteten Schnittstellenspezifikation basiert. Im Vergleich zu DCOM (Distributed Component Object Model) von Microsoft oder RMI (Remote Method Invocation) unter Java handelt sich bei CORBA um eine reine Schnittstellen-Spezifikation.

CORBA läuft unter zahlreichen Betriebsystemen und wird von vielen Anbietern in den unterschiedlichsten Programmiersprachen implementiert (siehe Abbildung 2.7).

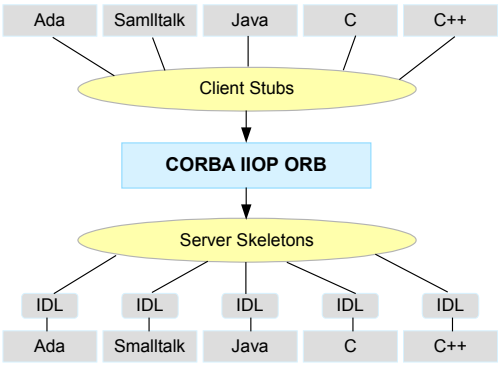

Abbildung 2.7: Kommunikation verschiedener CORBA-Implementierungen

Das CORBA Objektmodell (CORBA/OM) bildet die Grundlage zur Beschreibung von CORBA und basiert auf dem OMG Objektmodell (OMG/OM). CORBA/OM beschreibt und definiert

eine spezielle Technologie, die sich auf die Interaktion zwischen Client und Server konzentriert. Im Vordergrund stehen hierbei Konzepte für die Clientanfragen. Das Fundament der CORBA/OM spezifiziert die so genannte Object Management Architecture (OMA). Diese Architektur wird anhand eines Referenzmodells beschrieben, dessen zentraler Bestandteil der Object Request Broker (ORB) ist. Wie in Abbildung 2.8 zu sehen, besteht das Modell aus vier Komponenten:

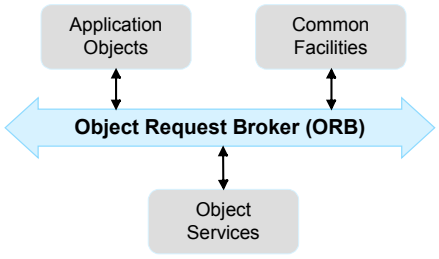

Abbildung 2.8: Object Management Architecture

1. *Object Request Broker (ORB):* Der ORB ist der zentrale Bestandteil der OMA, er stellt den Kommunikationsmechanismus zwischen Client und Server zur Verfügung. Er erlaubt Objekten, transparente Methodenaufrufe auf einem entfernten oder lokalen Server durchzuführen, d. h. für den Client sieht der Request wie ein ganz normaler lokaler Methodenaufruf aus. Es ist die Aufgabe des ORB, das entsprechende Server-Objekt zu lokalisieren, es evtl. zu starten, ihm den Methodenaufruf zu vermitteln und die Rückgabeparameter innerhalb der Antwortzeit an den Client zurückzuliefern.

2. *Object Services*: Object Services sind Objekte, die grundlegende Dienste anbieten, welche essentiell für die Entwicklung von interoperabler und verteilter Software sind und als Bausteine für verschiedenste Anwendungen verwendet werden können. Einige Beispiel hierfür sind: Naming-Service, Event-Service, Lifecycle-Service u.a.

3. *Common Facilities*: Common Facilities sind CORBA-Objekte, die spezielle Dienste zur Verfügung stellen. Beispiele hierfür sind Drucker-, Datenbanken- oder Electronic-Mail-Dienste. Common Facilities sind für diese Arbeit nicht weiter von Bedeutung.

4. *Application Objects*: Application Objects stellen die eigentlichen Applikationskomponenten dar, d. h. sie sind die von den Anwendungsentwicklern implementierten Objekte und können Object Services und Common Facilities der OMA – unter Nutzung des ORB – verwenden.

2.3.4.1 CORBA Standard-Request

Das Herzstück der oben beschriebenen Architektur bildet der ORB (Object Request Broker). Er ermöglicht die Interaktion zwischen Client und Server. Die Spezifikation definiert auf der Applikationsebene eine Kommunikationsinfrastruktur, die es Clients ermöglicht, Services (Methoden) von bestimmten Server-Objekten aufzurufen. Wie unter RPC (Remote Procedure Call)

werden von CORBA sowohl synchrone, asynchrone als auch unidirektionale Aufrufe unterstützt. Alle Aufrufe werden durch den ORB bearbeitet, der einen orttransparenten, programmiersprachen- und plattformunabhängigen Datenaustausch ermöglicht. Er lokalisiert die korrekte Implementierung des aufzurufenden Service und trifft alle notwendigen Vorkehrungen, damit der Server diesen Service ausführen kann (siehe Abbildung 2.9).

Abbildung 2.9: CORBA Standard-Request

Die Services, die ein Server-Objekt zur Verfügung stellt, werden durch die Interface Definition Language (IDL) definiert, die ein Bestandteil von CORBA ist. Die IDL ist ausschließlich eine Sprache zur Daten- und Schnittstellenbeschreibung von Server-Objekten und somit auch sprach- und plattformunabhängig. Der IDL-Precompiler, der bei jeder CORBA Implementierung mitgeliefert wird, generiert entsprechende Klassen, mit deren Hilfe der Client über den ORB auf die Services von Server-Objekten zugreifen kann. Eine solche IDL-Definition ist in Abbildung 2.10 beispielhaft vorgestellt.

```
1:    module BeispielIDL
2:    {
3:        interface Communication
4:      {
5:        void sendMessage(in string szMessage)
6:      };
7:    };
```

Abbildung 2.10: Beispiel für eine IDL-Definition

Diese IDL definiert einen Service „sendMessage“, der einem Client eine Übermittlung einer Zeichenkette an das Server-Objekt ermöglicht.

2.3.4.2 CORBA Event Service

Im Gegensatz zum Standard CORBA Request stellt der *Event Service* von CORBA einen asynchronen, dezentralen Kommunikationsmechanismus zur Verfügung. Es werden zwei Rollen definiert: *Supplier* (Erzeuger) sind Objekte, die Ereignisse produzieren, und *Consumer* (Verbraucher) sind Objekte, die Ereignisse verarbeiten. Die Entkopplung von *Supplier* und *Consumer* wird durch einen *Event Channel* (Ereigniskanal) realisiert. An einen *Event Channel* können sich beliebig viele *Supplier* und *Consumer* anbinden.

Die Behandlung der Ereignisse kann durch zwei verschiedene Modelle beschrieben werden. Im Pushmodell übernimmt der *Supplier* die Initiative und sendet die von ihm erzeugten Ereignisse an den *Event Channel*. Der *Event Channel* sendet automatisch das Ereignis an alle angeschlos-

senen *Consumer*. Im Gegensatz dazu nimmt beim Pullmodell der *Consumer* die aktive Rolle ein. Er fragt beim *Event Channel* aktiv nach Ereignissen, woraufhin der *Event Channel* bei den *Suppliern* bereits erzeugte Ereignisse abholt. Eine Mischung der beiden Modelle ist auch möglich, z. B. ein Push-Supplier und ein Pull-Consumer.

Der *Event Channel* ist die zentrale Komponente im *Event Service*. Er sendet eingehende Ereignisse der Push-Supplier an die *Consumer* weiter oder fordert von Pull-Suppliers Ereignisse an. Der *Event Channel* ist für die *Supplier* eine Art *„Proxy"-Consumer* und für die *Consumer* ein *„Proxy"-Supplier*, d. h. er spielt die Rolle der Gegenseite (siehe Abbildung 2.11).

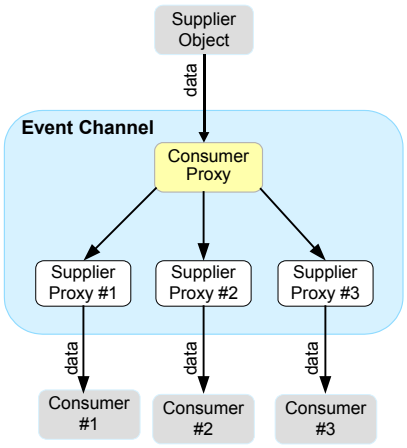

Abbildung 2.11: Beispiel mit einem Supplier, einem Event Channel und mehreren Clients

2.3.5 eXtensible Markup Language

2.3.5.1 Einführung

XML[3] gehört wie HTML zu den Auszeichnungssprachen (Markup Languages). XML ist also keine Programmiersprache, mit der Befehle für den Prozessor eines Computers beschrieben werden können. XML wurde von einer XML-Arbeitsgruppe (ursprünglich bekannt als das SGML Editorial Review Board) entwickelt, die 1996 unter der Schirmherrschaft des World Wide Web Consortium (W3C) gegründet wurde [Clar97].

Eine Auszeichnungssprache enthält Anweisungen für die Darstellung von Informationen. HTML kümmert sich vor allem darum, wie ein Dokument vom Internet-Browser dargestellt werden soll. Jedoch sind in HTML Angaben zum Inhalt des Dokuments und seiner Struktur nur in begrenztem Umfang möglich. Angaben zur Art von Informationen, so genannte Metadaten (Informationen über Informationen), sind sinnvoll, wenn es um eine gezielte Nutzung von Informationen oder ihre elektronische Verwendung geht.

---

[3] eXtensible Markup Language

Dies wird mit Hilfe von XML möglich: Hier werden Inhalt und Form, Struktur und Gestaltung getrennt behandelt. XML-Dokumente enthalten zunächst nur Anweisungen, die Auskunft über ihre inhaltliche Struktur und bestimmte Attribute von Informationen geben. „eXtensible“ bedeutet dabei, dass XML erweiterbar ist und sich an nahezu alle Bedürfnisse anpassen lässt [Bret00]. So können ganz verschiedene Strukturen für unterschiedliche Arten von Dokumenten aufgebaut werden. Die unterschiedlichen XML-Dokumente können mit dem gleichen Werkzeug erstellt, bearbeitet, transportiert und archiviert werden.

Ein großer Vorteil von XML ist die Tatsache, dass die nach XML erstellten Daten sowohl für Menschen als auch für Maschinen lesbar sind. XML-Daten liegen wie HTML als ASCII-Text vor. Ein weiterer Vorteil ist die Plattformunabhängigkeit. Die XML-Dokumente können von verschiedenen Betriebssystemen gelesen und bearbeitet werden. Außerdem unterstützen auch mobile Kleincomputer (wie **P**ersonal **D**igital **A**ssistent [PDA] oder Mobiltelefone) den XML-Standard.

2.3.5.2 Aufbau eines XML-Dokuments

Bei der Verarbeitung und Präsentation von XML-Dokumenten spielen folgende drei Dateien eine maßgebliche Rolle [Doss00]:

Das *XML-Dokument* ist ein Datenobjekt, das den Bedingungen der Wohlgeformtheit (s. 2.3.5.3) entspricht, wie sie in der XML-Spezifikation definiert wird. Es besteht aus Markup- und Zeichendaten [Megg98].

Die *Document Type Definition (DTD)* beinhaltet eine Reihe von Regeln, die die Grammatik eines Dokumentes bestimmen. Sie wird vom Benutzer definiert. DTD wird im folgenden Kapitel ausführlicher beschrieben.

Das *Stylesheet* ist die Beschreibung einer Instanz eines XML-Dokuments, die durch Layout-Vokabular transformiert ist. Es erweitert die logische und physikalische Struktur eines XML-Dokuments um eine visuell-formale Struktur, die bestimmt, wie das Dokument in einer beliebigen Umgebung dargestellt wird [Holz01].

2.3.5.3 Document Type Definition (DTD)

Die DTD legt den strukturellen Aufbau einer Klasse von XML-Dokumenten fest. Ein XML-Dokument, das den Ansprüchen der XML-Spezifikation des W3C genügt, aber nicht auf eine DTD verweist, nennt man *wohlgeformt*. Ein XML-Dokument, das sich sowohl an die XML-Spezifikation als auch an die Regeln hält, die von seiner DTD festgelegt werden, wird als *gültig* bezeichnet. Abbildung 2.12 zeigt ein Beispiel für ein einfaches XML-Dokument.

```
1:    <?xml version="1.0"?>
2:    <!DOCTYPE root SYSTEM "example.dtd">
3:    <root>
```

```
4:    <Vorname>John</Vorname>
5:    <Name/>
6:    <Adresse>
7:    <Strasse>Paffenwaldring</Strasse>
8:    <Nummer>47</Nummer>
9:    <PLZ>70550</PLZ>
10:   <Stadt>Stuttgart</Stadt>
11:   </Adresse>
12:   </root>
```

Abbildung 2.12: Beispiel für ein XML-Dokument

Die einzelnen Elemente von XML bestehen aus je einem Anfangs- und einem Ende-Tag. Allerdings muss bei XML streng darauf geachtet werden, dass jeder Anfangs-Tag auch wirklich ein Ende-Tag besitzt. Auch so genannte leere Elemente müssen bei XML mit einem Ende-Tag abgeschlossen werden. Zur besseren Lesbarkeit gibt es die Möglichkeit, beide Tags zu vereinen indem ein Schrägstrich („/“) vor das abschließende Größer-Zeichen („>“) gestellt wird. Dies ist in dem obigen Beispiel bei dem Element `Name` der Fall. Das einfachste XML-Dokument enthält in der ersten Zeile die XML-Deklaration, gefolgt von dem Root-Element, das wiederum weitere Elemente enthalten kann. In der XML-Deklaration wird dem XML-Prozessor (dem Programm, das XML liest und verarbeitet) unter anderem mitgeteilt, nach welcher Version der XML-Spezifikation das XML-Dokument zu interpretieren ist. Die zweite Zeile des Beispiels verweist auf seine DTD. Damit ist dieses Dokument *gültig* [Ecks00]. Abbildung 2.13 zeigt, wie eine DTD, in dem Fall für das obige XML-Dokument, zu erstellen ist.

```
1:    <!ELEMENT root (Name , Vorname , Adresse)*>
2:      <!ELEMENT Vorname (#PCDATA)>
3:      <!ELEMENT Name (#PCDATA)>
4:      <!ELEMENT Adresse (Strasse , Nummer , PLZ , Stadt)>
5:        <!ELEMENT Strasse (#PCDATA)>
6:        <!ELEMENT Nummer (#PCDATA)>
7:        <!ELEMENT PLZ (#PCDATA)>
8:        <!ELEMENT Stadt (#PCDATA)>
```

Abbildung 2.13: DTD für das Beispiel in Abbildung 2.12

Laut dieser DTD muss das Dokument mit dem Root-Element `root` beginnen, das aus den Kindelementen `Vorname`, `Name` und `Adresse` besteht. Die genannte Reihenfolge der Elemente muss dabei eingehalten werden. Der Stern am Ende der ersten Zeile bedeutet, dass die Element-Kombination in dieser Klammer beliebig oft vorkommen kann. Das Element Adresse selbst besteht aus den Elementen Strasse, Nummer, PLZ und Stadt, die wiederum vom Typ „#PCDATA“ sind, d. h. aus beliebigen Zeichenketten bestehen dürfen [BeMi00].

2.3.5.4 XML-Schema

Grundsätzlich werden XML-Dateien dazu verwendet, entweder Dokumente oder Daten zu speichern. Die bisher einzige Möglichkeit, die Bauweise einer XML-Datei festzulegen, besteht in der Zuweisung einer Document Type Definition. Wie der Name schon sagt, ist die DTD dafür ausgelegt, die Definition für Dokumente festzulegen, nicht für Daten. Die DTD prüft die Elemente lediglich auf Reihenfolge und Vollständigkeit im Dokument. Sie prüft nicht ihren Inhalt. Bei XML-Daten wäre allerdings eine zusätzliche Prüfung des Datentyps wünschenswert. Des Weiteren kann eine Prüfung der richtigen Formatierung oder des Wertebereichs notwendig sein (z. B. bei einem Datum) [BeMi00]. Das XML-Schema soll dieses Problem beheben, indem Definitionen bezüglich des Datentyps und des Wertebereichs ermöglicht werden.

2.3.5.5 eXtensible Stylesheet Language (Transformation)

Die Formatierungssprache eXtensible Stylesheet Language (XSL) erweitert die logische und physikalische Struktur eines XML-Dokuments um eine visuell-formale Struktur [Clar99]. Sie bestimmt, wie das Dokument in einer Umgebung dargestellt wird. Die ursprüngliche XSL ist in zwei Bereiche aufgeteilt worden. Ein Bereich beschäftigt sich mit der reinen Formatierung des Ausgabedokuments und wird im Allgemeinen unter der Bezeichnung XSL dargestellt. Der zweite Bereich dient dazu, XML-Dokumente von einer Struktur in eine andere zu überführen und läuft unter der Bezeichnung XSL-Transformation (XSLT) [Spen99]. Ein Stylesheet- Dokument kann neben der Beschreibung, wie die Daten dargestellt werden sollen, auch Regeln enthalten, welche Daten in den Ergebnisbaum überführt werden sollen, und zudem die Funktion eines Filters übernehmen.

Bei den Transformationen, die XSLT vornimmt, handelt es sich um die Überführung eines Quellbaums (source tree) in einen Ergebnisbaum (result tree). Die Anweisungen für die Transformation stehen in einem Stylesheet. Die Anweisungen für die Transformation sind eine Ansammlung von Regeln (template rules), die jeweils aus einem Muster (pattern) und einer Schablone (template) bestehen. Ein Stylesheet-Prozessor ersetzt den Knoten des Quellbaums, auf den das Muster passt, durch die Schablone und generiert daraus den Ergebnisbaum. Ein XML-Dokument kann in ein XML-Dokument mit einer komplett anderen Struktur überführt werden, um die Daten einer anderen Applikation zur Verfügung zu stellen, ohne dabei die Daten zu verändern oder gar zu verlieren. Auch die Generierung eines völlig anderen Dokumentformats (z. B. PDF) ist möglich [BeMi00]. Durch vielfältige Variationen in der Musterbeschreibung und die Möglichkeit einfacher Programmstrukturen können nicht nur Transformationen, sondern auch kleinere Algorithmen realisiert werden. Die Tabelle 2.4 fasst die Vor- und Nachteile von XML zusammen.

Tabelle 2.4: Vor- und Nachteile von XML bei der Übertragung von Daten

Vorteile	**Nachteile**
• Offenes Format • Systemunabhängig • Anpassbar • Unabhängigkeit von einem Ausgabeformat oder -gerät • Sowohl von Menschen als auch von Maschinen lesbar	• Hoher Initialaufwand • Laufende Entwicklung • Mangelhafte technische Realisierung (teilweise)

In diesem Kapitel wurden die wesentlichen Grundlagen vorgestellt, die für das Verständnis der vorliegenden Arbeit von großer Bedeutung sind. Der prinzipielle Aufbau eingebetteter Systeme und deren Unterteilung in zeit- und ereignisgesteuerte Systeme wurden erläutert. Im folgenden Kapitel werden die vorhandenen Konzepte für die systemunabhängige Übertragung von Prozessdaten diskutiert und bewertet.

3 Bestehende Konzepte für die systemunabhängige Übertragung von Prozessdaten

Die Vorstellung verschiedener Konzepte, die der systemunabhängigen Übertragung von Prozessdaten dienen, bildet den Gegenstand dieses Abschnitts. Diese Konzepte stellen den derzeitigen Stand der Technik dar. Sie wurden ursprünglich unabhängig von den Web-Technologien entwickelt, um Prozessdaten systemunabhängig für die Diagnose bzw. Visualisierung zur Verfügung zu stellen. Die Verbreitung von Web-Technologien in der Automatisierungstechnik nahm hierbei großen Einfluss und die Grundsteine für die Realisierung von Webbasierten Anwendungen waren gelegt.

Nach der Vorstellung der Konzepte werden verschiedene Bewertungskriterien eingeführt und ihre Eignung für den Einsatz in eingebetteten Systemen überprüft. Die Kriterien sollen beim Vergleich der Konzepte und für deren Einschätzung eine Hilfestellung bieten.

3.1 Aachener Prozessleittechnik / Kommunikationssystem

Das Aachener Prozessleittechnik / Kommunikationssystem (ACPLT/KS) wurde am Lehrstuhl für Prozessleittechnik (PLT) an der Rheinisch-Westfälischen Technischen Hochschule (RWTH) in Aachen konzipiert und entwickelt. Es ist ein offenes und freies Kommunikationssystem und bietet Zugriff auf Prozessdaten, Metainformationen und Objektverwaltung. Das ACPLT/KS dient der Verbindung von rechnergestützten Werkzeugen (im Bereich der Anwendung von Betriebs- und Prozessführungen) untereinander und mit Prozessleitsystemen. Modellgestützte Prozessführung, Online-Optimierung oder Messwertvalidierung sind typische Beispiele für solche Werkzeuge. ACPLT/KS lässt sich für den Betrieb in heterogenen Systemen verwenden, da es für den Nachrichtentransport auf bestehenden (Internet-)Standards aufbaut, wie beispielsweise TCP/IP und ONC/RPC. Außerdem bietet ACPLT/KS eine Anwendungsschicht mit einem Abstraktionsgrad, der den Aufgaben der Prozessleittechnik angepasst ist [Albr02a].

Das ACPLT/KS setzt auf einer Client/Server-basierten Kommunikation für den Datenaustausch auf. Die ACPLT/KS-Server dienen dabei zur Bereitstellung der angefragten Daten aus beispielsweise einem Prozessleitsystem oder einem Simulator. Die ACPLT/KS-Clients beanspruchen die Dienste eines oder mehrerer Server. Zum Beispiel können sie die aktuellen Prozessdaten abfragen und diese dann grafisch darstellen.

3.1.1 Kommunikationsarchitektur von ACPLT/KS

Wie bereits erwähnt, basiert das ACPLT/KS auf einer Client/Server-Architektur. Für die Kommunikation zwischen Client und Server wird das Protokoll TCP (Transmission Control Protocol) und für die Kommunikation innerhalb eines Rechners UDP (User Datagram Protocol) verwendet. Um die Dienste des Servers aufzurufen, wird die Remote-Procedure-Call- (RPC-) Methode verwendet. Es gibt drei Typen von Teilnehmern am KS-Protokoll:

- Manager
- Server
- Clients

Abbildung 3.1 zeigt die Teilnehmer am KS-Protokoll und deren Kommunikation.

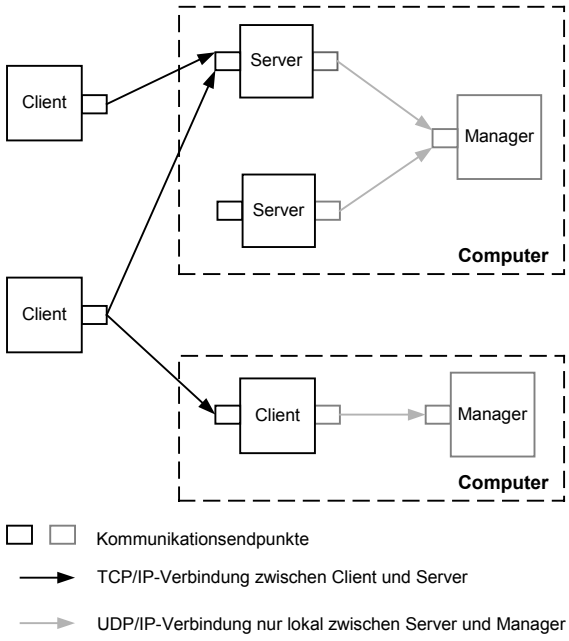

Abbildung 3.1: Infrastruktur der KS-Teilnehmer

Ein KS-Manager übernimmt die Verwaltung aller Server, die auf einem gemeinsamen Computer gestartet wurden. Er gibt den Kommunikationsendpunkt jedes KS-Servers allen anderen Teilnehmern bekannt. Damit ein Client eine TCP/IP-basierte Verbindung zu einem Server aufbauen kann, benötigt er folgende Informationen:

- IP-Adresse des Servers
- das einzusetzende Protokoll (TCP oder UDP)
- Portnummer
- KS-Protokollversion

Außerdem überwacht der KS-Manager, ob die Server zur Verfügung stehen. Falls er keine Rückmeldung von einem Server erhält, registriert er ihn zunächst als inaktiv und nach einer bestimmten Zeit als unerreichbar. Damit ist eine flexible und standardisierte Kommunikation zwischen Teilnehmern gewährleistet.

3.1.2 Datenmodell ACPLT/KS

Die Informationen werden in einer Baumstruktur von so genannten KS-Kommunikationsobjekten (kurz: Objekte) abgelegt, die beliebig tief sein darf. Damit wird eine flexible Abbildung von Anwendungssystemen auf Objekte erzielt [Albr02c]. Es existieren derzeit drei Typen von Kommunikationsobjekten (KS-Objekte):

- *Domain:* Container für beliebig viele KS-Objekte. Eine Domain kann auch beliebig viele Domains enthalten.
- *Variable:* Sie stellt beispielsweise aktuelle Zustandswerte eines Prozesses oder Prozessleitsystems dar.
- *History*: Sie erlaubt den Zugriff auf Zustandsarchive von Prozessleitsystemen.

Abbildung 3.2 zeigt, dass ein mögliches KS-Objektmodell eine hierarchische Strukturierung der Kommunikationsobjekte erlaubt. Die History-Objekte sind hier nicht von Bedeutung und werden daher nicht betrachtet.

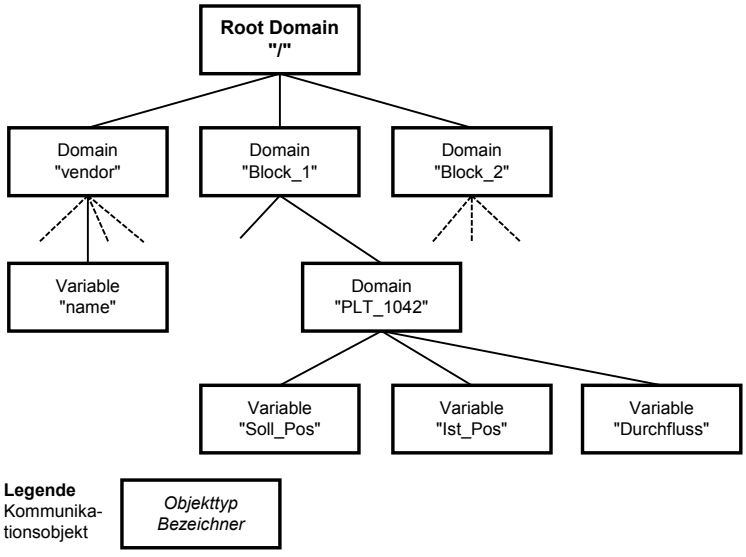

Abbildung 3.2: Hierarchische Strukturierung der Kommunikationsobjekte

Ein KS-Objekt wird durch einen Bezeichner, der bis zu 255 Zeichen lang sein darf, eindeutig identifiziert. Es wird immer über seinen vollständigen Namen adressiert, der sich aus den mit einem Schrägstrich „/“ verbundenen Bezeichnern aller übergeordneten Domains und dem eigenen Bezeichner zusammensetzt [Albr02c]. Jedes KS-Objekt besitzt entsprechend seines Typs

verschiedene Eigenschaften (engl. properties). Allgemein unterscheidet man zwischen *projektierten* und *aktuellen* Eigenschaften (vgl. Abbildung 3.3).

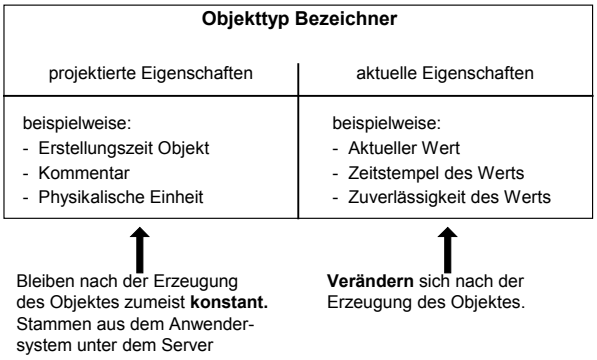

Abbildung 3.3: Eigenschaften eines KS-Kommunikationsobjekts

Die projektierten Eigenschaften werden bereits bei der Generierung eines Objekts festgelegt und bleiben im Laufe der Zeit unverändert. Die aktuellen Eigenschaften dagegen werden direkt aus dem KS-Server unterlagerten Anwendersystem generiert. Sie sind variabel und verändern sich im Laufe der Zeit.

Die Abbildung 3.4 zeigt die jeweiligen projektierten und aktuellen Eigenschaften von KS-Kommunikationsobjekten. Alle Kommunikationsobjekte in der Abbildung erben die projektierten und aktuellen Eigenschaften der jeweiligen Elternklasse, so dass diese Eigenschaften nicht erneut in der abgeleiteten Klasse aufgelistet werden [Albr02c].

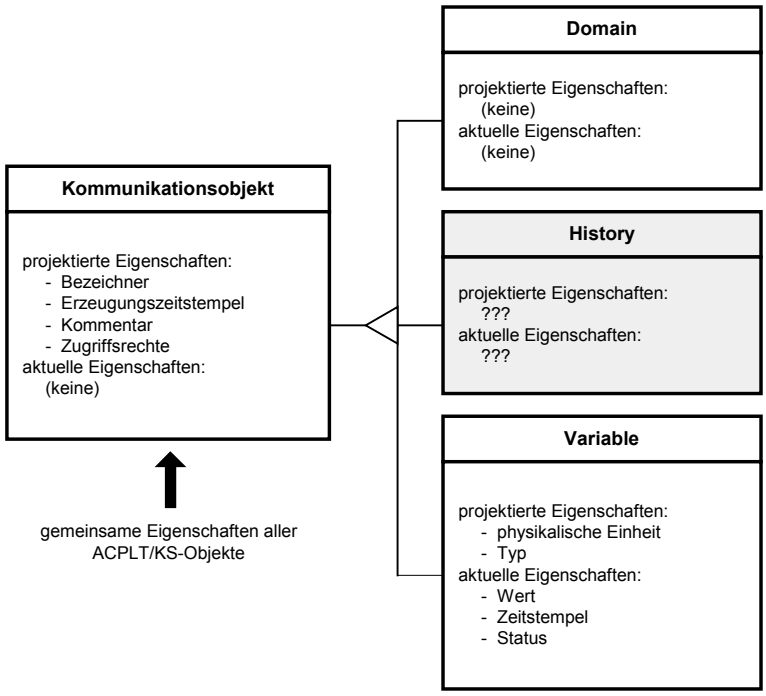

Abbildung 3.4: Projektierte und aktuelle Eigenschaften verschiedener KS-Objektklassen

ACPLT/KS basiert auf dem OSI-Schichtenmodell, benutzt jedoch nur die Schichten Applikation, Präsentation, Transport und Netzwerk. Es wendet ONC/RPC an, um den Nachrichtenaustausch auf Hochsprachenebene zu realisieren. Als Transportschicht wird das verbindungsorientierte TCP verwendet. Die Realisierung basiert also auf Standard-Technologien und führt somit zu einer Steigerung der Akzeptanz.

Das KS-Objektmodell orientiert sich an der Systemstruktur des Prozessleitsystems. Dabei hat jeder Server ein eigenes KS-Objektmodell. Ein Leitsystemrechner kann einen oder mehrere KS-Server enthalten, und es gibt keine Möglichkeit, serverübergreifende Objektmodelle zu definieren.

Das KS-Konzept wurde speziell für Belange der Anlagenautomatisierung entwickelt. Zur Realisierung der KS-Server bzw. -Clients werden viele Ressourcen und Rechenleistungen benötigt, die den Einsatz in eingebetteten Systemen unmöglich machen. Das Datenmodell-Konzept ist jedoch durchdacht und für die Entwicklung eines Verfahrens für eingebettete Systeme sehr innovativ nützlich.

3.2 OLE for Process Control

OLE for Process Control (OPC) ist eine Standardschnittstelle in der Automatisierungstechnik zur Erfassung von Prozessdaten. Mit Hilfe dieser Schnittstelle ist eine Interoperabilität zwischen Anwendungen und Kommunikationssystemen verschiedener Hersteller gewährleistet. Die OPC-Schnittstelle ermöglicht die einheitliche Anbindung von Hardware an die Software und einen geräteunabhängigen Zugriff auf Prozessdaten. OPC sorgt für einen effizienten Datenfluss zwischen Windows-Applikationen und Automatisierungsgeräten.

Ursprünglich stand die Abkürzung OLE für *Object Linking and Embedding* und verkörperte die Verbunddokument-Technik. Diese Technik ermöglicht die Einbettung von Dokumenten in andere, wie beispielsweise die Einbettung einer Excel-Tabelle in ein Word-Dokument. Diese neuartige Interprozesskommunikation hat letztlich zur Entwicklung der COM-Technik geführt. Im Laufe der Zeit wurde fast alles, was mit COM zu tun hatte, als *OLE* bezeichnet. Mittlerweile bezeichnet *OLE* jedoch wieder nur die Verbunddokument-Technik, da sich der Begriff ActiveX als Oberbegriff der COM/DCOM-Technologien durchgesetzt hat. Trotzdem ist *OLE* in diesem Zusammenhang als Oberbegriff für die COM/DCOM-Technologien anzusehen. Dabei ist OPC eine Schnittstellenspezifikation, die auf dem Komponentenmodell COM basiert.

Die OPC-Spezifikation definiert u. a. eine Komponente, die ganz bestimmte, in der OPC-Spezifikation festgelegte Leistungen zu erbringen hat. Eine solche Komponente wird **OPC-Server** genannt. Jedes Gerät wird nach außen durch seinen OPC-Server repräsentiert. Da alle OPC-Server die gleichen Schnittstellen und Schnittstellenfunktionen haben, werden unterschiedlichste Geräte nach außen einheitlich repräsentiert. Eine Anwendung, die die Dienste eines OPC-Serv-

ers nutzen kann, wird OPC-Client genannt. Die Kommunikation zwischen dem OPC-Client und dem OPC-Server erfolgt nach dem COM/DCOM-Standard.

Die Definition der OPC-Schnittstelle besteht aus einer Reihe von Spezifikationen, die für bestimmte Aufgaben in der Automatisierungstechnik vorgesehen sind. Zur Zeit liegen die folgenden Spezifikationen vor:

- Data Access Specification: beschreibt den Datenaustausch zwischen OPC-Server und Clients.
- Alarm & Events Specification: dient der Behandlung von Alarmen und Ereignissen.
- Historical Data Access Specification: erlaubt den Zugriff auf Zustandsarchive von Prozessleitsystemen.
- OPC Batch Specification: repräsentiert eine Definition von Ergänzungen von OPC-Data Access für das Batchprocessing [IwLa01].
- OPC Security Specification: Ein OPC-Client kann über einen OPC-Server den technischen Prozess beeinflussen. Die OPC Security Specification beschreibt Möglichkeiten und definiert Funktionalitäten, die das Realisieren flexibler Sicherheitskonzepte ermöglichen.
- OPC Compliance Test Specification: sorgt für die Sicherheit der Programmierung.

3.2.1 Anwendung der OPC-Schnittstelle

Der Hersteller eines Geräts, das Prozessdaten liefert, stellt den OPC-Server für das Gerät zur Verfügung. Der OPC-Server realisiert die Anbindung an das Gerät und repräsentiert es über die Standardschnittstellen nach außen. Der OPC-Server kapselt somit alle herstellerspezifischen Details vom Anwender ab. Der Anwender, der nun mit einem OPC-Server kommunizieren will, entwickelt die OPC-Client-Schnittstelle in seiner Anwendung. Abbildung 3.5 zeigt das Prinzip der Datenerfassung mit OPC.

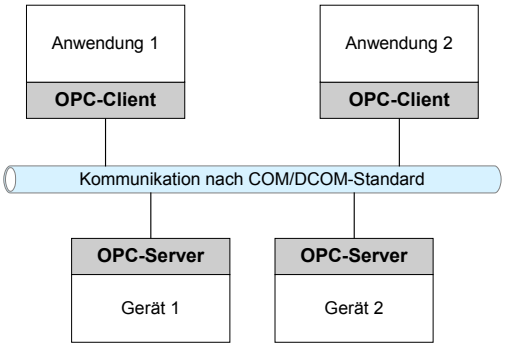

Abbildung 3.5: Datenerfassung mit OPC

Bei der Datenerfassung mit OPC stellt jedes Gerät einen OPC-Server zur Verfügung, auf den jeder OPC-Client zugreifen kann. Wie aus Abbildung 3.5 hervorgeht, implementiert jede Anwendung, die auf Prozessdaten zugreifen will, einen OPC-Client. Jeder dieser OPC-Clients kann auf jeden OPC-Server zugreifen. Dabei können die OPC-Clients und OPC-Server entweder auf dem gleichen Rechner oder verteilt auf verschiedenen Rechnern installiert sein. Ein OPC-Client greift auf die Prozessdaten immer auf die gleiche Art und Weise zu, unabhängig davon, welche Hardware sich hinter einem OPC-Server verbirgt, oder an welchem Ort sich ein OPC-Server befindet. Der Abbildung ist weiter zu entnehmen, dass jeder OPC-Client gleichzeitig mit mehreren OPC-Servern kommunizieren kann, wie auch jeder OPC-Server mehrere OPC-Clients bedienen kann. Die Integration neuer Software- und Hardwarekomponenten in das bestehende Automatisierungssystem wird mit Hilfe dieser Art der Datenerfassung mit geringem Aufwand möglich.

Es gibt viele Arten von OPC-Clients. Der Funktionsumfang eines OPC-Clients kann beispielsweise bei jeder Anwendung anders sein. Es gibt einfache und komplexe OPC-Clients. Ein einfacher OPC-Client kann z. B. nur mit einem ganz bestimmten OPC-Server kommunizieren. Denkbar sind auch OPC-Clients, die mit nur genau einem OPC-Server kommunizieren können, der aber beliebig gewählt werden kann. Schließlich gibt es OPC-Clients, die mit allen und mit beliebig vielen OPC-Servern Daten austauschen können.

Für die Entwicklung eines OPC-Clients stehen dem Anwender viele mächtige Werkzeuge zur Verfügung, die den Entwicklungsprozess vereinfachen und damit beschleunigen.

3.2.2 Technische Grundlagen

3.2.2.1 OPC-Schnittstellenarten

Die OPC-Spezifikation beschreibt zwei Arten von Schnittstellen, über die ein OPC-Server den OPC-Clients Zugang zu den Prozessvariablen ermöglicht. Die erste Schnittstellenart ist das so genannte *OPC Custom Interface*. Diese Schnittstelle ist eine reine COM-Schnittstelle, die jeder OPC-Server unterstützen muss. Diese Schnittstelle ist für die Kommunikation mit C++-Anwendungen gedacht. Die zweite Schnittstellenart ist eine Automatisierungsschnittstelle, die in erster Linie für die Kommunikation mit VisualBasic-Anwendungen gedacht ist. Diese Schnittstelle ist allerdings gemäß der OPC-Spezifikation optional und muss somit nicht von jedem OPC-Server unterstützt werden (siehe Abbildung 3.6).

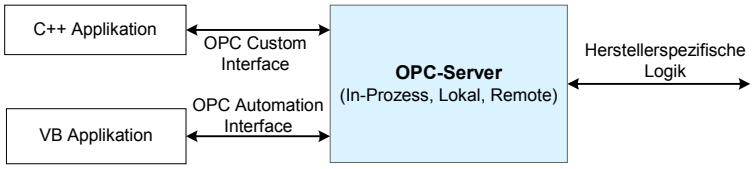

Abbildung 3.6: Schnittstellen eines OPC-Servers

In manchen Fällen liefert der Hersteller eines OPC-Servers, der nicht über die Automatisierungsschnittstelle verfügt, zusätzlich einen Schnittstellenwandler (wrapper) für die Umwandlung des *Custom interface* in eine Automatisierungsschnittstelle. Damit entsteht eine Schnittstellenarchitektur nach Abbildung 3.7. In der Praxis hat sich dieser Ansatz allerdings nicht bewährt. Der Stand der Technik ist weiterhin so, dass VisualBasic-Anwendungen nicht mit einem OPC-Server kommunizieren können, wenn er nicht über die Automatisierungsschnittstelle verfügt.

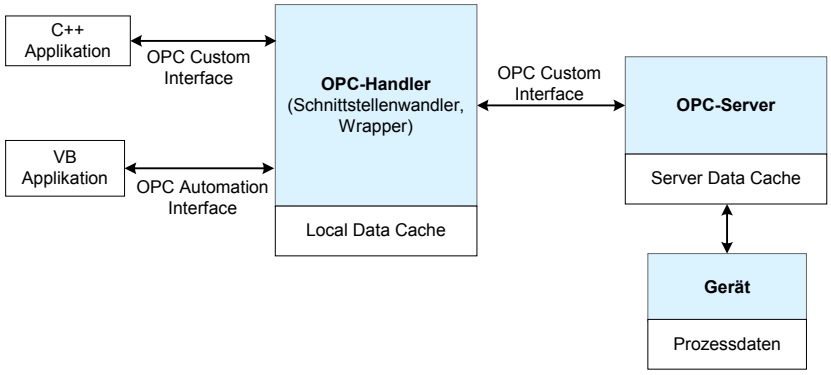

Abbildung 3.7: OPC-Architektur mit Schnittstellenwandler

3.2.2.2 Das Objektmodell

Die OPC-Spezifikation definiert folgende drei Komponenten:

- OPC-Server
- OPC-Group
- OPC-Item

Diese drei Komponenten haben eine hierarchische Beziehung zueinander und bilden gemeinsam den OPC-Server. Abbildung 3.8 zeigt diese Objekthierarchie. An oberster Stelle der Hierarchie steht das Objekt OPC-Server, gefolgt von einem oder mehreren Objekten der Klasse OPC-Group. Auf der untersten Ebene der Hierarchie befinden sich schließlich Objekte der Klasse OPC-Item. Objekte der Klasse OPC-Server verwalten die Instanzen der untergeordneten Klasse OPC-Group. Objekte der Klasse OPC-Group verwalten wiederum die Instanzen der Klasse OPC-Item.

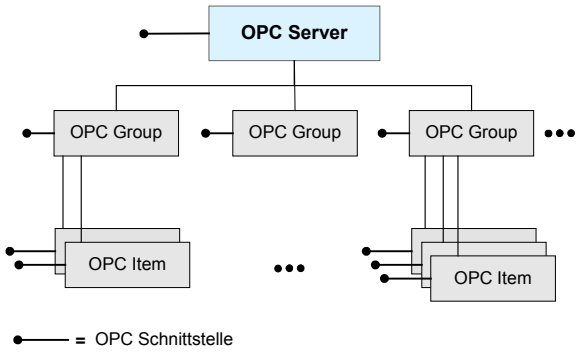

Abbildung 3.8: Objektmodell der OPC Data Access Spezifikation

In den folgenden Abschnitten werden die einzelnen Komponenten dieses Modells erläutert.

OPC-Server

Ein Objekt der Klasse OPC-Server repräsentiert ein bestimmtes Gerät und fungiert als dessen OPC-Server. Die Komponente OPC-Server verfügt über verschiedene Schnittstellen, die beispielsweise Informationen über den Status oder optional den Adressraum eines OPC-Servers liefern. Der Adressraum eines OPC-Servers ist die Summe aller Prozessdaten (Items), die ein OPC-Server zur Verfügung stellt. Die Objekte der Klasse OPC-Group werden ebenfalls über die Schnittstellen des OPC-Server-Objekts verwaltet. Abbildung 3.9 zeigt ein Objekt der Klasse OPC-Server.

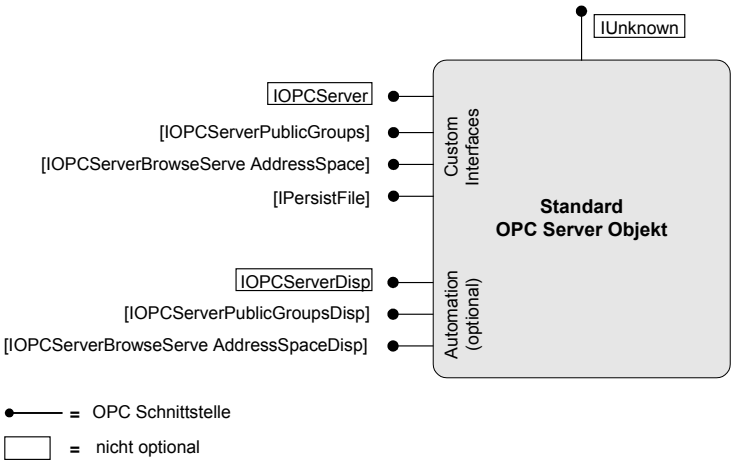

Abbildung 3.9: OPC-Server-Objekt mit dazugehörigen Schnittstellen

Das OPC-Server-Objekt besitzt viele optionale Schnittstellen, die ein OPC-Server nicht implementieren muss. Ein OPC-Client darf also nicht von diesen Schnittstellen abhängig sein. Die optionalen Automatisierungsschnittstellen sind in Abbildung 3.9 in eckigen Klammern dargestellt. Sollte ein OPC-Server seine Dienste über Automatisierungsschnittstellen anbieten, dann muss er nur eine Schnittstelle implementieren, das *IOPCServerDisp*, die restlichen sind optional. Die einzige Schnittstelle, die ein OPC-Server-Objekt implementieren muss, ist ein *IOP-*

CServer. Die Funktionen dieser Schnittstelle dienen hauptsächlich der Verwaltung der untergeordneten Objekte der Klasse *OPC-Group.*

OPC-Group

Objekte der Klasse OPC-Group strukturieren die Prozessdaten (Items) eines OPC-Servers. Ein OPC-Server kann seine Prozessdaten mit Hilfe der Objekte dieser Klasse in Gruppen zusammenfassen. So könnte beispielsweise ein OPC-Server eines Geräts, das über Ventile und Pumpen verfügt, seine Prozessdaten in folgende Gruppen einteilen:

- Wasserpumpen
- Luftpumpen
- Hochdruckventile
- Niederdruckventile

Ein OPC-Client fasst mit Hilfe der Objekte dieser Klasse die Prozessdaten in Gruppen zusammen, die aber nicht identisch mit denen eines OPC-Servers sein müssen. In einer Gruppe eines OPC-Clients können verschiedene Prozessdaten von verschiedenen OPC-Servern gebündelt werden. Ein OPC-Client hat die Möglichkeit, z. B. die Prozessdaten, die auf einem Screen in einem Visualisierungssystem erscheinen, in einer Gruppe abzuschließen. Die gleichen Prozessdaten können auch in verschiedenen Gruppen vertreten sein.

Die Komponente OPC-Group definiert einen Satz Schnittstellen, mit deren Funktionen hauptsächlich Prozessdaten gelesen und geschrieben werden können. Dabei können mehrere Prozessdaten gleichzeitig gelesen und geschrieben werden. Abbildung 3.10 zeigt ein Objekt der Klasse OPC-Group mit allen möglichen Schnittstellen. Die optionalen Schnittstellen sind in der Abbildung mit eckigen Klammern versehen, wobei alle Automatisierungsschnittstellen optional sind. Wird die Funktionalität über die Automatisierung angeboten, so sind die Schnittstellen in den eckigen Klammern wiederum optional.

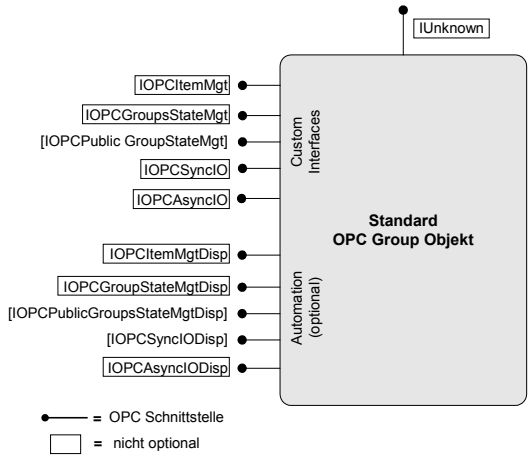

Abbildung 3.10: OPC-Group-Objekt mit dazugehörigen Schnittstellen

OPC-Item

Ein Objekt der Klasse OPC-Item repräsentiert eine Prozessvariable. Jede physikalische bzw. erfassbare Größe eines Geräts, also eine Prozessvariable, wird im OPC-Server durch ein Objekt der Klasse OPC-Item repräsentiert. Alle OPC-Item-Objekte bilden zusammen den so genannten Adressraum eines OPC-Servers. Ein OPC-Item wird im OPC-Server durch eine Item-ID eindeutig gekennzeichnet. Eine Item-ID ist ein vom Hersteller des OPC-Servers festgelegter Name für eine Prozessvariable, der innerhalb des Adressraums eines OPC-Servers eindeutig ist.

Jedes OPC-Item besitzt die Eigenschaften Wert, Qualität und Zeitstempel. Der Wert des OPC-Items ist der Messwert der mit ihm verknüpften Prozessvariablen. Die Qualität eines OPC-Items gibt an, ob der ermittelte Wert gültig ist. Der Zeitstempel gibt den Zeitpunkt der Ermittlung des Wertes an. Abbildung 3.11 zeigt ein Objekt der Klasse OPC-Item. Das Objekt besitzt eine einzige Schnittstelle.

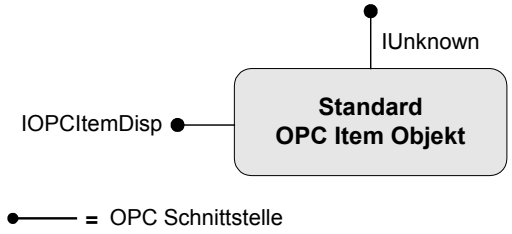

Abbildung 3.11: OPC-Item-Objekt mit dazugehörigen Schnittstellen

Die Schnittstelle besitzt nur zwei Funktionen, OPCRead und OPCWrite, um Prozessdaten zu lesen und zu schreiben.

3.2.2.3 Die Interaktion zwischen OPC-Server und OPC-Client

Da ein OPC-Client in der Praxis sehr viele Prozessvariablen, oft tausende, verwalten muss, wird er nach der OPC-Architektur durch den OPC-Server bei der Überwachung der Prozessvariablen entlastet. Der OPC-Client muss nicht ständig Prozessvariablen beim OPC-Server abfragen. Stattdessen meldet der OPC-Client einmalig die Prozessvariablen, die für ihn relevant sind, beim OPC-Server an und startet die Kommunikation. Der OPC-Server überprüft nun sowohl den Wert als auch die Qualität der angemeldeten Prozessvariablen zyklisch und übermittelt sie dem OPC-Client. Die Datenübermittlung erfolgt nur dann, wenn sich entweder die Qualität oder der Wert geändert hat. Die Kommunikation zwischen OPC-Client und OPC-Server erfolgt dabei gruppenweise und nicht itemweise. Das heißt, der OPC-Server liefert immer eine komplette OPC-Group und nicht einzelne OPC-Items.

Der Zyklus, in dem die Daten übermittelt werden, wird dabei vom OPC-Client festgelegt. Damit ist die Kommunikation zwischen OPC-Server und OPC-Client auf das Nötigste beschränkt.

Der Hersteller eines OPC-Servers kann auf ein von der OPC-Spezifikation optional definiertes Verfahren zurückgreifen, um die Projektierung eines OPC-Clients zu erleichtern. Dieses Verfahren ist als *Browsing* bekannt und ist in einer optionalen Schnittstelle, *IOPCBrowseServerAddressSpace*, realisiert. Das Browsing ermöglicht dem OPC-Client den Adressraum des OPC-Servers zu durchsuchen. Die Suche im Adressraum bedeutet, dass der Client ermittelt, welche OPC-Groups und OPC-Items in einem OPC-Server zur Verfügung stehen. Der OPC-Client kann dann relativ einfach auf eine Teilmenge des Adressraums zugreifen.

3.3 Industrielle Standardisierung

Es gibt viele Bemühungen seitens der Unternehmen, den Datenzugriff auf die Feldebene zu standardisieren. Marktbeherrschende Firmen versuchen ihre eigene Vorstellung von Standardisierung zum De-facto-Standard zu erklären. PROFInet repräsentiert so eine Vorstellung. Es stellt ein offenes und durchgängiges Konzept für die vertikale Integration von PROFIBUS dar. Damit ist es möglich, aus der Office-Welt direkt auf die Prozess- und Automatisierungsstrukturen inklusive der Feldebene zuzugreifen [Prof02]. PROFInet ist eine Proxy-Lösung, die den Einsatz von Ethernet und PROFIBUS parallel ermöglicht.

Ein weiterer Standardisierungsversuch wird durch die IDA[4]-Gruppe verfolgt. IDA besteht aus einer Entwicklungsgruppe mittelständischer Unternehmen, die zur Zeit einen Standard für verteilte Intelligenz auf Basis Ethernet und TCP/IP entwickelt [Krau01]. Sie versucht alle notwendigen Protokolle und Schnittstellen zwischen Geräten und Softwarewerkzeugen zu definieren. Sie haben zwar eine ausführliche Spezifikation herausgegeben, die konkrete Umsetzung fehlt jedoch bis dato.

Das amerikanische Unternehmen „Rockwell Automation“ definiert seine Vorstellung in EtherNET/IP, wobei EtherNET/IP für Ethernet Industrial Protocol steht. Das Unternehmen setzt bei industriellen Netzwerken auf die Vorteile der offenen Netzwerke DeviceNet (Gerätenetzwerk) und ControlNet (Steuerungsnetzwerk), die auf dem Producer-/Consumer-Datenübertragungsmodell basieren [Rock02]. Das Konzept EtherNET/IP übernimmt diese Dienste für Ethernet. Dieser Netzwerkstandard unterstützt die implizite Nachrichtenübertragung (Übertragung von E/A-Nachrichten in Echtzeit) und die explizite Nachrichtenübertragung (Nachrichtenaustausch). EtherNet/IP ist ein offenes Netzwerk, das auf bereits vorhandener handelsüblicher Technologie aufbaut.

---

[4] Interface for Distributed Automation

3.4 Bewertung der vorgestellten Ansätze

Die spezifischen Merkmale jedes Lösungsansatzes bilden die Grundlage für die Entscheidung darüber, ob das Konzept für die Übertragung der Prozessdaten eingebetteter Systeme geeignet ist. Von großer Bedeutung sind außerdem die Kosten für die Realisierung einer Anwendung. Nachfolgend werden einzelne Charakteristika der unterschiedlichen Ansätze diskutiert und ein Vergleich durchgeführt.

Schnittstelle zum eingebetteten System

Es gibt zwei Voraussetzungen für den Einsatz von Web-Technologien bei eingebetteten Systemen, zum einen den Zugriff auf Prozessdaten und zum anderen den Internet-Zugang. Das bedeutet, es gibt zwei Schnittstellen, die eine zum Gerät und die andere zu einem Netzwerk. Genau an dieser Stelle wird entschieden, ob ein Konzept zum Erfolg oder Misserfolg führt, denn beim Einsatz von Web-Technologien wird der Großteil der Kosten für die Realisierung dieser Schnittstellen eingesetzt. Die herstellerspezifischen Schnittstellen sind kostenanfällig und unflexibel. Das Konzept ACPLT/KS wurde auf Basis der Client/Server-Architektur realisiert, wobei der Server ein Rechner mit großer Rechenleistung ist. Dieses Konzept wurde für die Belange von Automatisierungsanlagen entwickelt. Hierbei sind die zusätzlichen Kosten im Vergleich zu denen einer Automatisierungsanlage vernachlässigbar. Dies würde allerdings bei eingebetteten Systemen den mehrfachen Kosten eines Gerätes entsprechen. Das gleiche Argument gilt für OPC. Hier wird vorausgesetzt, dass ein OPC-Server für jedes Gerät gekauft wird. Abgesehen von den hohen Kosten, gibt es diese Server derzeit nur für drei Feldbussysteme. Die industriellen Bestrebungen nach einer Standardisierung dieser Schnittstellen ist zwar eine hervorragende Idee, sie führt aber seit mehreren Jahrzehnten zu keiner Einigung, da die wirtschaftlichen Folgen solcher Aktivitäten nicht zu unterschätzen sind.

Verteilbarkeit von Software

Neben der guten Beherrschung der Komplexität bietet die Aufteilung der Software in Schichten den großen Vorteil, dass diese in einem Netzwerk auf beliebige Knoten verteilt werden können. Hierbei müssen die beteiligten Protokolle in der Lage sein, die Netzwerkverteilung und Interoperabilität zu unterstützen [MGSD01]. Durch die Verteilung kann eine bessere Ausnutzung der Rechenleistung von mehreren Rechnern anstelle eines zentralen Rechners mit großer Rechenleistung und eine günstigere Lastverteilung im Netzwerk erreicht werden. Außerdem steigen die Ausfallsicherheit und die Verfügbarkeit der Software. Durch die Auslagerung rechenintensiver Prozesse eines eingebetteten Systems auf verteilte Knoten in einem Netzwerk kann die geringe Rechenleistung solcher Systeme kompensiert werden. ACPLT/KS unterstützt mit seiner Client/Server-Architektur diese Vorgehensweise. Das Problem ist hierbei die Server-Applikation, die sehr rechenintensiv ist. Die Verteilbarkeit bei der OPC wurde durch Einführung der DCOM- anstatt der COM-Technologie realisiert. Bei den industriellen Standardisierungen entstehen in der Regel durchgängige Anwendungen, bei denen die Interoperabilität innerhalb der Anwen-

dung zwar gegeben ist, der vorgegebene Rahmen allerdings nicht verlassen werden darf, da sonst bei der Kommunikation mit anderen Anwendungen große Schwierigkeiten auftreten. Der Grund dafür ist der Einsatz spezieller Protokolle, die sehr häufig nicht mit Standardprotokollen kommunizieren können.

Bedienungssoftware

Die meisten Fernservice-Applikationen haben ihre eigene Benutzungsoberfläche. Diese Software beinhalten viele Funktionen, ermöglichen aber keine intuitive Bedienung. Die Benutzungsoberflächen stellen Bedienungsmöglichkeiten für verschiedene Anwendergruppen zur Verfügung, die in der Regel aber unterschiedliche Bedürfnisse haben: Ein Applikationsingenieur möchte Zugang zu allen Features haben, ein Wartungstechniker dagegen braucht nur bestimmte Bedienelemente, um seine Arbeit durchführen zu können. Die Benutzung dieser Software ist daher meist mit großem Einarbeitungsaufwand verbunden. Außerdem muss sie zunächst auf einem Rechner installiert werden. Nach der Installation ist der Anwender dann gezwungen, genau diesen einen Rechner zu benutzen, d. h. es besteht diesbezüglich keine Flexibilität. Eine Browser-Technologie bietet dagegen eine rechner- und ortsunabhängige Bedienung einer Fernservice-Applikation, wie Ferndiagnose oder Fernvisualisierung. Das ACPLT/KS unterstützt diese Anforderungen, da mit diesem Verfahren die Prozessdaten einer Automatisierungsanlage zur Verfügung gestellt werden und dann beliebig weiter verarbeitet werden können. OPC bietet zwar auch die Möglichkeit, eine webbasierte Benutzungsoberfläche (OPC-Client) zu entwickeln, die starke Abhängigkeit von Microsoft-Technologien beschränkt jedoch die Erweiterbarkeit. Industrielle Anwendungen setzen auf ihre eigenen spezifischen Benutzungsoberflächen.

In der Tabelle 3.1 werden alle betrachteten Konzepte für die systemunabhängige Übertragung der Prozessdaten gegenübergestellt. Sie werden anhand der beschriebenen Bewertungskriterien zusammenfassend bewertet. Jedes Konzept wird danach beurteilt, ob das zugrunde gelegte Kriterium gut (+), neutral (0) oder schlecht (-) erfüllt wird.

Tabelle 3.1: Übertragungskonzepte im Vergleich

	ACPLT/KS	**OPC**	**Industrielle Ansätze**
Schnittstelle zum eingebetteten System	-	-	-
Verteilbarkeit	+	0	0
Bedienungssoftware	+	0	-

Das größte Problem bei einer systemübergreifenden Übertragung der Prozessdaten eingebetteter Systeme ist die Schnittstellenproblematik, wie dies aus Tabelle 3.1 zu erkennen ist. Der Zugriff

auf Prozessdaten der Geräte und der Internet-Zugang verursachen die meisten Kosten, was den Einsatz dieser Technologien in diesem Anwendungsgebiet erschwert.

Als Fazit kann festgestellt werden, dass durch die hohen Kosten bei der Realisierung von Fernservice-Applikationen für eingebettete Systeme der Einsatz dieser Anwendungen bis dato keine große Verbreitung gefunden hat. Die Bewertung der vorgestellten Konzepte hat ergeben, dass die Schnittstellenproblematik die größte Herausforderung beim Einsatz von Web-Technologien bei eingebetteten Systemen ist. Diese Schnittstellenproblematik wird im nächsten Kapitel aufgegriffen und intensiv diskutiert.

4 Web-Technologien bei eingebetteten Systemen

In diesem Kapitel werden Probleme und Schwierigkeiten erläutert, die aus heutiger Sicht mit dem Einsatz von Web-Technologien bei eingebetteten Systemen einhergehen. Ein besonderes Augenmerk soll dabei auf die Ursachen dieser Probleme gelegt werden. Weiterhin werden Lösungen zur Problembeseitigung vorgestellt und bewertet. Abschließend werden Anforderungen an einen Lösungsansatz für den Einsatz von Web-Technologien bei eingebetteten Systemen formuliert.

4.1 Einsatz-Problematik

Das Internet stellt eine Kommunikationsinfrastruktur zum gleichberechtigten Informationsaustausch zur Verfügung, analog zum Telefonnetz, welches eine Infrastruktur für die Sprachkommunikation bietet [Balz99]. Der Einsatz dieser weltweiten Kommunikationsinfrastrukturen bei eingebetteten Systemen bietet zahlreiche technologische und wirtschaftliche Vorteile. Das Internet und die damit verbundenen Web-Technologien ermöglichen einen globalen Zugriff auf einzelne Prozessdaten wie Sensor- oder Aktordaten, und zwar lesend sowie schreibend. Diese Informationen können für die Ferndiagnose und Fernvisualisierung bzw. Fernwartung und Fernbedienung von jedem beliebigen Ort zu jeder beliebigen Zeit verwendet werden.

Das Problem bei der Wartung eingebetteter Systeme ist offensichtlich. Wenn beispielsweise ein Fernseher oder eine Waschmaschine defekt ist, muss entweder das Gerät zur Wartungsfirma gebracht werden oder ein Service-Techniker das Problem vor Ort beheben. Der Ablauf beim Ausfall eines Geräts ist dabei häufig ähnlich: Zu Anfang versucht ein Techniker das Problem vor Ort zu diagnostizieren und zu lokalisieren. Nach der Problemerkennung ist häufig das passende Ersatzteil nicht vorhanden, was zu einer Wartezeit führt. Wenn das eingebettete System z. B. in einer Produktion eingesetzt wird, könnte dieser Umstand fatale Folgen für das Unternehmen haben, da der Stillstand der Produktion mit großen Einnahmenverlusten verbunden wäre. Die Wartezeiten für die Instandsetzung könnten durch eine Ferndiagnose zumindest verkürzt werden. Durch die Ferndiagnose ist eine Erkennung des Problems über das Internet möglich. Falls das Problem die Software betrifft, ist eine Behebung aus der Ferne denkbar. Im Falle eines notwendigen Vor-Ort-Services könnte vorab per Ferndiagnose das passende Ersatzteil ermittelt und gleich mitgenommen werden. Die Wartungsarbeiten würden verkürzt und unnötige oder doppelte Fahrten zum Kunden vermieden, was die Kosten u. U. erheblich senkt.

Durch den Einsatz von Web-Technologien wird eine globale Bereitstellung von Dienstleistungen wie Diagnose, Wartung, Software-Update etc. möglich. Leider sind diese Vorzüge durch Schwierigkeiten belastet, die sich im Einsatz von Web-Technologien bei eingebetteten Syste-

men offenbaren. Es können vier grundlegende Problemkategorien identifiziert werden, die mit dem Einsatz von Web-Technologien bei eingebetteten Systemen verbunden sind:

- *Geringer Speicher:* Den eingebetteten Systemen steht in der Regel sehr wenig Speicher zur Verfügung. Wegen hoher Stückzahlen fallen Entwicklungskosten von Software und Hardware kaum ins Gewicht, der Stückpreis wird maßgeblich durch Hardware-Kosten bestimmt. Jedes Bit Speicherplatz mehr würde den Preis unweigerlich in die Höhe treiben.

- *Geringe Rechenleistung:* Bei der Entwicklung von eingebetteten Systemen werden meist 8- bzw. 16-Bit-Mikrocontroller mit einer Taktfrequenz von in der Regel kleiner als 40 MHz verwendet. Die Hauptaufgabe eingebetteter Systeme ist die Durchführung bestimmter Automatisierungsaufgaben, für die diese Mikrocontroller zugeschnitten sind. Der Einsatz leistungsfähigerer Mikrocontroller mit großer Rechenleistung wird bewusst vermieden, da dies zur Steigerung des Stückpreises führt.

- *Kompaktheit:* Eingebettete Systeme sind meist kleine Geräte, die physikalisch sehr kompakt gebaut sind. Sie können daher nicht ohne weiteres erweitert werden.

- *Bus-Ankopplung:* Der Einsatz von Feldbussystemen bei eingebetteten Systemen gewinnt immer mehr an Bedeutung. Damit wird eine Möglichkeit für den Zugriff auf Prozessdaten für die Diagnose oder Wartung dieser Systeme gegeben. Für die Bus-Ankopplung wird aber spezielle Hardware benötigt, die mit hohen Kosten verbunden ist.

4.2 Probleme derzeitiger Fernservice-Applikationen

Viele Unternehmen haben die enorme wirtschaftliche Bedeutung der Web-Technologien in diesem Sektor erkannt und versuchen, ihre Geräte mit einer „Internet-Schnittstelle“ zu versehen. Für diese Geräte werden derzeit verschiedene Fernservice-Applikationen entwickelt, um Ferndiagnose, Fernvisualisierung, Fernwartung und Fernbedienung über das Internet zu ermöglichen. Dies ist jedoch zur Zeit noch wenig verbreitet, da die entwickelten Anwendungen die oben erwähnten Nachteile aufweisen. Die bereits existierenden Softwarelösungen sind kostenintensiv und in ihren Möglichkeiten eingeschränkt. Die schwerwiegendsten Probleme, die die Lösungen mit sich bringen, werden im Folgenden diskutiert.

Proprietäre Lösungen

Viele bereits entwickelte Fernservice-Applikationen wurden für spezielle eingebettete Systeme oder spezifische Feldbussysteme entwickelt. Dies zeigt sich in erster Linie darin, dass die Hardwareankopplung Gerät- bzw. Feldbus-spezifisch angelegt ist. Die Fernservice-Applikation ist dagegen an den Client angepasst. Für ein neues eingebettetes System muss daher die Fernservice-Applikation mit großem Aufwand einerseits an das Gerät und andererseits an die Anwenderseite (Client) angepasst werden – falls das überhaupt möglich ist. Sehr häufig haben solche

Änderungen einen neuen Entwurf und eine neue Entwicklung der Fernservice-Applikation zur Folge. Aus wirtschaftlicher Sicht hat dies schwerwiegende Folgen.

Kostenintensiver Zugriff auf Prozessdaten

Viele Fernservice-Applikationen sind für große Anlagen entwickelt worden, wobei für die Anbindung an den technischen Prozess ein PC verwendet wird. Die dabei entstehenden Kosten sind im Vergleich zum Preis einer Automatisierungsanlage vernachlässigbar gering. Die eingebetteten Systeme gehören jedoch zur unteren bis mittleren Preisklasse. Der Einsatz eines PCs, z. B. neben einer Waschmaschine für die Internet-Anbindung, käme daher nicht in Frage. Die Fernservice-Applikationen, die eine große Rechenleistung in Anspruch nehmen, lassen sich jedoch nicht ohne weiteres auf einen kleinen Mikrocontroller integrieren.

Spezielle Software für die Anwenderseite

Bei vielen Fernservice-Applikationen wird spezielle Software für den Anwender entwickelt, die auf dessen Rechner installiert sein muss, bevor er in der Lage ist, über das Internet eine Verbindung zum eingebetteten System herzustellen. Die Applikationen werden proprietär realisiert, wobei teilweise sehr spezifische Formate für die Übermittlung und Verwaltung der Prozessdaten verwendet werden, weshalb sich eine Weiterverarbeitung der Prozessdaten recht schwierig gestaltet. Für die Darstellung der Daten kommt neben spezieller Software häufig auch individuell gestaltete Hardware (z. B. spezielle Bedienpanels) zum Einsatz, die meist um ein Vielfaches teurer ist als Standard-Hardware.

Keine Erweiterbarkeit

Proprietäre Realisierungen von Fernservice-Applikationen erschweren bzw. verhindern deren Erweiterung oder Anpassung an spezielle Bedürfnisse der Anwender (z. B. ein Wartungsingenieur) dieser Dienste. Der Einsatz solcher proprietären Lösungen bindet den Anwender an herstellerspezifische Software und fordert für jede Erweiterung den Kauf einer neuen Software. Besonders kritisch wird das Ganze, wenn die Software nicht mehr gewartet wird oder wenn ein neuer Release freigegeben wird, der aber abwärts nicht kompatibel ist.

4.3 Problemursachen

Fernservice-Applikationen sorgen auf der einen Seite für einen kontrollierten Zugriff auf die Prozessdaten des eingebetteten Systems und bereiten auf der anderen Seite die Informationen für eine Fernabfrage des Anwenders (Clients) vor. Der Anwender muss dafür auf dem Rechner eine auf die Fernservice-Applikation zugeschnittene Spezialsoftware installieren, um über das Internet Informationen abfragen zu können. Abbildung 4.1 verdeutlicht die Problematik der derzeit auf dem Markt erhältlichen Lösungen.

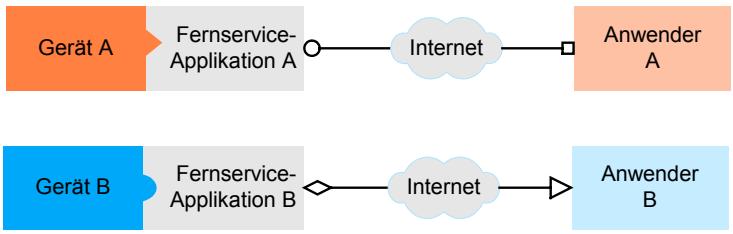

Abbildung 4.1: Probleme bestehender Fernservice-Applikationen

Die Fernservice-Applikation ist einerseits an das eingebettete System und andererseits an die spezielle Anwender-Applikation angepasst. Kleine Firmen, die nur einen Teil, z. B. eine Anwender-Applikation oder eine Fernservice-Applikation, vertreiben möchten, müssen sich an den Schnittstellen anderer Anbieter orientieren. Eine Standardisierung ist nicht abzusehen, denn es fehlt einerseits eine allgemeingültige Schnittstelle für den Zugriff auf Prozessdaten und andererseits eine für die Anwender-Applikationen. Die Anbindung eingebetteter Systeme ist nicht vereinheitlicht und somit weder flexibel noch einfach. Dabei können Informationen, die aus dem eingebetteten System gewonnen wurden, für unterschiedliche Clients (z. B. PC oder Personal Digital Assistent, PDA) nicht auf die gleiche Art und Weise aufbereitet werden.

Ein weiteres Problem ist die notwendige Hardware für die Prozessanbindung. Sie wird meist in Form eines PCs oder Embedded PCs realisiert, was aus Kostensicht nicht akzeptabel ist. Aus diesem Grunde muss auf der Seite des eingebetteten Systems eine flexible, an Randbedingungen dieser Systeme angepasste und kostengünstige Fernservice-Applikation entwickelt werden. Der Anwender soll in der Lage sein, mit möglichst wenig Aufwand auf das eingebettete System zuzugreifen, um die gewünschten Informationen zu erhalten.

4.4 Bestehende proprietäre Lösungsansätze

Es gibt eine Vielzahl von herstellerspezifischen Lösungen, die die Verwendung bestimmter Hardware und Software voraussetzen. Im Folgenden werden drei relativ verbreitete Lösungen vorgestellt.

4.4.1 S7 Simatic

SIMATIC PCS 7 ist das Prozessleitsystem im Siemens Automatisierungskonzept „Totally Integrated Automation“. Es wurde speziell für den PROFIBUS entwickelt. Es erfüllt alle Anforderungen der verfahrenstechnischen Branchen und basiert vollständig auf SIMATIC S7-Standardkomponenten wie Controller, PC, Kommunikation und dezentrale Peripherie. Damit wird eine umfassende Durchgängigkeit in einer Anlage erreicht und alle verfahrens- und fertigungstechnischen Aufgaben können bearbeitet werden. In der Feldebene lässt sich eine breite Produktpalette an normgerechten Feldgeräten (PROFIBUS-DP/PA, HART) in SIMATIC PCS 7 integrieren [Gieß01]. Abbildung 4.2 zeigt die Integration von PROFIBUS-PA in SIMATIC PCS7.

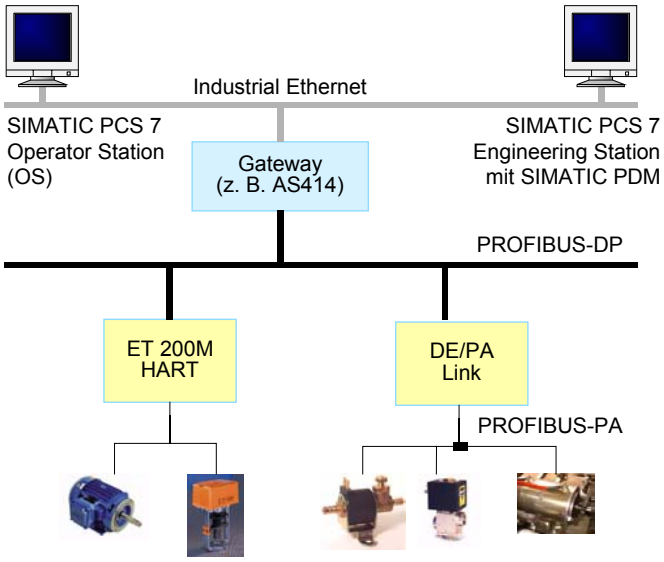

Abbildung 4.2: Integration von PROFIBUS-PA in SIMATIC PCS 7

Die Kommunikation zwischen Automatisierungssystem und Feldgeräten an PROFIBUS-PA erfolgt über Gateway-Bausteine in der CPU.

SIMATIC PDM

SIMATIC PDM (Process Device Manager) ist ein durchgängiges Werkzeug zur Projektierung, Parametrierung, Inbetriebnahme und Diagnose intelligenter Feldgeräten und Komponenten. SIMATIC PDM ermöglicht es, mit einer Software eine Vielzahl von Prozessgeräten unter einer einheitlichen Bedienoberfläche zu projektieren. Es ist auf zwei Arten einsetzbar [Müll02]:

- Als integraler Bestandteil des Engineeringsystems von SIMATIC PCS 7 unter Windows NT sowie
- unabhängig vom Systemlieferanten auf einem PC/PG unter Windows 95/98, Windows NT oder Windows 2000.

Die Darstellung der Geräteparameter und -funktionen ist für alle unterstützten Prozessgeräte einheitlich und unabhängig von ihrer Kommunikationsanbindung, z. B. über PROFIBUS-DP/PA oder HART-Protokoll. Die Kernfunktionen von SIMATIC PDM sind:

- Einstellen und Ändern von Geräteparametern
- Vergleichen von Soll- und Ist-Parametern
- Prüfen auf Plausibilität der Eingaben
- Simulieren
- Diagnose
- Verwalten
- Inbetriebsetzungsfunktionen, z. B. Messkreistests von Prozessgerätedaten

Zusätzlich ermöglicht SIMATIC PDM auch eine Online-Beobachtung von angewählten Werten, Alarmen und Zustandssignalen des Gerätes.

Zentrale Gerätebedienung

Mit SIMATIC PDM kann das sogenannte Routing genutzt werden. Routing ermöglicht die Kommunikation von einer zentralen Engineering-Station bis zu den Prozessgeräten. Alle Gerätedaten können von einem zentralen Punkt aus erreicht werden. Insbesondere die Diagnosedaten der Geräte sind sofort verfügbar. Auf der Feldebene sind Geräte unterschiedlich anschließbar: PROFIBUS PA-Geräte an DP/PA-Koppler und DP/PA-Link oder HART-Geräte an den HART-Analogeingabe-/Analogausgabebaugruppen der ET 200M oder ET 200iS. An herkömmlichen Analogeingabe-/Analogausgabebaugruppen sind ebenfalls HART-Geräte anschließbar.

Kommunikation

SIMATIC PDM unterstützt mehrere Kommunikationsprotokolle und -komponenten zur Kommunikation mit folgenden Geräten:

1. Geräte mit PROFIBUS-DP-Interface, die direkt am PROFIBUS-DP angeschlossen sind. Ein Beispiel hierfür ist der Kompaktregler SIPART DR20.
2. Geräte mit PROFIBUS-PA-Interface, z. B. der Messumformer SITRANS P.
3. Remote I/O Stationen, z. B. ET 200M: Die von SIMATIC PDM unterstützten PROFIBUS-PA-Geräte werden über DP/PA-Link oder DP/PA-Koppler an das PROFIBUS-DP-Segment angebunden. Parametrierbar sind die vollintegrierten PROFIBUS-PA-Geräte sowie nahezu alle PROFIBUS-PA-Geräte, die über folgende PA-Profile der Version 2.0 und 3.0 verfügen:

    - Druck und Temperatur
    - Füllstand und Durchfluss
    - Aktoren
    - Diskret-I/O
    - Analysegeräte (nur PA-Profil Version 3.0)

4. Geräte mit HART-Interface. Diese Geräte können auf unterschiedliche Arten angeschlossen werden:

    - über die dezentrale Peripherie SIMATIC ET 200M mit den HART-Baugruppen
    - über die dezentrale Peripherie ET 200iS mit HART on PROFIBUS gemäß PNO
    - über ein HART-Modem, mit dem eine Punkt-zu-Punkt-Verbindung zwischen PC bzw. Engineering Station und dem HART-Gerät aufgebaut wird und
    - über HART-Multiplexer, die im HART-Server der HCF (HART Communication Foundation) enthalten sind.

Parametrierbar sind neben den vollintegrierten HART-fähigen Geräten auch alle weiteren HART-Geräte.

5. Regler mit serieller SIPART DR-Kommunikation über RS 232 / RS 485: Über das sog. SIPART DR-Netz sind die Regler der SIPART-Reihe parametrierbar.

Device Description Language

SIMATIC PDM unterstützt alle Feldgeräte entsprechend den Profilbeschreibungen durch die PROFIBUS Nutzerorganisation (PNO). In SIMATIC PDM werden laufend Prozessgeräte von Siemens integriert. Die Hersteller müssen ihre Geräte in SIMATIC PDM integrieren, um dies nutzen zu können. Für die Parametrierung von HART-Geräten existiert die Device Description Language (HART-DDL). Diese Sprache ist eine Richtlinie der HCF (Hart Communication Foundation); sie ist genormt, herstellerunabhängig und sehr weit verbreitet. Die Beschreibung der PROFIBUS-Geräte erfolgt durch die Electronic Device Description Language (EDDL). Anhand dieser Beschreibungen erstellt SIMATIC PDM automatisch die Bedienoberfläche.

4.4.2 EMIT™ (Embedded Micro Internetworking Technology™) der Firma emWare™

EMIT-Produkte werden für die Realisierung von Fernservice-Applikationen bei eingebetteten Systemen eingesetzt. Sie bilden ein Framework, das den kompletten Anwendungsbereich vom Gerät bis hin zur Anwenderapplikation abdeckt. Bei der Realisierung dieser Lösung wurde ganz bewusst auf eine schlanke Anbindungskomponente geachtet, die auf dem Gerät installiert werden muss. Abbildung 4.3 zeigt die Architektur der EMIT-Lösung.

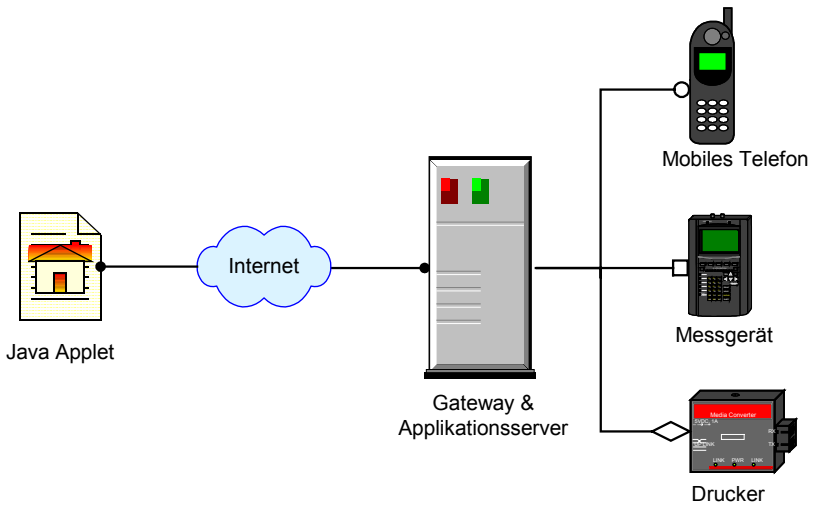

Abbildung 4.3: Systemarchitektur von EMIT

Pro Gerät wird eine herstellerspezifische Hardware-Komponente eingesetzt, die nach Programmiervorschriften programmiert werden muss. Auf dieser Hardware-Komponente ist ein soge-

annter *Micro Webserver* implementiert, der mit einem Proxy-Webserver mittels eines firmenspezifischen Protokolls kommuniziert. Nachdem die Anbindungshardware und der Proxy-Webserver implementiert wurden, kann von einem PC-Client mittels Java-Applet auf die Prozessdaten zugegriffen bzw. eine Fernservice-Applikation gestartet werden.

EMIT besteht aus den folgenden Modulen:

- emMicro
- emGateway
- Webbrowser mit emObjects (Java Beans)
- EMIT Acess Library

Im Folgenden wird die Funktionalität einzelner Module detailliert dargestellt.

emMicro

emMicro ist ein spezifischer und schlanker HTML-Webserver, der es ermöglicht, auf elektronische Geräte oder Systeme mit Hilfe eines Webbrowsers von einem entfernten Rechner aus zuzugreifen und diese zu steuern. Die Schnittstelle zwischen *emMicro* und der Geräte-Applikation bilden Tabellen mit Variablen und Funktionsaufrufe. Die Applikation benötigt etwa 30 Byte RAM und 1 KB ROM bei einer MC-8051 Applikation und erlaubt den Zugriff auf Gerätedaten. Die Implementierung in der Programmiersprache C ermöglicht eine einfache Portierung auf gängige Mikrocontroller. Bei der Entwicklung wurden so genannte *„Microtags“* definiert, die eine Referenzierung auf Objekte der Applikationsschnittstelle in komprimierter Form ermöglichen. *Microtags* werden auf dem Gerät gespeichert und vom *emGateway* expandiert, um Speicherresourcen auf dem Gerät zu sparen. Für die Kommunikation zwischen *emMicro* und *emGateway* wurde das Modul *emNet* realisiert. Dieses beinhaltet zwar eine Vielzahl von Protokollen, muss jedoch kundenspezifisch erstellt werden, um die jeweiligen Geräte in bestehenden Netzwerken an das *emGatway* anzubinden.

emGateway

emGateway stellt ein Framework für Netzwerke eingebetteter Systeme bereit und dient zugleich als Gateway für große, leistungsfähige Netzwerke, wie z. B. das Internet. Es unterstützt unterschiedliche Transport-Protokolle, wie RS232, RS4885 und CAN. Für die Kommunikation mit dem Gerät muss auch hier ein kundenspezifisches Modul (Device Link Modul – DLM) für jedes Gerät neu implementiert werden. *emGateway* verwaltet den Datenfluss vom und zum Gerät und Browser bzw. von und zur Kommandozeilen-Eingabe und kundenspezifische Applikationen. *emObjects* und EMIT-spezifische Java Klassendateien lösen im Web-Browser über HTTP-Kommandos einen Start von *emGateway*-Diensten aus und stellen die nötige Benutzerschnittstelle bereit, um die Kommunikation mit dem *emMicro* zu ermöglichen. Diese *emObjects* und Java-Klassendateien werden auf dem *emGateway*-Host gespeichert, um die Kommunikation mit solchen Computern zu ermöglichen, die keine *emClients* sind [Willi00]. Der HTTP-Server emp-

fängt vom *emClient* Startbefehle zum Verbindungsaufbau. Er sendet Adresse und Informationen zum Verbindungsaufbau zurück, die für eine direkte Kommunikation mit dem *emGateway* benötigt werden. Der Browser richtet eine Datenverbindung zwischen dem EMIT Java Runtime Interface (*EmitJri*) des Browsers und dem *emGateway* ein. Der HTTP-Server überträgt auf Anforderung auch HTML-Seiten zum Browser und ersetzt vorher die enthaltenen Microtags.

Webbrowser

Der Webbrowser stellt die nötige Benutzungsschnittstelle zur Kommunikation mit *emMicro* zur Verfügung. Der Webbrowser nutzt *emObjects* und EMIT-spezifische Java-Klassendateien zum Verbindungsaufbau und zum Start von *emGateway*-Diensten mit HTTP-Kommandos. *emClients* sind Rechner mit Webbrowsern, die lokal auf die *emObjects* und Java-Klassendateien zugreifen. *emClients* können einen anfänglich schnelleren Aufbau der Benutzungsschnittstelle vorweisen als Webbrowser, die sich diese Clientdateien erst via Netz vom *emGateway*-Rechner laden müssen. Ein Universal Resource Locator (URL) wird vom Browser zum *emGateway* gesandt. Die URL enthält Informationen über die Adresse des Gateways, den Anschluss, das Gerät und kundenspezifische Dokumente. *emGateway* holt sich das HTML-Dokument von *emMicro*. Dokumente, die Microtags enthalten, werden expandiert. Sind die vom Browser benötigten *emObjects* nicht lokal verfügbar, werden sie beim *emGateway* angefordert und dann übermittelt. *emObjects* sind grafische oder funktionelle Hilfsmittel, die konfiguriert und angepasst werden können, um eine intuitive und interaktive Oberfläche zu erstellen. Die Oberfläche kann dem tatsächlichen Bedienpanel optisch gleichen, wenn das Applet im Browser ein naturgetreues Bild versendet. Der Browser initiiert daraufhin für den Datenaustausch eine Verbindung zwischen *EmitJri* und *emGateway* und eine Verbindung zum HTTP-Server für die HTML-Seiten auf dem Gerät. Variablen, Funktionen, Ereignisse und Dokumente des Geräts werden vom Webbrowser über *emObjects* angesprochen und angezeigt. Interaktionen des Anwenders mit den *emObjects* werden zum *emGateway* und von dort zu *emMicro* gesandt. Änderungen im elektronischen Gerät werden unmittelbar angezeigt.

EMIT Access Library

Die EMIT Access Library ist eine Sammlung von Software-Hilfsmitteln, die die EMIT-Technologie mit zusätzlicher Funktionalität versehen. Hat ein Kunde spezielle Anforderungen, die den Einsatz eines Webbrowsers oder die Eingabe von Kommandozeilen ausschließen, dann kann eine Applikation auch mit gängigen Programmiersprachen erstellt werden. Dadurch wird eine direkte Kommunikation mit dem *emGateway* ermöglicht.

Der grobe Vorteil dieser Lösung ist ihre Durchgängigkeit. Die verschiedenen Komponenten sind aufeinander abgestimmt und die Datenübertragung zwischen ihnen erfolgt problemlos. Ein schwerwiegender Nachteil ist dagegen der Einsatz spezifischer Protokolle, die von Standardprotokollen nicht unterstützt werden, d. h. diese Lösung ist mit Anbindungskomponenten anderer Hersteller nicht kompatibel. Außerdem muss der Proxy-Server für jeden Einsatz neu pro-

grammiert werden. Die Implementierung ist auch mit großem Aufwand verbunden, da herstellerspezifische Programmiervorschriften beachtet werden müssen. Zur Bedienung muss ein Applet lokal installiert werden, wobei der Anwender-Rechner wiederum bestimmte Voraussetzungen erfüllen muss, damit das Applet auf dem lokalen Rechner funktionsfähig ist. Ein weiterer wichtiger Nachteil dieser Lösung ist die Tatsache, dass sie mit Werkzeugen anderer Hersteller nicht kompatibel ist. Danach ist der Anwender beim Einsatz dieser Lösung absolut abhängig von den Vorschriften eines speziellen Herstellers.

4.4.3 OPC-basierte Fernservice-Applikationen

OPC ist eine Softwareschnittstellendefinition in der Microsoft-Welt, um Windows-Anwendungen den Zugang zu Daten der Automatisierungstechnik zu erleichtern und zu standardisieren (siehe auch 3.2). Die Windows-Anwendung, die die Daten zwecks Visualisierung, Archivierung, Erstellung von Berichten usw. benötigt, wird Client-Anwendung genannt. Das Programm, das die Daten bereitstellt, heißt Server-Anwendung. Ein Client kann auf mehrere Server zugreifen, und ein Server kann wiederum an verschiedene Clients Daten senden. Client und Server können auf demselben oder auf unterschiedlichen Rechnern laufen. Letzteres bedingt die Kommunikation über ein geeignetes Netzwerk, wie z. B. LAN (Local Area Network) oder das Internet.

Der große Vorteil dieser Lösung ist die Standardschnittstelle zwischen zwei Windows-Applikationen. Damit hat OPC einen großen Einfluss auf automatisierungstechnische Softwarebausteine unter Windows. Im Rahmen der PC-basierten Steuerung und Regelung spielt OPC eine große Rolle, da hier die Programme, die die Steuerung und Regelung durchführen, ebenfalls Windows-Applikationen sind.

Um OPC für den Zugriff auf Prozessdaten eines eingebetteten Systems einsetzen zu können, muss ein OPC-Server für den speziellen Feldbus angeschafft werden. Der OPC-Server läuft auf einem PC, der mittels einer zusätzlichen Karte mit dem Feldbus kommuniziert. Die Kosten eines OPC-Servers erschweren den Einsatz im Bereich eingebetteter Systeme. Außerdem sind die Anwendungen durch den Einsatz des OPC von Microsoft-Technologien abhängig.

4.4.4 Bewertung der Lösungsansätze

Zur Bewertung der vorgestellten Ansätze wurden folgende vier Kriterien, die sich aus der Beschreibung der Probleme derzeitiger Fernservice-Applikationen im Abschnitt 4.2 ergaben, herangezogen:

- Herstellerunabhängigkeit
- Systemunabhängigkeit
- Anwenderspezifische Software

- Erweiterbarkeit durch Anwender

Während die oben beschriebenen Bemühungen für den Zugriff auf Prozessdaten eingebetteter Systeme und die Bereitstellung der Informationen für den Anwender über das Internet interessante Lösungen und zum Teil in sich durchgängige Anwendungen realisieren, so wird die Schnittstellenproblematik nicht in vollem Umfang gelöst. Simatic und EMIT™ aus der Industrie stellen durchgängige Lösungen dar. Diese Durchgängigkeit wird jedoch unterbrochen, sobald die Grenzen dieser Lösungen überschritten werden. Bei Industrielösungen ist der Anwender auf Software bestimmter Hersteller angewiesen. Der Anwender ist gezwungen, auf seinen Rechner spezifische Software zu installieren, um auf Informationen aus eingebetteten System zugreifen zu können. Außerdem sind die Beschaffungskosten meistens zu hoch, um einen Einsatz bei kleineren Preisklassen von eingebetteten Systemen zu rechtfertigen. OPC bietet hingegen ein allgemeingültiges Konzept, allerdings nur für die Microsoft-Welt und für eine begrenzte Anzahl von Feldbussystemen. Die Anschaffungskosten eines OPC-Servers erschweren den Einsatz bei eingebetteten Systemen. Tabelle 4.1 stellt die Vor- und Nachteile der vorgestellten Lösungsansätze gegenüber.

Tabelle 4.1: Bewertung der Fernservice-Lösungsansätze

	S7 Sematik	**EMIT**	**OPC**
Herstellerunabhängigkeit	-	+	Server-Seite: - Anwenderseite: +
Systemunabhängigkeit	-	-	Falls Feldbusankopplung möglich +
Spezielle Anwendersoftware	-	-	-
Erweiterbarkeit durch Anwender	-	-	+
Kosten	-	+	-

4.5 Anforderungen an Lösungsansätze für den Einsatz von Web-Technologien bei eingebetteten Systemen

Für die Realisierung von Fernservice-Applikationen für eingebettete Systeme mittels Web-Technologien kann die notwendige Software und Hardware in zwei Bereiche unterteilt werden. Auf dem eingebetteten System wird bestimmte Software und Hardware benötigt, die zwei Aufgaben zu erfüllen hat. Einerseits müssen sie den Zugriff auf die Prozessdaten des Gerätes ermöglichen, andererseits einen Internet-Zugang realisieren. Auf der Anwenderseite wird ein Internet-Zugang vorausgesetzt. Außerdem ist eine Schnittstelle sowohl für die Kommunikation mit dem eingebetteten System über das Internet als auch für die Darstellung der Informationen notwendig. Im Folgenden werden die Anforderungen an die Prozessanbindung und die Anwenderseite zusammengefasst.

4.5.1 Anforderungen an die Prozessanbindung eingebetteter Systeme

Aus dem vorherigen Abschnitt ist zu entnehmen, dass bestehende Ansätze entweder eine durchgängige, dafür aber herstellerabhängige Lösung oder lediglich Teillösungen für die Prozessanbindungs- oder Anwenderseite anbieten. Im ersten Fall hat der Anwender keine Möglichkeit, die Informationen weiterzuverarbeiten. Zudem ist er gezwungen, die Bedienung bestimmter herstellerspezifischer Software zu beherrschen, damit er zur Information gelangt. Die Lösungen sind meist anwendungsspezifisch aufgebaut, für verschiedene Fernservice-Applikationen wie Visualisierung oder Bedienung wird unterschiedliche Software benötigt und zudem fallen Kosten an. Der OPC-Lösungsansatz beinhaltet ein sehr durchdachtes Datenmodell. Der relativ teure Server auf Seite des eingebetteten Systems und die Entwicklung spezifischer Anwendersoftware erschweren jedoch dessen Einsatz bei eingebetteten Systemen. Aus den oben beschriebenen Vorteilen und Problemen aktueller Ansätze leiten sich folgende Anforderungen für ein umfassendes Gesamtkonzept ab, das die Unterstützung des Anwenders gewährleistet.

- *Kostengünstige Prozessanbindung:* Die Hardware und Software für die Prozessanbindung an ein eingebettetes System soll möglichst schlank und kostengünstig sein.
- *Internet-Zugang:* Es sollen entsprechende Hardware und Software für einen Internet-Zugang vorhanden sein. Der Internet-Zugang soll unterschiedliche Kommunikationsprotokolle unterstützen.
- *Verlagerung der rechenintensiven Applikationen:* Die Fernservice-Applikationen benötigten vom eingebetteten System nur die Prozessdaten. Alle weiteren Informationen, wie History-Daten und rechenintensive Applikationen, sollen daher aus dem Gerät ausgelagert werden.

4.5.2 Anforderungen an die Schnittstelle zum Anwender

Ein wichtiger Faktor für den Erfolg eines Lösungsansatzes ist neben einer schlanken Prozessanbindung die Unterstützung des Anwenders. Sobald sich der Anwender zur Bedienung des Werkzeugs mit Programmierdetails der Software auseinandersetzen muss, sinkt seine Motivation für dessen Einsatz drastisch. Aus diesem Grund werden folgende Anforderungen aus der Sicht des Anwenders zusammengefasst:

- *Keine zusätzliche Installation:* Es soll beim Anwender kein zusätzlicher Aufwand an Installation von Hard- und Software notwendig sein. Er soll mit gängigen Browsern in der Lage sein, Informationen zu erhalten.
- *Intuitive Bedienung:* Die Software soll auf Standard-Bedienelementen des Browsers aufbauen, damit der Anwender die Software ohne zusätzliche Lerneinheiten bedienen kann.

- *Rechnerunabhängige Bedienung:* Der Anwender soll in der Lage sein, mittels unterschiedlicher Bediengeräte (PC, Personal Digital Assistent etc.) auf gleiche Art und Weise zu Informationen zu gelangen.

- *Keine unnötige Information:* Der Anwender soll nur die Informationen zu sehen bekommen, die für ihn gedacht sind. Das bedeutet, dass alle überflüssigen Informationen, die er nicht nutzen möchte bzw. brauchen kann, verborgen bleiben.

In diesem Kapitel wurden die Schwierigkeiten untersucht, die sich einerseits aus den spezifischen Randbedingungen eingebetteter Systeme ergeben und andererseits aus der Sicht des Anwenders an die Fernservice-Applikationen gekoppelt sind. Es wurden bestehende Lösungsansätze aus der Literatur vorgestellt und bewertet. Auf der Grundlage dieser Betrachtungen wurde deutlich, dass bestehende Lösungsansätze nicht oder kaum auf die Randbedingungen eingebetteter Systeme zugeschnitten sind. Auch die Bedienung der Software ist nicht intuitiv. Der Anwender muss auf seinem Rechner spezifische Software des Herstellers installieren, teilweise sogar selber entwickeln, sich mit der Software vertraut machen, und erst danach ist er in der Lage, zu Informationen aus dem Gerät zu gelangen.

Aus diesem Grund wird ein neues Lösungskonzept benötigt, das einerseits den Randbedingungen eingebetteter Systeme gerecht wird und andererseits dem Anwender in vollem Umfang Unterstützung bietet. Nachfolgend wird das Konzept *Universelle Fernservice Infrastruktur* als eine neue Vorgehensweise für die Entwicklung von Fernservice-Applikationen vorgestellt, das das besondere Augenmerk auf die Belange eingebetteter Systeme und Unterstützung des Anwenders legt.

5 Konzept für eine universelle Fernservice-Infrastruktur

Im Folgenden wird das Konzept der universellen Fernservice-Infrastruktur eingeführt und erläutert. Es wird erklärt, warum das bloße Bereitstellen von Prozessdaten mit entsprechenden Werkzeugen für eine effiziente Unterstützung des Anwenders nicht ausreicht. Daneben wird beschrieben, warum dieses Konzept auf die Drei-Schichten-Architektur aufbaut und welche Merkmale kennzeichnend sind.

5.1 Drei-Schichten-Architektur

Die unterschiedlichen Komponenten einer *verteilten* (Web)-Anwendung sind auf mehrere Computer eines Netzwerks verteilt. Diese Verteilung kann nach verschiedenen Konzepten erfolgen. Ein gebräuchliches Entwurfsmuster ist die Drei-Schichten-Architektur (*three-tier architecture*), die ihren Ursprung in der MVC-Architektur (*Model-View-Control*) hat und erstmals bei Samlltalk-80 Verwendung fand. Die MVC-Architektur besteht aus den drei Klassen *Model*, *View* und *Controller* [KrPo88]. Das *Model*-Objekt ist über die Art und Weise der Präsentation der Daten auf Benutzerebene nicht informiert. Es ist nicht erlaubt, auf assoziierte Objekte (*View* und *Controller*) direkt zuzugreifen und Daten zu ändern. Wenn *View*- oder *Controller*-Objekte geändert oder ausgetauscht werden, ist das *Model*-Objekt nicht betroffen [Pösc01]. Die Drei-Schichten-Architektur besteht aus folgenden Schichten:

- Benutzungsoberfläche
- Eigentliche Anwendung (Fachkonzept-Schicht)
- Datenhaltung

Innerhalb einer Schicht können die Komponenten beliebig aufeinander zugreifen und Daten austauschen. Für die Kommunikation zwischen den Schichten gelten allerdings strenge Zugriffsregeln. Die Benutzungsoberflächen-Schicht hat in dieser Architektur zwei unterschiedliche Aufgaben. Zum einen ist sie für die Darstellung der Information zuständig und zum anderen für die Kommunikation mit der Fachkonzept-Schicht. Bei der Mehr-Schichten-Architektur werden diese zwei Aufgaben getrennt, die Benutzungsoberflächen-Schicht übernimmt nur die Präsentation der Information und eine separate Zugriffsschicht die Kommunikation mit der Fachkonzept-Schicht [Fowl97]. Die Fachkonzept-Schicht repräsentiert den funktionalen Kern der Anwendung. Sie ist außerdem für den Zugriff auf die Datenhaltungs-Schicht zuständig. In der Datenhaltungs-Schicht wird die Datenspeicherung, z. B. in Form einer Datenbank, realisiert.

Eine optimale Schichten-Architektur realisiert folgende Entwurfsziele [Balz99a]:

- *Wiederverwendbarkeit:* Jede Schicht besitzt eine präzis definierte Aufgabe und Schnittstelle. Eine vorhandene Schicht kann entweder direkt wieder verwendet werden oder sie wird um eine weitere Schicht ergänzt, die zusätzlich benötigte Aufgaben durchführt.
- *Änderbarkeit/Wartbarkeit:* Solange die Schnittstelle einer Schicht unverändert bleibt, kann deren interne Organisation beliebig verändert werden, ohne dass sich die Änderungen außerhalb der Schicht auswirken.
- *Portabilität:* Hardware-Abhängigkeiten können in einer Schicht isoliert und modifiziert werden, ohne den Rest des Systems zu tangieren.

Eine solche Architektur ermöglicht es, die einzelnen Schichten je nach Bedarf zwischen Client und Server aufzuteilen. Bei der Client/Server-Verteilung erfolgt manchmal noch eine Aufteilung des Fachkonzepts in zwei Bereiche [Balz99]. Abbildung 5.1 zeigt, wie eine allgemeine Drei-Schichten-Architektur auf Client und Server verteilt werden kann. In diesem Fall sind die zwei Schichten *Fachkonzept* und *Datenhaltung* auf einem Server implementiert. Es ist denkbar, diese zwei Schichten auf zwei unabhängige Server zu verteilen.

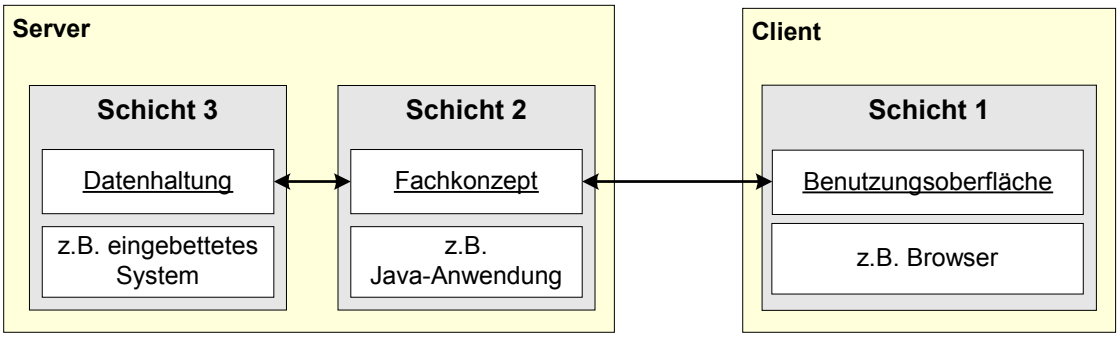

Abbildung 5.1: Verteilung bei einer Client/Server-Architektur

In der Regel wird in der verteilten Web-Anwendung die Benutzungsoberfläche auf einem separaten Client-Computer abgelegt, damit sich viele unabhängige Clients das gleiche Fachkonzept auf einem Server teilen. Im Internet wird der Webbrowser als Benutzungsoberfläche verwendet. Dies bedeutet, dass er auf einem TCP/IP-Netzwerk läuft und das HTTP-Protokoll verwendet, um mit dem Server zu kommunizieren. Das HTTP-Protokoll verwendet HTML, XML und andere Standardformate für Daten, die zwischen Server und Client ausgetauscht werden [NaMy00].

5.2 Beschreibung der universellen Fernservice-Infrastruktur

Im Kapitel 4.5 wurden die Anforderungen an einen Lösungsansatz für den Einsatz von Web-Technologien bei eingebetteten Systemen formuliert. Dabei wurden die Anforderungen in zwei Gruppen unterteilt, die Anforderungen an die Prozessanbindung zum eingebetteten System und zur Anwender-Schnittstelle. Bei genauer Betrachtung wird deutlich, dass die vorgestellte *Drei-Schichten-Architektur* eine hervorragende Basis für die Realisierung solcher Lösungsansätze darstellt. Im Folgenden werden die Realisierungsmöglichkeiten der wichtigsten Anforderungen durch diese Architektur im Detail formuliert:

- *Verlagerung der rechenintensiven Applikationen:* Eingebetteten Systemen stehen wenig Rechenleistung und wenig Speicher zur Verfügung. Daraus entsteht der Grundgedanke der Auslagerung rechenintensiver Applikationen aus dem Gerät. Übertragen auf die Drei-Schichten-Architektur bedeutet dies, dass das eingebettete System als Datenhaltungs-Schicht konzipiert werden kann. Rechenintensive Applikationen können auf die Fachkonzept-Schicht ausgelagert werden. Folglich wird nur ein Minimum aus Hardware und Software auf dem Gerät installiert, die als Aufgaben allein die Übertragung der Prozessdaten und die Kommunikation mit der Fachkonzept-Schicht (Server) haben. Dabei gilt es, die notwendige Hardware und Software auf dem Gerät so zu minimieren, dass der Einsatz wirtschaftlich ist.

- *Keine zusätzliche Installation beim Anwender:* Die Kommunikation zwischen Benutzungsoberfläche- und Fachkonzept-Schicht erfolgt über Standard-Protokolle (TCP/IP, HTTP). Diese Protokolle sind schon seit Jahren im Einsatz und repräsentieren den Stand der Technik auf dem Gebiet der Internet-basierten Kommunikation. Die enorme Verbreitung dieser Technologien verringert den Implementierungsaufwand maßgeblich. Außerdem dienen gängige Webbrowser als Benutzerschnittstelle, d. h. der Anwender braucht keine zusätzliche Software auf seinem Rechner, da inzwischen der Browser ein fester Bestandteil jedes Betriebssystems ist.

- *Intuitive Bedienung:* Mit dem Browser sind die meisten Computer-Nutzer vertraut, so dass keine spezielle Einarbeitung für dessen Bedienung notwendig ist.

- *Rechnerunabhängige Bedienung:* Durch den Einsatz der Standard-Technologien (TCP/IP und *Browser-Technologie*) ist die Benutzungsoberfläche systemunabhängig. Der Nutzer benötigt lediglich den Browser zur Bedienung. Es spielt also keine Rolle, auf welchem Betriebssystem der Browser gestartet wurde. Die abgefragten Daten können mit unterschiedlichen Browsern dargestellt werden.

- *Bedarfsorientierte Präsentation der Information:* Web-Technologien wie XML ermöglichen eine einfache Realisierung dieser Anforderung, sodass anwenderspezifische Bedienungsoberflächen automatisch zur Laufzeit generiert werden können. Durch Festlegung bestimmter Anwendergruppen ist es möglich, die Informationen spezifisch für jede Gruppe vorzube-

reiten und darzustellen. Damit wird eine gezielte Informationsweitergabe erleichtert, sodass der jeweilige Anwender nur die Informationen sieht, die er auch benötigt. Dies hilft einerseits einer einfachen Administration der Information und andererseits ermöglicht es einen übersichtlichen Zugriff auf die Daten.

Die größte Herausforderung bei der Implementierung von Fernservice-Applikationen für eingebettete Systeme ist die Realisierung einer kostengünstigen Internet-Anbindung dieser Systeme. Nur die Konzepte, die diese Anforderung erfüllen, können sich auf dem Weltmarkt etablieren. Um dieses Ziel zu erreichen, wurde im Rahmen dieser Arbeit ein Ansatz entwickelt, der ein Minimum aus Hardware und Software für die Realisierung von Fernservice-Applikationen für eingebettete Systeme benötigt. Hierzu wurde eine Serverapplikation konzipiert und anschließend realisiert, die einerseits Anfragen eines beliebigen Clients entgegennimmt, sie verarbeitet und abschließend an das Gerät weiterleitet. Andererseits nimmt sie Informationen aus dem Gerät auf, bearbeitet sie und stellt sie beim Client dar. Mit diesem Ansatz ist es möglich, die auf dem Gerät benötigten Komponenten zu minimieren. Diese gerätspezifischen Komponenten auf dem eingebetteten System haben folgende Aufgaben:

- Aufbau einer Internet-Verbindung zum Server
- Entgegennahme der Anfragen des Servers
- Durchführung der Aktionen auf dem Gerät
- Zurücksendung der Ergebnisse der Anfragen an den Server

Das Konzept *Universelle Fernservice-Infrastruktur* baut auf dem weithin akzeptierten Standard der *Drei-Schichten-Architektur* auf. Basierend auf dieser Architektur und mit Hilfe der standardisierten Web-Technologien, wurde ein system- und plattformunabhängiges Format für die Aufbereitung der Prozessdaten definiert, sodass eine flexible Weiterverarbeitung möglich ist. Hierzu wurde die Beschreibungssprache XML eingesetzt, die eine Vorbereitung der Informationen für unterschiedliche Zielplattformen ermöglicht. Es wurde weiterhin eine einheitliche Schnittstelle konzipiert, die die Voraussetzung für den Zugriff für verschiedene Typen von Clients ist.

Auf diese Art und Weise ist eine Anbindung des eingebetteten Systems an den Server realisierbar. Die Prozessdaten dieses Gerätes können somit für unterschiedliche Clients zur Verfügung gestellt werden und sind dann in der Lage, diese Information nach Bedarf weiterzuverarbeiten (z. B. mittels einer Ferndiagnose-Applikation). Die Schwachstelle dieser Überlegung liegt darin, dass für jedes Gerät eine spezifische Server-Applikation implementiert werden muss. Um dieses Problem zu beheben, wurde eine systemübergreifende Schnittstelle konzipiert. Dadurch sind einerseits die Anbindung unterschiedlicher eingebetteter Systeme an genau einen Server und an-

dererseits ein einheitlicher Mechanismus für den Zugriff auf Prozessdaten realisiert. Abbildung 5.2 veranschaulicht diese Idee:

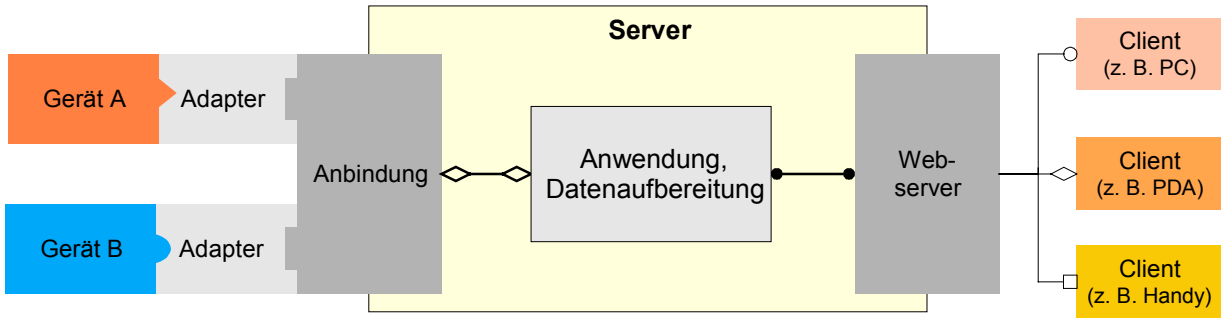

Abbildung 5.2: Prinzip einheitlicher Schnittstellen

Die an den Einsatz von Web-Technologien in eingebetteten Systemen gestellten Anforderungen, wie Kosten und Flexibilität, sind damit erfüllt. Das eingebettete System wird um eine Adapter-Komponente erweitert, sodass eine einheitliche Schnittstelle für den Zugriff auf Prozessdaten realisiert wird. Auf diese Weise wird die jeweils gerätespezifische Schnittstelle des eingebetteten Systems in eine allgemein gültige und definierte Schnittstelle umgewandelt. Auf dem Server wird eine entsprechende, allgemein gültige und systemübergreifende Anbindungskomponente implementiert, die in der Lage ist, unterschiedliche eingebettete Systeme über ein Netzwerk anzubinden und einheitlich auf Prozessdaten zuzugreifen. Die gewonnenen Prozessdaten werden in einem weiteren Schritt von der Datenaufbereitungskomponente in ein Standardformat überführt. Dieses system- und plattformunabhängige Format ermöglicht eine einfache und flexible Transformation in unterschiedliche Ausgabeformate, sodass die Daten auf unterschiedlichen Clients, wie PC, PDA oder mobiles Telefon dargestellt werden können. Die aufbereiteten Daten werden mit Hilfe eines Webservers übermittelt. Die Kosten für die Realisierung der Hardware zur Prozessanbindung beträgt in der prototypischen Version ca. 100 Euro. Es ist weiterhin möglich, die spezifischen Hardware- und Software-Komponenten, die für die Anbindung des eingebetteten Systems an das Internet notwendig sind, zu optimieren, um die Kosten weiter zu senken. Tabelle 5.1 fasst die Bewertung dieses Lösungsansatzes zusammen.

Tabelle 5.1: Bewertung des Lösungsansatzes „Universelle Fernservice-Infrastruktur“

	Universelle Fernservice-Infrastruktur
Herstellerunabhängigkeit	+
Systemunabhängigkeit	+
Spezielle Anwendersoftware	+
Erweiterbarkeit durch Anwender	+
Kosten	+

Nachdem nun das Konzept der universellen Fernservice-Infrastruktur vorgestellt und erläutert wurde, wird nun ein Verfahren zur Realisierung vorgestellt, das auf einer allgemein gültigen Schnittstelle zwischen verteilten Schichten aufbaut.

5.3 Verfahren zur Realisierung einer universellen Fernservice-Infrastruktur für den Einsatz von Web-Technologien in eingebetteten Systemen

Zum besseren Verständnis des Verfahrens werden zunächst die speziellen Rahmenbedingungen diskutiert, die bei der Anbindung eingebetteter Systeme entstehen. Danach werden die Schichten, deren Aufgaben und Bestandteile definiert. Anschließend wird die Systemarchitektur detailliert erläutert.

5.3.1 Web-fähige eingebettete Systeme

Die mit dem Produkt im Zusammenhang stehenden Dienstleistungen in den Bereichen Ferndiagnose, Fernvisualisierung, Fernwartung und Fernbedienung gewinnen immer mehr an Bedeutung. Bei der Sicherstellung dieser Dienstleistungen, vor allem bezüglich Software-Update und Störungsbeseitigung, haben es große Unternehmen in der Regel einfacher, da sie auf allen Weltmärkten durch Filialen oder zumindest durch so genannte „Sales Center“ vor Ort präsent sind. Anders sieht es für kleine- und mittelständische Unternehmen aus. Diese können sich auf Grund ihrer Größe keine weltweite Präsenz leisten.

Der Wettbewerb macht es notwendig, auch kleine und mittelständische Unternehmen bei der Vermarktung ihrer Produkte und der Bereitstellung ihrer Dienstleistungen zu unterstützen. Durch die weltweite und international standardisierte Kommunikationsinfrastruktur des Internets ist der Datenaustausch sowie der Zugriff auf Informationen von nahezu jedem Ort der Welt möglich. Das bietet vor allem kleinen und mittelständischen Unternehmen die Möglichkeit, eine virtuelle Präsenz aufzubauen und auf dem Weltmarkt wettbewerbsfähig zu bleiben. Genau an dieser Stelle bieten Web-Technologien eine hervorragende Möglichkeit für Ferndiagnose, Fernvisualisierung, Fernwartung, Fernbedienung über das Internet. Diese Web-Anwendungen werden unter dem Namen Fernservice-Applikationen zusammengefasst.

5.3.2 Rahmenbedingungen für den Einsatz von Web-Technologien

Um Fernservice-Applikationen mittels Web-Technologien für eingebettete Systeme zu realisieren, müssen die spezifischen Randbedingungen dieser Systeme beachtet werden. Bei der Realisierung solcher Web-basierten Applikationen für Automatisierungsanlagen ist der Freiheitsgrad wesentlich größer, da die Kosten beim Einsatz eines PCs im Vergleich zu den Anlagenkosten

vernachlässigbar sind. Eingebettete Systeme gehören den unteren bis mittleren Preisklassen an, was den Einsatz eines PCs unmöglich macht. Außerdem unterliegen diese Systeme strengen Randbedingungen, die im Folgenden nochmal zusammengefasst werden.

5.3.2.1 Anforderungen an Prozessanbindung bei eingebetteten Systemen

Eine wichtige Anforderung sind die Kosten. Die Erfahrungen in Industrieprojekten zeigen, dass durch zusätzliche Hardware und Software entstehende Kosten die Grenze von (ca.) 1,3% des Gerätepreises nicht überschreiten dürfen. Um dieser Anforderung gerecht zu werden, soll einerseits ein möglichst großer Teil der Software und – wenn möglich – auch der Hardware ausgelagert werden, andererseits müssen Hardware und Software komprimiert und effektiv gestaltet werden.

5.3.2.2 Anforderungen der Anwender

Ein wesentlicher Aspekt dieser Arbeit ist die Vereinheitlichung der Diagnose, Visualisierung, Wartung und Beobachtung eingebetteter Systeme. Diese Aktivitäten können mit Hilfe bewährter und verbreiteter Web-Technologien erfolgen, ohne teures Fachpersonal oder lange Einarbeitungszeiten in Kauf nehmen zu müssen. Voraussetzung dafür ist allerdings eine intuitive Bedienung, ohne zusätzliche komplexe Software installieren zu müssen. Der Anwender soll von jedem beliebigen Rechner und zu jeder beliebigen Zeit in der Lage sein, zur gewünschten Information zu gelangen.

5.3.3 Aufgabenbeschreibung der Schichten

Ausgehend von den soeben beschriebenen Anforderungen, wurde die Drei-Schichten-Architektur zugrunde gelegt und darauf aufbauend das Konzept der universellen Fernservice-Infrastruktur entworfen. Abbildung 5.3 zeigt eine schematische Darstellung dieses Lösungsansatzes, der die genannten Forderungen erfüllt.

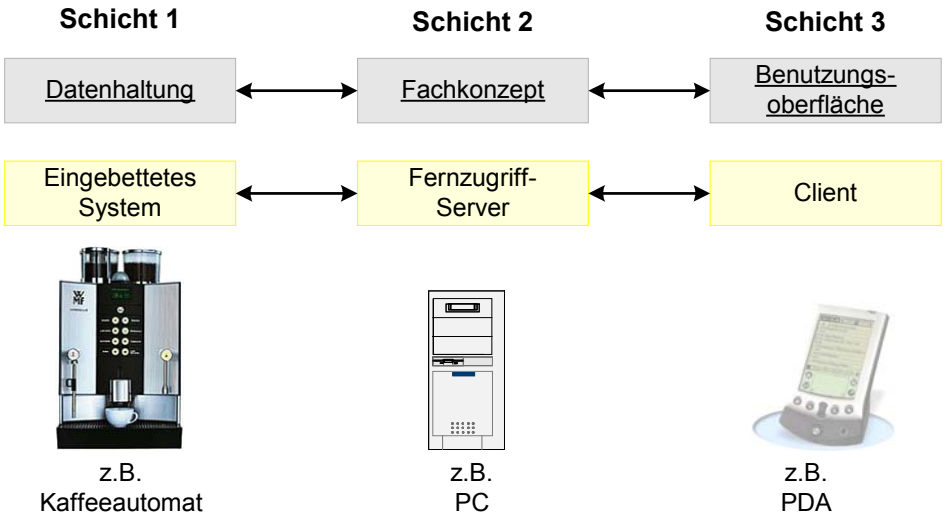

Abbildung 5.3: Schematische Darstellung des Lösungsansatzes

Das eingebettete System entspricht der Datenhaltungsschicht und wird um eine spezifische Komponente zur Bereitstellung der Prozessdaten erweitert. Das Modul wurde weitgehend schlank gehalten, damit es den bereits formulierten Anforderungen entspricht. Auf der anderen Seite steht ein Client, der die Benutzungsoberflächenschicht verkörpert. Ein Client kann ein PC, ein PDA oder ein mobiles Telefon sein. Die Verbindung zwischen diesen zwei Schichten stellt der Fernzugriff-Server (*FzS*) dar. Er hat zwei Aufgaben: Einerseits kommuniziert er mit verschiedenen eingebetteten Systemen und tauscht Daten aus, andererseits bereitet er die gewonnenen Informationen in passende Formate auf, sodass sie auf unterschiedlichen Clients dargestellt werden können. Die gewonnenen Informationen aus den eingebetteten Systemen können auf einfache Art und Weise auch von Fernservice-Applikationen verwendet werden. Diese Applikationen können entweder auf dem Fernzugriff-Server (*FzS*) sein oder nach Bedarf auf einen weiteren Server (sog. Fernservice-Server) ausgelagert werden. Dieser Vorgang unterliegt technischen Randbedingungen wie z. B. Rechenleistung und Zugriffzeiten.

Im Folgenden werden die Aufgaben einzelner Schichten detaillierter beschrieben.

5.3.3.1 Aufgaben der Datenhaltungschicht (Eingebettetes System)

Das eingebettete System wird um eine Adapterkomponente erweitert, um eine einheitliche Schnittstelle für den Zugriff auf Prozessdaten zu schaffen. Damit wird die spezifische Schnittstelle eines eingebetteten Systems in eine definierte Schnittstelle umgewandelt. Folglich hat das eingebettete System zwei Aufgaben:

- *Kommunikation:* Über entsprechende Hardware und Software wird das eingebettete System in die Lage versetzt, über ein beliebiges Netzwerk, z. B. das Internet, mit dem Fernzugriff-Server (*FzS*) zu kommunizieren.

- *Bereitstellung der Prozessdaten:* Durch die Erweiterung des eingebetteten Systems um eine spezifische Softwarekomponente werden dem Fernzugriff-Server (*FzS*) die geforderten Prozessdaten zur Verfügung gestellt.

5.3.3.2 Aufgaben der Fachkonzeptsschicht (Fernzugriff-Server)

Der Fernzugriff-Server (*FzS*) kann um die Funktionalität eines Fernservice-Servers erweitert werden. In diesem Fall werden rechenintensive Fernservice-Applikationen (Ferndiagnose, Fernvisualisierung, Fernwartung, Fernbedienung) auf diesem Server ausgeführt. Die Entscheidung für eine Erweiterung der Funktionalität ist nicht abhängig von der technischen Machbarkeit, sondern von der Logistik. Im Rahmen dieser Arbeit wurde ein Fernzugriff-Server realisiert, der folgende Aufgaben hat:

- *Kommunikation:* Der Fernzugriff-Server ist in der Lage, eine Kommunikation einerseits mit verschiedenen eingebetteten Systemen und andererseits mit unterschiedlichen Clients aufzubauen, um Informationen auszutauschen.

- *Datenverarbeitung:* Der Fernzugriff-Server leitet die Anforderungen eines Clients an ein eingebettetes System. Die zurückgelieferten Prozessdaten werden hier weiterverarbeitet.

- *Datenaufbereitung:* Die verarbeiteten Prozessdaten werden in ein passendes Format umgewandelt und an den Client weitergegeben.

- *Bereitstellung von Geräteinformation:* Der Fernzugriff-Server benötigt unterschiedliche Informationen über eingebettete Systeme, die angebunden werden sollen, wie die Art der Kommunikation, vorhandene Prozessdaten, statische Information etc. Diese Informationen werden als „Geräteinformation“ bezeichnet. Auch hier ist eine Auslagerung der Informationen auf einen externen Server denkbar.

Der Vorteil der Mehr-Schichten-Architektur liegt in der einfachen Skalierbarkeit, der beliebig komplexen Gestaltung der Anwendungslogik und der höheren Stabilität begründet. Es können mehrere eingebettete Systeme angebunden und gleichzeitig verwaltet werden. Im industriellen Einsatz ließe sich der Fernzugriff-Server (*FzS*) auch ohne weiteres als Cluster realisieren, um hohe Verfügbarkeit zu erreichen, wie dies im e-Business schon seit geraumer Zeit der Fall ist. Abbildung 5.4 zeigt eine mögliche Verteilung der bereits vorgestellten Aufgaben auf mehrere Schichten. Damit kann der Fernzugriff-Server (*FzS*) sehr kompakt und kostengünstig gestaltet werden. Erweiterungen sind in Abhängigkeit von den Anforderungen durchführbar.

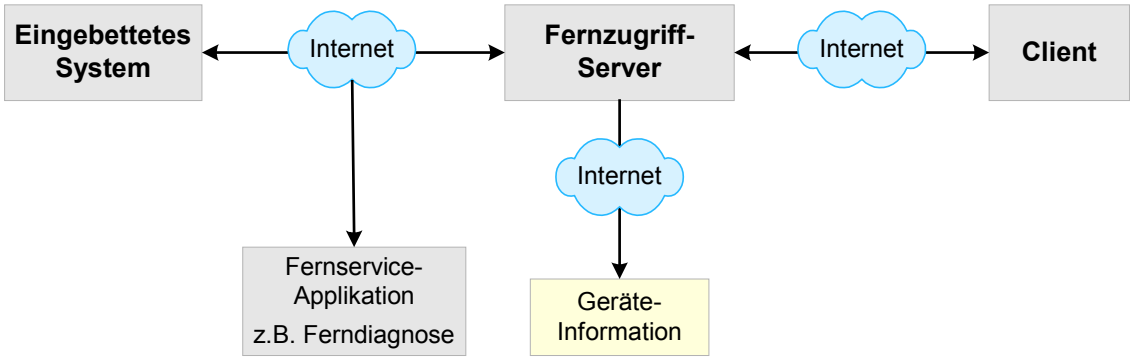

Abbildung 5.4: Alternative Verteilung der Aufgaben auf weitere Schichten

Die Geräteinformation wird zur Konfiguration des Fernzugriff-Servers benötigt. Sie legt fest:

- welche eingebetteten Systeme angebunden werden können,
- wie die Anbindung des eingebetteten Systems an den Fernzugriff-Server erfolgen kann,
- welche Prozessdaten einzelner Geräte abgefragt werden können und
- wie die Prozessdaten für den Client in der Datenaufbereitungskomponente aufbereitet werden müssen.

Für die Realisierung der Geräteinformation auf Grundlage der oben beschriebenen Aufgaben wurde die Form einer eigenständigen Spezifikationssprache gewählt. Da in der Literatur keine adäquate Sprache bekannt ist, wurde zu diesem Zweck die *Service Description Markup Language* (SDML) neu entwickelt. SDML ist eine auf der Metasprache XML (eXtensible Markup Language) [Jeck00] basierende Sprache, die als formalisierte Notation zur Beschreibung der Geräteinformation eingesetzt wird. Durch die enge Beziehung zu XML profitiert SDML von allen Vorteilen, die XML bietet [Dujm01]:

- *Standardisierte Sprache*: XML ist bereits heute ein verbreiteter Industriestandard, der dank seiner einfachen Syntax in vielen Anwendungsbereichen und Produkten eingesetzt wird.
- *Unabhängigkeit von Systemplattformen*: Da XML-Informationen in Textdateien gespeichert werden, können sie verhältnismäßig einfach auf verschiedenen Rechner-, Betriebssystem- und Anwendungsplattformen verarbeitet werden.
- *Flexible Strukturierungsregeln*: XML ermöglicht die Definition von bedarfsgerechten Strukturierungsregeln innerhalb von SDML-Beschreibungsdateien, die auch eine nachträgliche Erweiterung des Sprachumfangs erlauben.

- *Breite Werkzeugunterstützung*: Für XML ist eine große Auswahl unterschiedlicher Werkzeuge verfügbar (z. B. Editoren, Parser, Generatoren), die teilweise sogar kostenlos verfügbar sind [BRA00].

Durch die Erstellung einer Geräteinformation (SDML-Datei) ist es möglich, ein eingebettetes System zu beschreiben und somit die vollständige Anpassung des Fernzugriff-Servers zu erreichen, ohne diesen neu zu implementieren oder gar verändern zu müssen.

5.3.3.3 Aufgaben der Benutzungsoberflächensschicht (Anwender)

Als Benutzungsoberfläche wird die Browser-Technologie eingesetzt. Der Vorteil dieser Technologie ist ihre Standardisierung. Jedes Betriebssystem besitzt einen Browser für die Nutzung des Internets. Selbst kleine PDA besitzen entsprechend kompakte Browser. So braucht der Anwender keine zusätzliche Installation der Software auf seinem Rechner. Er kann weiterhin die gewohnten Browser verwenden. Der Client hat zwei Aufgaben:

- *Verbindungsaufbau:* Der Client baut die Verbindung zum Fernzugriff-Server auf.
- *Darstellung:* Der Client stellt die zurückgelieferten Informationen dar.

6 Spezifikationssprache für die Beschreibung von Geräteinformationen

Im vorangehenden Kapitel wurde die Service Description Markup Language (SDML) als Spezifikationssprache zur Beschreibung der Geräteinformation eingeführt. Da SDML als Beschreibungssprache eine wesentliche Rolle in dieser Arbeit und für den darin vorgeschlagenen allgemeinen Schnittstellenansatz spielt, soll hier die Spezifikationssprache SDML im Detail erläutert werden. Dazu werden nach einer kurzen Einführung die einzelnen Sprachelemente vorgestellt. Anschließend wird ein Beispiel für die Spezifikation eines im Rahmen dieser Arbeit als Modellprozess eingesetzten industriellen Kaffeeautomaten gegeben.

6.1 Grundlagen der Service Description Markup Language

Um auf Prozessdaten eingebetteter Systeme zugreifen zu können, müssen diese Systeme an ein Netzwerk angebunden werden. Ein Fernzugriff-Server hat dann die Möglichkeit, eine Verbindung zu einzelnen Geräten aufzubauen. Der Fernzugriff-Server benötigt für den Verbindungsaufbau zu einem eingebetteten System drei signifikante Informationen über dieses Gerätes:

- *Eindeutige Identifikation*: Der Fernzugriff-Server muss in der Lage sein, jedes eingebettete System zur internen Verwaltung eindeutig identifizieren zu können.
- *Adresse*: Die Adressen aller anzubindenden Geräte in Form *Uniform Resource Locator (URL)* müssen dem Fernzugriff-Server bekannt sein.
- *Verbindungsart:* Dem Fernzugriff-Server muss bekannt sein, welche Technologie (Sockets, RMI oder CORBA) für welches Gerät zu verwenden ist.

Der Fernzugriff-Server muss für den Aufruf der Services konfiguriert werden, damit ihm die bereits beschriebenen Informationen bekannt werden. Für die Konfiguration des Fernzugriff-Servers wird SDML eingesetzt. In einem SDML-Dokument wird festgelegt, wie die Anbindung des eingebetteten Systems an den Fernzugriff-Server erfolgt. Mit Hilfe eines SDML-Dokuments werden außer den Verbindungsmerkmalen noch weitere Informationen zu den eingebetteten Systemen bereitgestellt:

- *Vorhandene Services*: Ein SDML-Dokument beschreibt, welche Services für den Zugriff auf Prozessdaten des jeweiligen Gerätes zur Verfügung stehen.
- *Art der Aufbereitung:* Mittels SDML wird festgehalten, wie Prozessdaten für Clients aufbereitet werden müssen.

6.1.1 Gliederung der SDML

Die SDML ist in der ersten Hierarchieebene in drei Bereiche gegliedert:

- Konfigurationsmanagement
- Beschreibung des eingebetteten Systems
- Detaillierte Beschreibung einzelner Services

Der erste Teil dient dem Konfigurationsmanagement des Dokumentes selbst. Hier werden Informationen zu den Verfassern und zum aktuellen Status des Dokumentes festgehalten. Auch ist eine Versionskontrolle zum Dokument möglich, bei der die durchgeführten Änderungen kurz beschrieben werden.

Im zweiten Teil werden Informationen zum Verbindungsaufbau mit dem eingebetteten System festgelegt.

Der dritte Bereich umfasst die Services, die vom eingebetteten System zur Verfügung gestellt werden. Die Beschreibung eines Service gliedert sich dabei in zwei unterschiedliche Teilbereiche. Zum einen muss definiert werden, wie der Aufruf eines Services zu erfolgen hat, zum anderen benötigt der Fernzugriff-Server Informationen darüber, wie das Ergebnis des Serviceaufrufs zur Darstellung auf dem Client aufbereitet werden soll.

Abbildung 6.1 stellt die Struktur der Spezifikationssprache SDML grafisch dar.

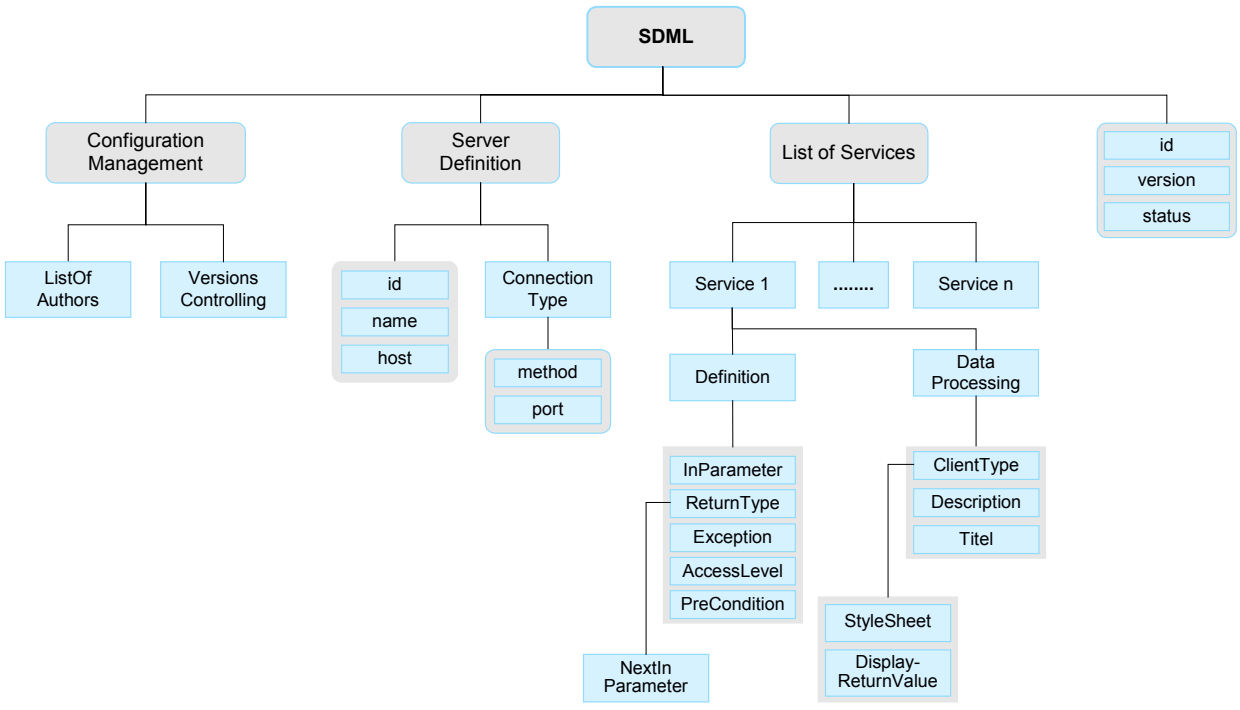

Abbildung 6.1: Gliederung der SDML

Das Attribut `id` des Root-Elements `<sdml>` weist der aktuellen SDML-Instanz eine eindeutige Identifikation zu. Zur Erinnerung sei nochmal gesagt, dass ein Attribut ein namentlich gekennzeichnetes Wertepaar (Name und Wert) ist, das von Tags eingeschlossen ist. Ein Attribut ist ein Charakteristikum eines Datensatzes. Die Attribute `version` und `status` kennzeichnen den aktuellen Stand der SDML-Instanz. Die Version wird durch eine fortlaufende Nummer festgehalten, der Status einer SDML-Instanz durch die Werte *inProgress*, *betaVersion* und *release* beschrieben.

Der Bereich Konfigurationsmanagement, gekennzeichnet durch das Element `<ConfigurationManagement>`, enthält Informationen über die Verfasser des Dokuments sowie über die Versionskontrolle. Abbildung 6.2 zeigt die Beschreibung des Konfigurationsmanagements in SDML.

```
1:    <?xml version="1.0" encoding="ISO-8859-1"?>
2:    <!DOCTYPE sdml SYSTEM "sdml.dtd">
3:    <sdml id="WMF Combination S" version="0.9" status="betaVersion">
4:    <!-- Hier beginnt der Konfigurationsmanagement-Teil -->
5:      <ConfigurationManagement>
6:       <ListOfAuthors>
7:         <Author name="IAS"/>
8:       </ListOfAuthors>
9:       <VersionsControlling lastChange="15.11.2001" authorName="IAS">
10:       <Version number="0.9" status="betaVersion" authorName="IAS"
11:         date="15.11.2001" why="change" what="adjust"/>
12:      <Version number="0.1" status="inProgress" authorName="IAS"
13:         date="23.09.2001" why="create" what="all"/>
14:      </VersionsControlling>
15:     </ConfigurationManagement>
16:   </sdml>
```

Abbildung 6.2: Beispielcode zum Konfigurationsmanagement

Bei der Erstellung von SDML-Dokumenten werden Auszeichnungselemente (englisch: Tag), wie für jedes XML-Dokument, verwendet. Ein Tag ist Teil der Kennzeichnung eines Elementes, der in „<“ (Start) und „>“ oder „/>“ (Ende) eingeschlossen den Namen des Elements, zum Beispiel <element> </element> [BeMi00], repräsentiert. Die ersten zwei Zeilen sind allgemein gültige Beschreibungen, die für alle SDML-Dokumente gelten. In der ersten Zeile wird das Dokument als ein XML-Dokument identifiziert und in der zweiten Zeile wird der Aufbau des Dokuments beschrieben. Dazu werden bestimmte Strukturierungsregeln verwendet, die in dem Dokument *sdml.dtd* festgelegt sind. Alle SDML-Dokumente sind nach den Regeln der DTD (Document Type Definition) aufzubauen. Die explizite Referenzierung ist zwingend [Stla98]. Zeile 3 ist der eigentliche Anfang des SDML-Dokuments, wobei das Root-Element `<sdml>` als Start-Zeichen dient. Das Attribut `id` bekommt den Wert `WMF Combination S` und damit kann das dazugehörige eingebettete System eindeutig identifiziert werden. In Zeile 4 ist ein Kom-

mentar hinzugefügt. In der fünften Zeile beginnt der Konfigurationsmanagement-Teil mit dem Start-Tag `<ConfigurationManagement>` und wird in der Zeile 15 mit dem entsprechenden Ende-Tag `</ConfigurationManagement>` geschlossen. Zwischen den Zeilen 6 und 8 werden alle Autoren, die an dem Dokument mitgearbeitet haben, mittels der Tags `<ListofAuthors>` und `</ListofAuthors>` beschrieben. Der Tag `<Author>` beschreibt einen Autor. Die Versionen werden durch den Tag `<VersionsControlling>` in der neunten Zeile verwaltet. Dieser Tag bekommt zwei Attribute, `lastChange` (wann die letzte Änderung durchgeführt wurde) und `authorName` (wer diese Änderung durchgeführt hat). Die Zeilen 10 bis 13 beschreiben die verschiedenen Versionen. Der Tag `<Version>` hat folgende Attribute:

- `number`: für die Versionsnummer.
- `status`: zeigt, ob das Dokument im Bearbeitungs-, Beta- oder Release-Zustand ist.
- `authorName`: zeigt, wer diese Version bearbeitet hat.
- `date`: Datum der Änderung.
- `why`: Begründung, warum eine neue Version erstellt wurde.
- `what`: Angaben über die Art der Änderung.

Schließlich werden die Versionskontrolle in Zeile 14, das Konfigurationsmanagement in Zeile 15 und das SDML-Dokument insgesamt in Zeile 16 beendet.

Im Abschnitt `<ServerDefinition>` in Abbildung 6.1 wird das eingebettete System beschrieben, das angebunden werden soll. Abbildung 6.3 zeigt ein Beispiel für solche Definitionen.

```
1:    <ServerDefinition id="wmf" name="WMF Combination IAS"
2:       host="129.69.225.82">
3:     <ConnectionType method="socket" port="6325">
4:     </ConnectionType>
5:    </ServerDefinition>
```

Abbildung 6.3: Beispielcode zur Server-Definition

Der Tag `<serverDefinition>` in der ersten Zeile startet diesen Abschnitt. Über das Attribut `id` wird dem eingebetteten System eine eindeutige Identifikation, in diesem Falle „wmf“, zugewiesen. Der Name des Gerätes, der bei der Auswahl beim Fernzugriff-Server angezeigt werden soll, wird durch das Attribut `name` gekennzeichnet. Das Attribut `host` in der zweiten Zeile beschreibt mittels des Uniform Resource Locator (URL) den Standort des eingebetteten Systems im Netz. Zuletzt wird über das Kindelement `<ConnectionType>` in Zeile 3 die Art der Verbindung festgelegt. Es ist vorgesehen, dass Verbindungen sowohl für Sockets, RMI als auch CORBA deklariert werden können. Bei einer Socketverbindung muss zusätzlich über das Attri-

but `port` die Nummer des Ports für die Verbindung angegeben werden. Die Zeilen 4 und 5 beinhalten schließlich die zugehörigen End-Tags.

Im dritten Abschnitt erfolgt die detaillierte Beschreibung der Services, die vom eingebetteten System zur Verfügung gestellt werden sollen, wobei jedes Kindelement `<Service>` des Elements `<ListOfServices>` für einen einzelnen Service steht.

6.1.2 Beschreibung eines Services

Die Beschreibung eines Service gliedert sich in zwei verschiedene Teilbereiche. Zum einen muss definiert werden, wie der Service aufgerufen wird. Diese Definition wird unter dem Element `<Definition>` festgehalten. Zum anderen muss durch das Element `<DataProcessing>` beschrieben werden, wie die Datenaufbereitungskomponente resultierende Daten für den jeweils anfragenden Client aufbereiten soll. Im Folgenden werden diese zwei Teilaufgaben beschrieben.

6.1.2.1 Definition eines Serviceaufrufs

Oft werden für einen Serviceaufruf Parameter benötigt, um die Anfrage genauer zu spezifizieren und das Ergebnis einzuschränken. Diese Parameter können dabei meist aus dem Ergebnis des vorherigen Serviceaufrufs gewonnen werden. Das führt zu einer hierarchischen Schachtelung der Services untereinander. Abbildung 6.4 veranschaulicht diesen Sachverhalt.

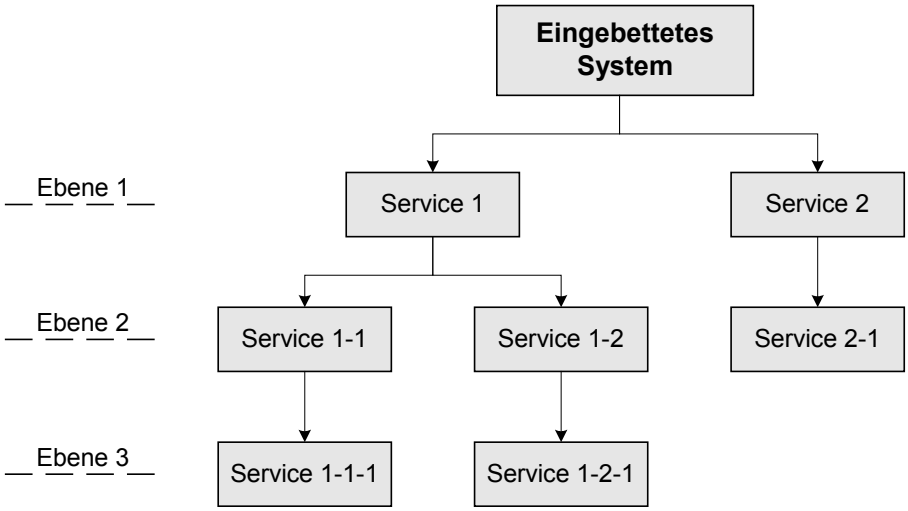

Abbildung 6.4: Hierarchische Schachtelung von Services

Um die Abhängigkeit der Services untereinander nachvollziehbar festzuhalten, wird bei der Definition eingetragen, welche Services als Vorbedingung notwendig sind.

Das Attribut `id` legt die Identifikation einer Servicedefinition fest. So kann ein Service auf verschiedene Weise deklariert werden. Der Aufruf des Services wird durch das Attribut `call` festgelegt. Der Text, den der Benutzer bei der Auswahl erhält, ist durch das Attribut `buttontext` festgelegt. Für die genauere Spezifikation des Service stehen folgende Tags zur Verfügung:

<ReturnType>

Um den Typ des Rückgabewerts eines Serviceaufrufs festzulegen, wird der Tag `<ReturnType>` verwendet. Zur Auswahl stehen einfacher Text oder ein boolescher Wert. Das Attribut `dimension` gibt die Mächtigkeit (0=einzelner Wert; 1=Liste; 2=Matrix) des Rückgabewertes an.

<PreCondition>

Dieser Tag legt den Service fest, der zuvor aufgerufen werden muss. Der Service wird dabei durch das Attribut `id` identifiziert.

<AccessLevel>

Mittels dieses Tags wird die Zugangsberechtigung für einen Service über das Attribut `level` festgelegt. So können verschiedene Benutzergruppen mit unterschiedlichen Berechtigungen für den Fernzugriff-Server festgelegt werden. Damit bekommt jede Benutzergruppe nur diejenigen Informationen zur Ansicht, die für sie gedacht sind.

<Exception>

Sollte beim Aufruf eines Services ein Fehler auftreten, so kann durch diesen Tag eine entsprechende Nachricht an den Benutzer definiert werden.

<InParameter>

Dieser Tag definiert jeweils einen Parameter, der für den Serviceaufruf benötigt wird. Über die Attribute `id` kann der Name des Parameters und über `type` sein Typ festgelegt werden. Beim Typ wird zwischen einem einfachen Text und einem booleschen Wert unterschieden. Ist durch das Auszeichnungselement `<PreCondiction>` ein Serviceaufruf als Vorbedingung festgelegt, so wird dessen Rückgabewert als Parameter übernommen. Ist nur ein Teil des Rückgabewertes als Eingabeparameter interessant, so wird durch die Attribute `row` und `col` der Bereich eingegrenzt. Wird hierbei statt einer festen Zahl der Begriff `user` zugeordnet (z. B. *row=user*), erfolgt die Auswahl des Parameters durch den Benutzer.

Das folgende Beispiel verdeutlicht die eben vorgestellten Zusammenhänge.

Das eingebettete System *Kaffeeautomat* bietet den Service „Liste_der_Kaffeesorten“ und den Service „erstelle_Kaffee“ an. Der erste Service „Liste_der_Kaffeesorten“ übergibt als Ergebnis

eine Liste der verschiedenen Kaffeesorten, die dieser Kaffeeautomat produzieren kann (siehe Tabelle 6.1).

Tabelle 6.1: Liste der Kaffeesorten des Kaffeeautomaten

Kaffeesorte des Kaffeeautomaten	**ID**
Cappuccino	1
Kaffee	2
Latte Macchiato	3
Milchkaffee	4

In der ersten Spalte befindet sich der Name des Produkts und in der zweiten Spalte die dazugehörige eindeutige ID.

Für den ersten Service muss nur dessen Aufruf (Abbildung 6.5) deklariert werden.

```
1:    <Definition id="Kaffeesorten" call="Liste_der_Kaffeesorten"
2:       buttontext="Liste der Kaffeesorten">

3:     <ReturnType type="text" dimension="2">
4:     </ReturnType>

5:     <AccessLevel level="1"/>
6:     </AccessLevel>
7:    </Definition>
```

Abbildung 6.5: Definition des Services „Liste_der_Kaffeesorten“

In der ersten Zeile ist der Start-Tag `<Definition>` mit den Attributen `id` zur Identifikation, `call` zum Aufruf des Services und `buttontext` zu sehen. In der dritten Zeile wird der Typ des Rückgabewerts und dessen Mächtigkeit festgelegt. Die Zugangsberechtigung wird mittels des Attributs `level` in Zeile 5 beschrieben. Schließlich wird die Service-Definition in der siebten Zeile beendet.

Der Eingabeparameter Kaffeesorte des zweiten Services „erstelle_Kaffee“ soll durch den Benutzer auf Grund der Auswahl vom Ergebnis des ersten Service bestimmt werden. Abbildung 6.6 zeigt die dazugehörige Service-Definition.

```
1:    <Definition id="Kaffee" call="erstelle_Kaffee"
2:       buttontext="erstelle Kaffee">

3:     <InParameter id="product" type="text" row="user" col="2"/>
4:     <ReturnType type="text" dimension="0"/>
```

```
5:      <PreCondition id="getProducts"/>
6:      <AccessLevel level="1"/>

7:    </Definition>
```

Abbildung 6.6: Definition des Services „Erstelle_Kaffee“

Beim zweiten Service wird festgelegt, dass durch das Ergebnis des ersten Service der Eingabeparameter ermittelt wird, nämlich die eindeutige ID der Kaffeesorte. Dies geschieht in Zeile 3.

Zur allgemeinen Darstellung der Struktur der Definition eines Services in der SDML dient Abbildung 6.7.

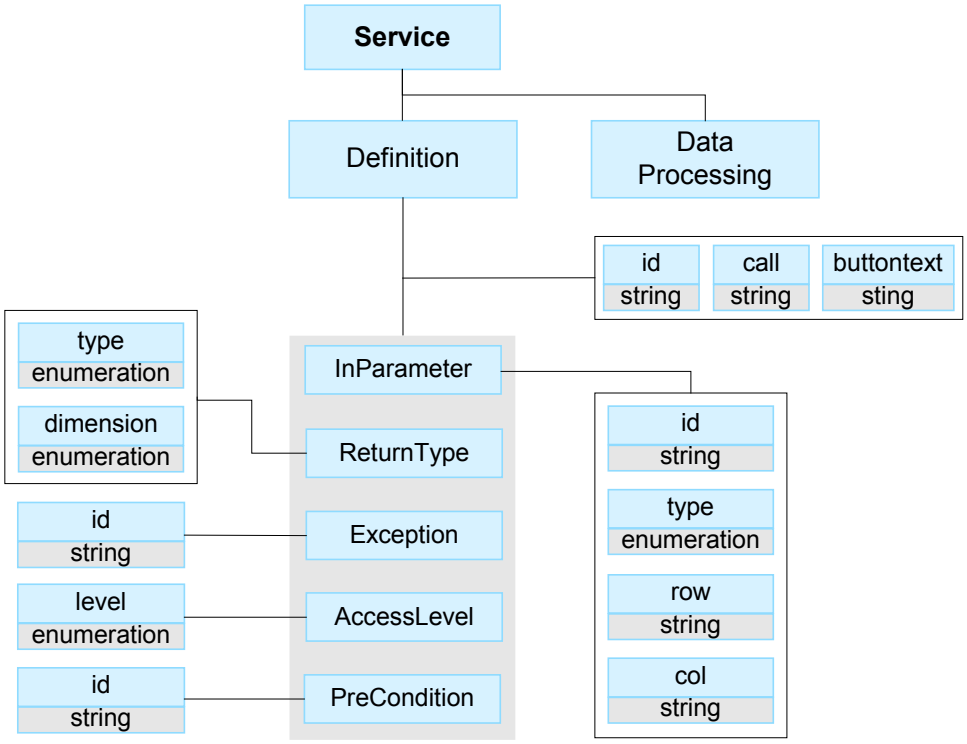

Abbildung 6.7: Struktur der Definition eines Services

6.1.2.2 Definition der Datenaufbereitung

Das Auszeichnungselement `<DataProcessing>` wird eingesetzt, um die Beschreibung zur Aufbereitung der Daten zu definieren, die ein Service als Ergebnis zurückliefert. Hierfür kann für jeden Typ Client durch das Auszeichnungselement `<ClientType>` die spezifische Aufbereitung festgelegt werden. Liegt für einen bestimmten Clienttyp keine Beschreibung vor, so wird der Service bei der Auswahl gar nicht erst angeboten.

Wie schon in der Definition eines Serviceaufrufs erwähnt, unterscheidet sich das Ergebnis, das ein Service zurückliefert, zum einen durch seinen Typ und zum anderen durch seine Dimension.

Für die Darstellung des Ergebnisses reicht es aus, zwischen Text und einem booleschen Wert als Typ zu unterscheiden. Ein Text kann sofort dargestellt werden, bei einem booleschen Wert macht es allerdings keinen Sinn, einfach nur `True` oder `False` auf dem Display anzuzeigen. Hier bedarf es einer Zuordnung, die bei Erfolg oder Misserfolg des Serviceaufrufs den einen oder anderen Text ausgibt. Über die Dimension kann angegeben werden, ob es sich beim Rückgabewert nur um einen einzelnen Wert (Dimension=0), eine Liste von Werten (Dimension=1) oder eine Tabelle (Matrix) von Werten (Dimension=2) handelt. Beim Rückgabewert eines Login-Service z. B. dürfte es sich in den meisten Fällen um einen einzelnen Wert, also mit Dimension 0, vom Typ `Boolean` handeln. Hingegen liefert ein „gibAlleAktuellenProzessdaten"-Service die Prozessdaten als Liste von Name/Werte-Paaren, also in Form einer Tabelle (Dimension=2), zurück. Hierbei ist eine gesonderte Datenaufbereitung für verschiedene Typen von Clients sinnvoll. Das kann eine größe Übersichtlichkeit und bessere Navigationsführung bewirken.

Mit dem Auszeichnungselement `<Title>` wird der Titel der Ergebnis-Seite festgelegt. Um eine kurze Erklärung über das Ergebnis des Serviceaufrufs abzugeben, wird das Auszeichnungselement `<Description>` verwendet.

Das Attribut `type` des Auszeichnungselements `<ClientType>` bestimmt den Typ des Clients, für den die Aufbereitung gelten soll. Im Rahmen dieser Arbeit ist hier die Auswahl zwischen PC, PDA und mobilem Telefon vorgesehen. Durch das Attribut `form` kann eine Auswahl zwischen vordefinierten Aufbereitungen gewählt werden, die in Tabelle 6.2 aufgelistet sind.

Tabelle 6.2: Arten der Aufbereitung

Form	**Beschreibung**
text	Die Daten werden in einen zusammenhängenden Text umgewandelt.
table	Die Daten werden in eine Tabelle umgewandelt.
radiobutton	Die Daten werden als Radiobuttons übergeben. Diese Möglichkeit bietet eine exklusive Auswahl der Ergebnisse für den nächsten Service.
checkbox	Die Daten werden als Checkboxes übergeben. Diese Möglichkeit bietet eine kumulierte Auswahl der Ergebnisse für den nächsten Service.

Dem Auszeichnungselement können zwei weitere Kindelemente zugeordnet werden. Das Auszeichnungselement `<Stylesheet>` legt über das Attribut `link` das Stylesheet fest, das für die Transformierung in das richtige Ausgabeformat verwendet werden soll. Mit Hilfe des Elements `<DisplayReturnValue>` und des Attributs `col` kann bei tabellarischen Rückgabewerten festgelegt werden, ob nur eine oder mehrere Spalten angezeigt werden sollen. Dies ist z. B. bei einer zweispaltigen Tabelle als Rückgabewert sinnvoll, bei der sich in der ersten Spalte die darzustellenden Werte und in der zweiten die dazugehörigen Identifikationsnummern befinden, die

zum Aufruf des nachfolgenden Service benötigt werden. Zur Verdeutlichung soll auch hier das Beispiel aus der Definition eines Serviceaufrufs (siehe Abbildung 6.5) dienen.

Damit der Benutzer eine Kaffeesorte aus der Liste auswählen kann, soll diese in Form von Radiobuttons angezeigt werden. Die Auswahl soll sowohl von einem Standard-PC als auch von einem PDA aus möglich sein, jedoch nicht von einem mobilen Telefon. Zudem wird für den jeweiligen Client ein eigenes Stylesheet zur Formatierung festgelegt. Da der Benutzer mit den eindeutigen Id's nichts anfangen kann, soll nur die Spalte mit den Namen der Kaffeesorten angezeigt werden. Dies führt zur folgenden Definition der Datenaufbereitung für den Service „Liste_der_Kaffeesorten":

```
1:    <DataProcessing>
2:     <Title>Kaffeesorten des Kaffeeautomaten</Title>

3:     <Description>Diese Kaffeesorten können von dem
4:      Kaffeeautomaten produziert werden.
5:     </Description>

6:     <ClientType type="pc" form="radiobutton">
7:      <DisplayReturnValue col="1"/>
8:      <Stylesheet link="html_pc.xsl"/>
9:     </ClientType>

10:    <ClientType type="pda" form="radiobutton">
12:     <DisplayReturnValue col="1"/>
13:     <Stylesheet link="html_pda.xsl"/>
14:    </ClientType>

15:   </DataProcessing>
```

Abbildung 6.8: Datenaufbereitung des Services „Liste_der_Kaffeesorten"

In den Zeilen 5 bis 8 wird ein spezifisches Stylesheet für die PC-Clients (html_pc.xsl) und in den Zeilen 9 bis 12 dasselbe für die PDA-Clients (html_pda.xsl) festgelegt.

Der zweite Service „erstelle_Kaffee" gibt die Meldung zurück, dass der Kaffee beim Kaffeeautomaten abgeholt werden kann. Diese Meldung kann auf dem Client einfach angezeigt werden (z. B. in Form „Der Kaffee ist fertig!"). In Abbildung 6.9 ist der Vorgang der Datenaufbereitung des Service „erstelle_Kaffee" dargestellt.

```
1:    <DataProcessing>
2:     <Title>Kaffee erstellt</Title>

3:     <Description>Der Kaffee ist fertig!
4:     </Description>

5:     <ClientType type="pc" form="text">
```

```
6:       <Stylesheet link="html_pc.xsl"/>
7:      </ClientType>

8:      <ClientType type="pda" form="text">
9:       <Stylesheet link="html_pda.xsl"/>
10:     </ClientType>

11:   </DataProcessing>
```

Abbildung 6.9: Datenaufbereitung des Services „Erstelle_Kaffee“

Das Stylesheet für die Meldung an PC-Clients wird in den Zeilen 5 bis 7 und für die PDA-Clients in den Zeilen 8 bis 10 festgelegt. Die Struktur der Datenaufbereitung eines Services in SDML ist in Abbildung 6.10 grafisch dargestellt.

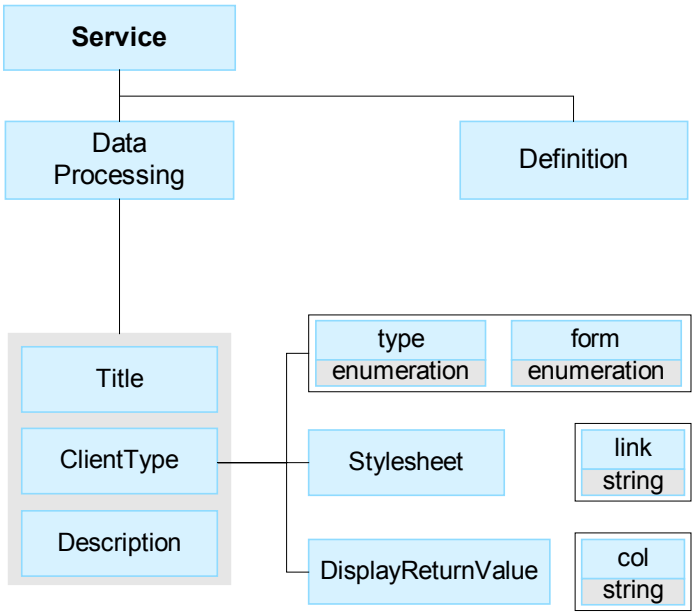

Abbildung 6.10: Struktur der Datenaufbereitung eines Services in SDML

Die vorgestellte Spezifikationssprache SDML stellt einen Ansatz einer allgemein gültigen Beschreibungssprache zum Aufruf von Services dar. Speziell bei der Abbildung komplexer Strukturen hierarchisch verschachtelter Services stößt die derzeitige Version der SDML an ihre Grenzen. Mit der Einführung von Logikelementen zur Definition der Serviceaufrufe könnte die Flexibilität erhöht werden.

7 Funktionaler Aufbau der universellen Fernservice-Infrastruktur

Nachdem die abstrakten Aufgaben einzelner Schichten erläutert wurden, wird nun in diesem Kapitel der funktionale Aufbau aller Knoten beschrieben. Für die Anbindung wird das eingebettete System um eine *spezielle Anbindungskomponente (sAk)* erweitert. Diese spezielle Anbindungskomponente hat die Funktion einer Adapterkomponente und bietet eine einheitliche Schnittstelle für den Zugriff auf Prozessdaten. Die spezifische Schnittstelle des eingebetteten Systems wird in eine einheitlich definierte Schnittstelle umgewandelt. Die Funktionalität der speziellen Anbindungskomponente (*sAk*) wird im Abschnitt 7.1.2 beschrieben. Der Fernzugriff-Server (*FzS*) kann, ausgerüstet mit der *allgemeinen Anbindungskomponente (aAk)*, eine Verbindung zur speziellen Anbindungskomponente des eingebetteten Systems herstellen und auf Prozessdaten zugreifen. Der Aufbau des Fernzugriff-Servers (*FzS*) wird in Abschnitt 7.2 behandelt. Abschließend beschreibt Abschnitt 7.3, wie ein Client eine Anfrage an den Fernzugriff-Server (*FzS*) senden kann.

7.1 Das Gerät als Datenquelle

Ein eingebettetes System verfügt über eine Steuerung, die mittels Sensoren aktuelle Informationen über den Prozess erhält und mit Hilfe von Aktoren den Ablauf des eingebetteten Systems regelt. Der Zugriff auf Prozessdaten erfolgt dabei mit Hilfe der Methoden der Steuerung. Die Methoden, die für den Fernzugriff auf die Prozessdaten vom eingebetteten System zur Verfügung gestellt werden, werden im Folgenden als „Service“ bezeichnet.

Für den Zugriff des Fernzugriff-Servers (*FzS*) auf Prozessdaten wird das eingebettete System um eine spezielle Anbindungskomponente (*sAk*) erweitert. Über diese Komponente kann der Fernzugriff-Server mittels der allgemeinen Anbindungskomponenten (*aAk*) eine Netzwerkverbindung zum eingebetteten System herstellen. Der Zugriff auf die Prozessdaten erfolgt über Aufruf der Services, die vom eingebetteten System zur Verfügung gestellt werden. Für den Aufruf dieser Services wurde eine abstrakte Methode „callService“ eingeführt, die wie nachfolgend beschrieben von der sAk implementiert wird.

7.1.1 Abstrakte Methode „callService“

Mit der abstrakten Methode „callService“ wird eine übergeordnete Methode definiert, die den eigentlichen Serviceaufruf wie einen Umschlag umhüllt. Die spezielle Anbindungskomponente (*sAk*) bietet dem Fernzugriff-Server somit nur diese eine abstrakte Methode für den Fernzugriff an. Die Funktionsweise ist dabei wie folgt:

- Ein Client stellt eine Anfrage nach einem Service (z. B. "*Zustand_Ventil_Heisswasser"*) an den Fernzugriff-Server.
- Auf Grund der Beschreibung in der „Geräteinformation“ wird der „callService-Aufruf“ vom Fernzugriff-Server für diesen Aufruf parametrisiert.
- Der parametrisierte „callService“-Aufruf wird mittels der allgemeinen Anbindungskomponente (*aAk*) via Netzwerk an die spezielle Anbindungskomponente (*sAk*) des eingebetteten Systems übermittelt.
- Die spezielle Anbindungskomponente (*sAk*) ermittelt auf Grund der Parametrisierung den eigentlichen Serviceaufruf und führt diesen durch.

Das Ergebnis des Serviceaufrufs wird der abstrakten Methode „callService“ als Rückgabewert mitgegeben. Abbildung 7.1 stellt diesen Vorgang grafisch dar.

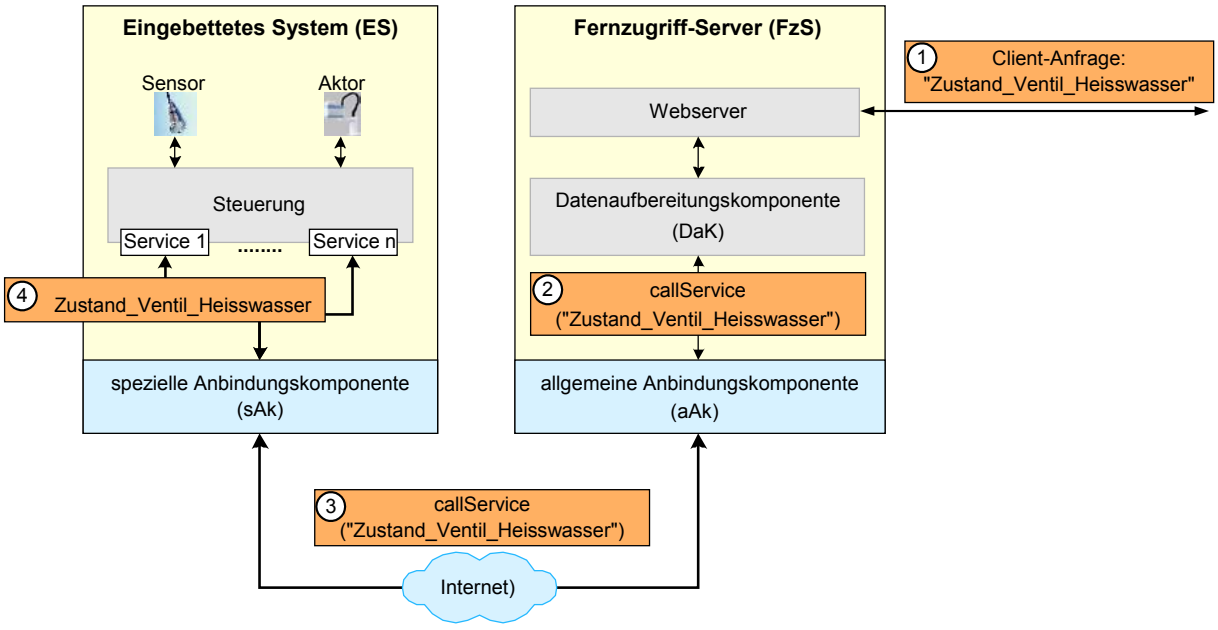

Abbildung 7.1: Abstrakte Methode „callService“

7.1.2 Spezielle Anbindungskomponente

Die spezielle Anbindungskomponente (*sAk*) besteht aus einer Fernzugriffskomponente und einer Wrapper-Komponente. Die Fernzugriffskomponente stellt die abstrakte Methode „callService“ für den Fernzugriff zur Verfügung. Der Fernzugriff soll unter Sockets, Remote Method Invocation (RMI) und der Common Object Request Broker Architecture (CORBA) möglich sein. Damit wird eine Plattformunabhängigkeit auch bei der Kommunikation erreicht. Die Wrapper-Komponente zerlegt Parameter der abstrakten Methode „callService“ und wertet diese aus. Auf

Grund dieser Parameter erfolgt der eigentliche Serviceaufruf durch die Wrapper-Komponente beim eingebetteten System. Abbildung 7.2 dient zur Veranschaulichung.

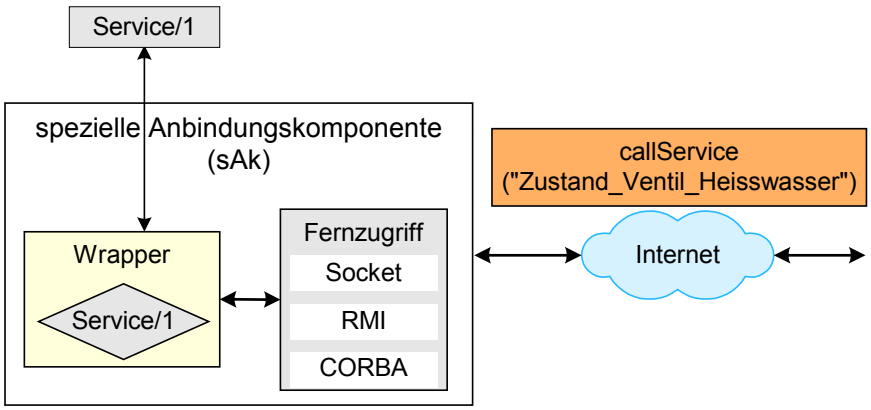

Abbildung 7.2: Aufbau der speziellen Anbindungskomponente

Der Ansatz des abstrakten Serviceaufrufs in Verbindung mit der speziellen Anbindungskomponente, die auf Grund der Parametrisierung den eigentlichen Serviceaufruf durchführt, ermöglicht die Anbindung beliebiger eingebetteter Systeme an den Fernzugriff-Server. Der Fernzugriff-Server (*FzS*) ruft dabei immer nur eine Methode auf, die abstrakte Methode „callService“. Hierdurch ist gewährleistet, dass der Fernzugriff-Server (*FzS*) nicht neu implementiert oder verändert werden muss. Die Parametrisierung des abstrakten „callService“ reicht aus, um einen bestimmten Service im eingebetteten System aufzurufen und somit den Zugriff auf Prozessdaten zu erlangen. Die Parametrisierung erfolgt mittels der *Geräteinformation*.

7.1.3 Anbindung beliebiger Anwendungen an den Fernzugriff-Server

Durch den beschriebenen Lösungsansatz können neben eingebetteten Systemen auch beliebige Applikationen an den Fernzugriff-Server (*FzS*) angebunden werden. Dies bietet z. B. die Möglichkeit, Daten einer Diagnose-Applikation über den Fernzugriff-Server einem Client zur Verfügung zu stellen. Dabei ist das Prinzip der Anbindung analog zur Anbindung des eingebetteten Systems aufgebaut. Die Anwendung, die ihre Daten mittels Service bereitstellen möchte, wird ebenfalls um die spezielle Anbindungskomponente (*sAk*) erweitert, womit der Zugriff auf die Daten der Applikation mit Hilfe von Serviceaufrufen ermöglicht wird. Als Services werden in diesem Fall die Methoden der Applikation bezeichnet. Der Service, der bereitgestellt werden soll, wird analog zur Geräteinformation über das eingebettete System in einer *Applikationsinformation* beschrieben und kann dann vom Fernzugriff-Server aufgerufen werden. Abbildung 7.3 zeigt eine mögliche Erweiterung des Fernzugriff-Servers um zwei weitere Fernservice-Applikationen.

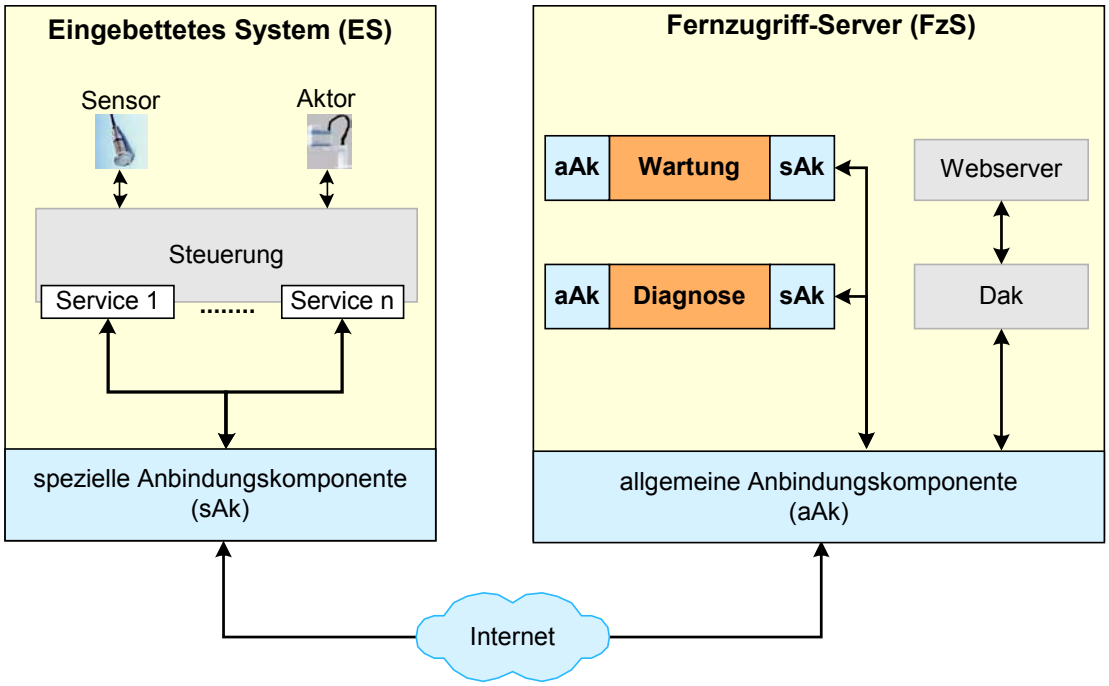

Abbildung 7.3: Erweiterung des Fernzugriff-Servers um weitere Fernservice-Applikationen

Beim vorgestellten Ansatz spielt es keine Rolle, ob die anzubindende Anwendung oder das eingebettete System über ein Netzwerk oder direkt angesprochen wird. Dadurch ist es z. B. möglich, die Diagnose-Anwendung auf einem separaten Rechner unterzubringen. Des Weiteren bietet es sich an, auch den Zugriff der Diagnose- bzw. Wartungs-Applikation auf das eingebettete System mit Hilfe des vorgestellten Konzeptes umzusetzen, um so eine einheitliche Infrastruktur zu schaffen.

Abbildung 7.4 zeigt eine mögliche Verknüpfung zwischen eingebettetem System, Diagnose-Applikation und Fernservice-Server. Die Diagnose-Applikation übernimmt dabei zwei Rollen: Sie greift auf Daten des eingebetteten Systems zu und liefert Daten an den Fernservice-Server.

Der Ansatz bietet somit die Möglichkeit, Systemkomponenten nach dem Baukastenprinzip beliebig miteinander zu verknüpfen. Die Herkunft der Daten wird transparent.

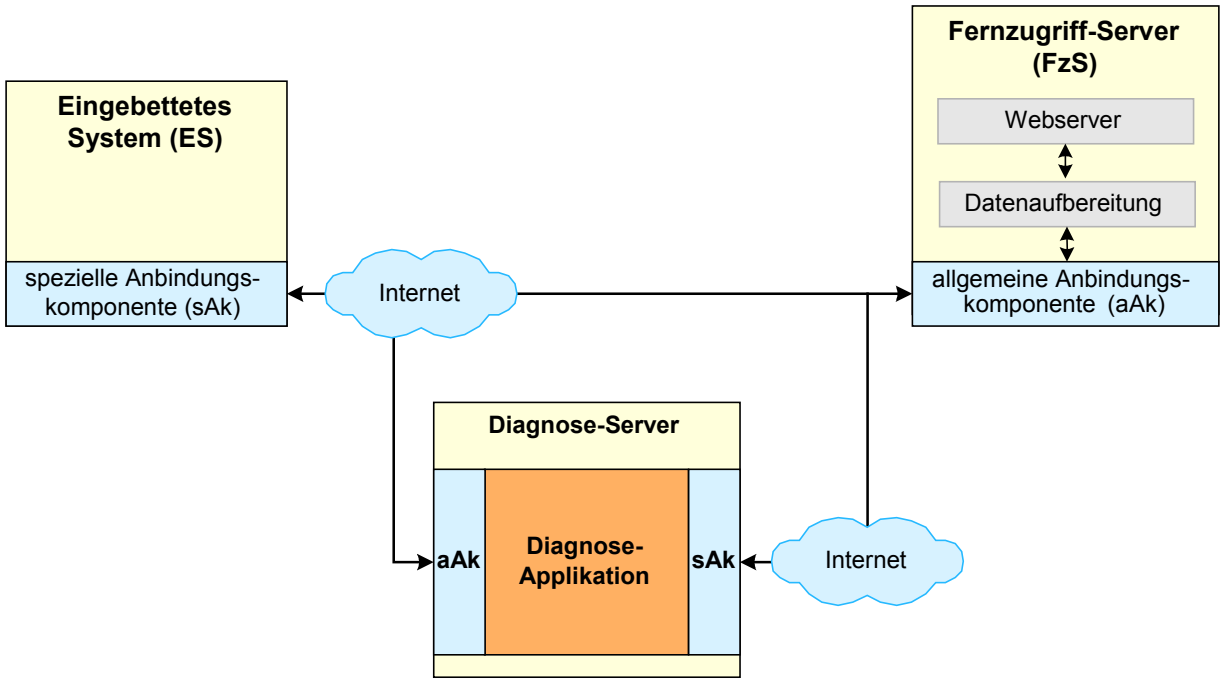

Abbildung 7.4: Anbindung beliebiger Applikationen nach dem Baukastenprinzip

7.2 Fernzugriff-Server

Der Fernzugriff-Server spielt eine zentrale Rolle in der Realisierung der universellen Fernservice-Infrastruktur. Er besteht aus drei Komponenten (Webserver, Dak und aAk), einem Editor, den Geräteinformationen und zwei allgemeingültigen Schnittstellen. Die Schnittstelle *aAK* ermöglicht die einheitliche Anbindung unterschiedlicher eingebetteter Systeme, der Webserver bietet eine universelle Kommunikation zu verschiedenen Clients wie PDA, mobiles Telefon oder PC. Dieses Prinzip schafft die Voraussetzung für eine einmalige Implementierung des Fernzugriff-Servers, der dann in der Lage ist, mehrere Geräte bzw. Clients anzubinden. Die einmalig entstehenden Kosten für die Realisierung des Fernzugriff-Servers minimieren sich auf Grund der Anbindung einer großen Anzahl von Geräten und Anwendern. Abbildung 7.5 zeigt den Aufbau des Fernzugriff-Servers in der realisierten Form.

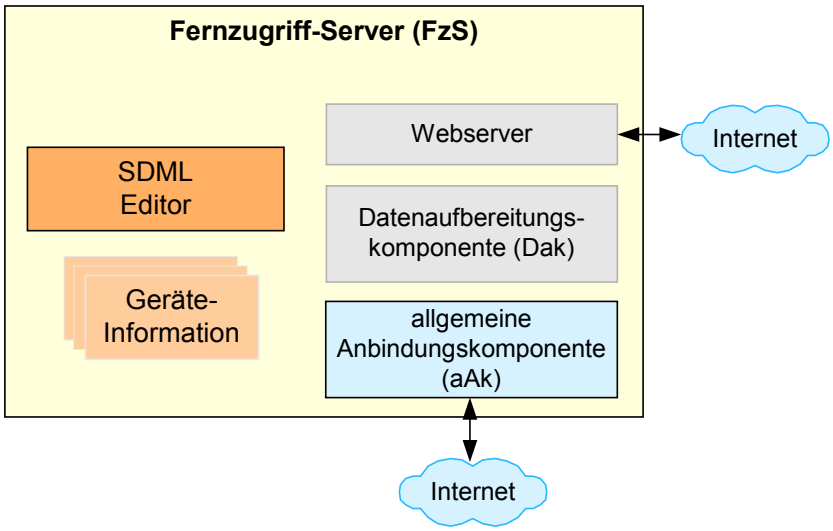

Abbildung 7.5: Aufbau des Fernzugriff-Servers

Die allgemeine Anbindungskomponente (*aAk*) dient als Schnittstelle zu den Geräten und ist für die Anbindung unterschiedlicher eingebetteter Systeme zuständig. Die Webserver-Komponente übernimmt die Kommunikation mit Clientstationen. Die Datenaufbereitungskomponente wiederum nimmt Service-Aufrufe der Clients und geforderte Prozessdaten entgegen und bereitet die Informationen für die Clients auf. Schließlich übernimmt die *Geräteinformation* die Konfiguration des Fernzugriff-Servers. Für die Erstellung der Geräteinformation wurde im Rahmen dieser Arbeit ein Editor realisiert, der auf dem Fernzugriff-Server integriert ist. Die einzelnen Komponenten können dabei beliebig verteilt werden. Die Geräteinformation wird für jedes Gerät spezifisch erstellt. Hierzu ist das Fachwissen des Geräteherstellers notwendig. Aus diesem Grunde ist es sinnvoll, dass der SDML-Editor und die Geräteinformation auf einem Server beim Hersteller lokalisiert werden. Der Fernzugriff-Server kann bei einem Dienstleistungsanbieter, der z. B. die Diagnose, Wartung und das Software-Update der Geräte durchführt, vorhanden sein. Abbildung 7.6 veranschaulicht dieses Vorgehen.

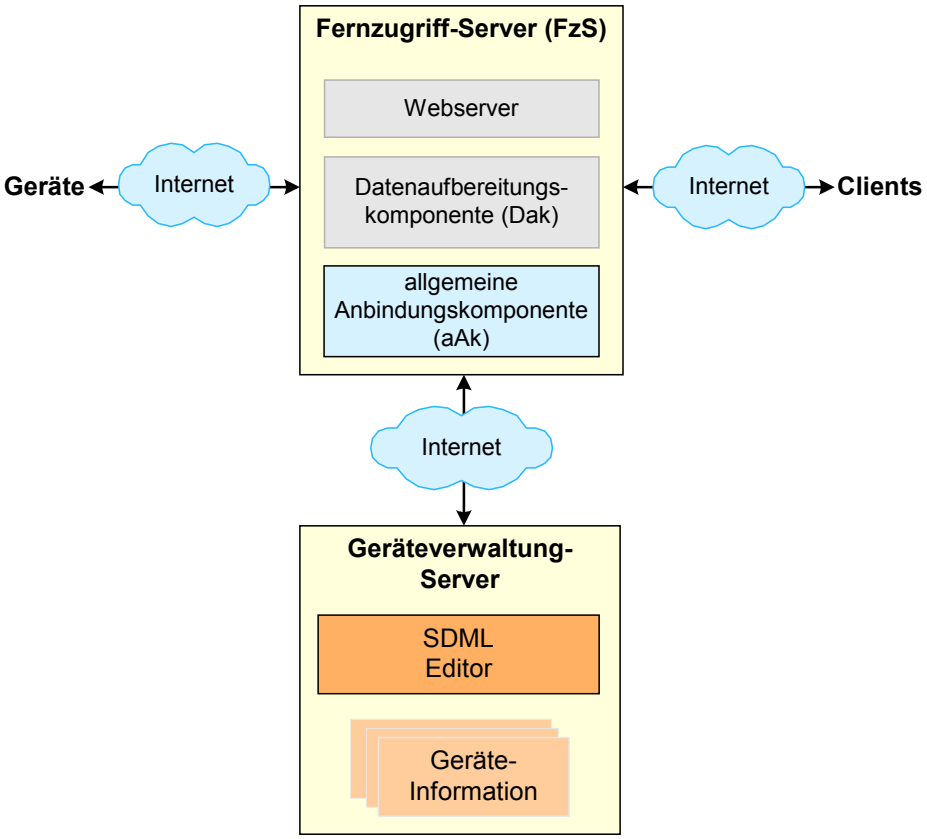

Abbildung 7.6: Eine mögliche Verteilung des Fernzugriff-Servers

Hier kümmert sich der Hersteller lediglich um die Erstellung der Geräteinformation, die in seinem Kompetenzbereich liegt. Sie kann von ihm leicht aktualisiert und gepflegt werden. Der Dienstleistungsanbieter kümmert sich nur um die Wartung der Geräte. Im Folgenden werden die einzelnen Komponenten näher beschrieben.

7.2.1 Allgemeine Anbindungskomponente

Die allgemeine Anbindungskomponente (*aAk*) besteht aus einer Fernzugriffskomponenten und einer definierten Schnittstelle (siehe Abbildung 7.7). Über die Fernzugriffskomponente wird die Verbindung zur speziellen Anbindungskomponente eines eingebetteten Systems via Netzwerk hergestellt. Die Fernzugriffskomponente ermöglicht es, eine Verbindung mittels Sockets, RMI oder CORBA aufzubauen. Die Fernzugriffskomponente erlaubt den Aufruf der abstrakten Methode „callService“ bei der speziellen Anbindungskomponente (*sAk*). Die definierte Schnittstelle (*aAk-Schnittstelle*) stellt alle Funktionen der allgemeinen Anbindungskomponente bereit, unabhängig von der verwendeten Verbindungsart.

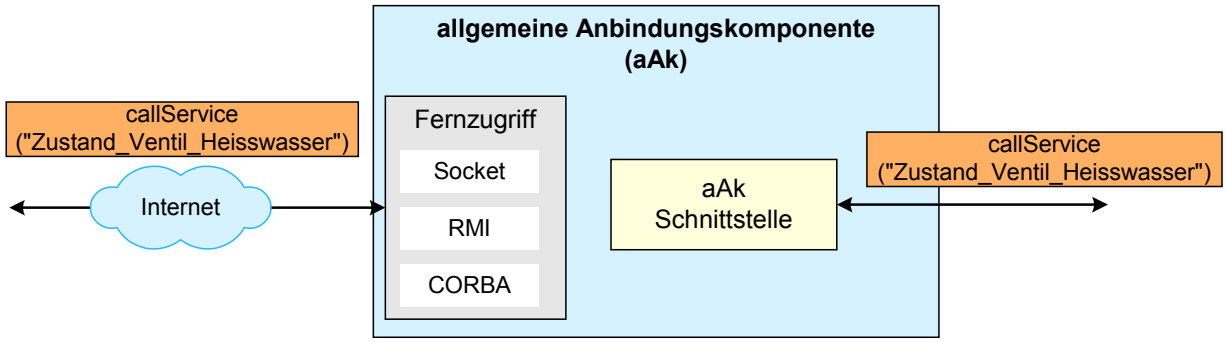

Abbildung 7.7: Aufbau der allgemeinen Anbindungskomponente

7.2.2 Aufbau der Datenaufbereitungskomponente

Die Datenaufbereitungskomponente bereitet die Daten, die durch einen Serviceaufruf geliefert werden, in die benutzergerechte Form des anfragenden Clienttyps auf. In der *Geräteinformation* ist dabei festgelegt, wie die Aufbereitung für den jeweiligen Clienttyp erfolgen soll. Abbildung 7.8 veranschaulicht den Aufbau der Datenaufbereitungskomponente.

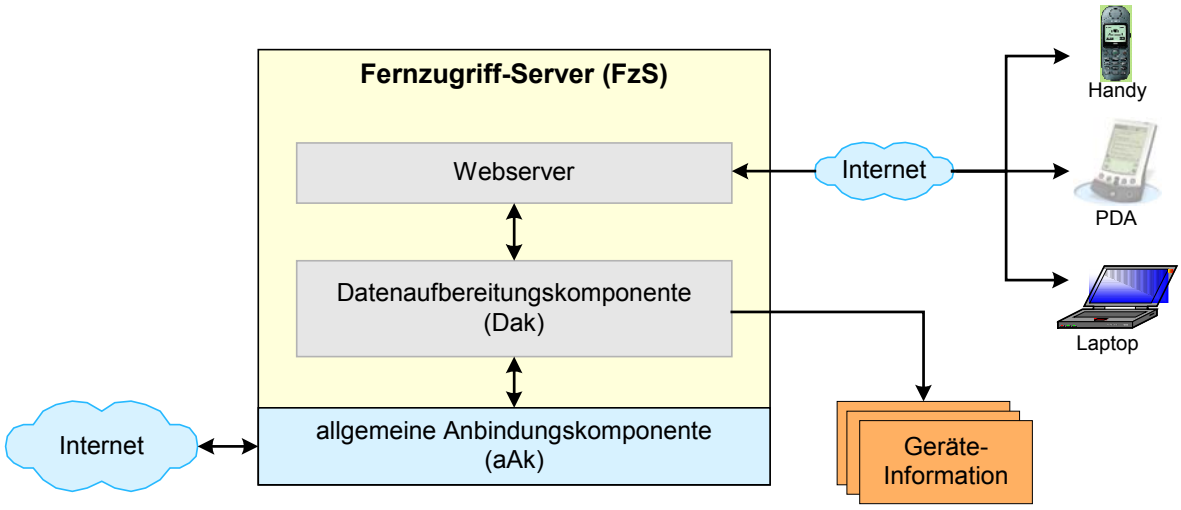

Abbildung 7.8: Aufbau der Datenaufbereitungskomponente

7.2.2.1 Funktionsweise der Datenaufbereitungskomponente

Die Anfrage eines Clients wird von der Datenaufbereitungskomponente entgegengenommen. Aus dieser Anfrage wird der aufzurufende Service extrahiert. Die Datenaufbereitungskomponente parametrisiert dementsprechend die abstrakte Methode „callService“ und ruft diese über die allgemeine Anbindungskomponente auf. Das Ergebnis des eigentlichen Serviceaufrufs beim Gerät wird als Rückgabewert der abstrakten Methode „callService“ in definierter Form zurückgeliefert. Die Datenaufbereitungskomponente generiert aus diesem Rückgabewert eine XML-

basierte Antwortsseite für den anfragenden Fernservice-Client. Der Aufbau der Seite richtet sich dabei nach der Beschreibung in der SDML-Instanz. Ebenso ist in der SDML-Instanz ein Verweis auf ein Stylesheet untergebracht, das mit dieser Seite verknüpft werden soll. Durch das Stylesheet wird die XML-basierte Antwortsseite vor der Auslieferung durch den Webserver in das richtige Ausgabeformat konvertiert. Abbildung 7.9 zeigt die Anfrage eines Clients, die durch die Datenaufbereitungskomponente bearbeitet wird.

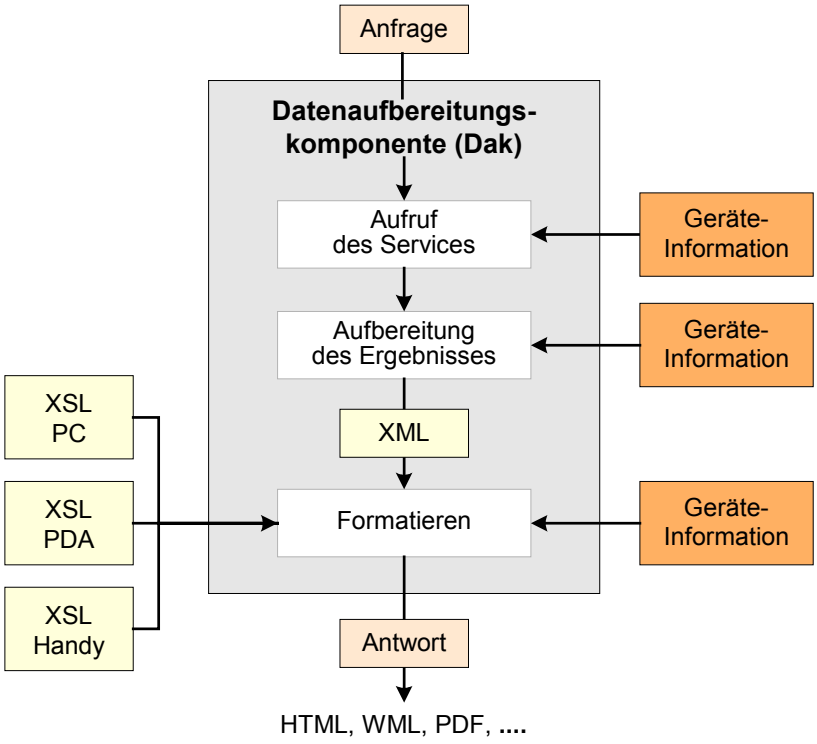

Abbildung 7.9: Funktionsweise der Datenaufbereitungskomponente

7.2.3 Aufbau des SDML-Editors

Im vorherigen Kapitel wurde die Spezifikation der Beschreibungssprache SDML erläutert. Dieser Abschnitt widmet sich dem SDML-Editor, der zur Entwicklung von SDML-Dokumenten konzipiert und realisiert wurde. Dazu wird zunächst die grafische Benutzungsoberfläche präsentiert, die eine komfortable Erstellung von SDML-Dokumenten ermöglicht. Danach werden Architektur und einzelne Bedienelemente der Software vorgestellt.

7.2.3.1 Grafische Benutzungsoberfläche des SDML-Editors

Nachdem die Spezifikation der SDML vorgestellt wurde, ist es offensichtlich, dass für die Entwicklung eines SDML-Dokuments zur Beschreibung einer Geräteinformation umfassende Kenntnisse über die Struktur der Beschreibungssprache SDML Voraussetzung sind. Außerdem muss der Entwickler des SDML-Dokuments unbedingt die Semantik der Sprache sehr gut be-

herrschen. Der Entwickler einer Geräteinformation ist in der Regel mit dem eingebetteten System vertraut und kennt dessen Dienste gut. Um nun den zusätzlichen Aufwand für die Erstellung der Geräteinformationen mittels SDML zu minimieren, wurde ein Editor entwickelt. Der Editor wurde durch den Einsatz von XML so spezifiziert und dokumentiert, dass der Anwendungsentwickler bei der Erstellung von SDML-Dokumenten möglichst umfassend unterstützt wird.

Der Editor besteht aus insgesamt vier Dialogfenstern, die über Tabulatoren auszuwählen sind. Die SDML-Auszeichnungselemente sind thematisch gruppiert auf folgende vier Dialoge verteilt:

- Author
- Server-Definition
- Service-Definition
- Datenaufbereitung (Data-Processing)

Abbildung 7.10 zeigt die wesentlichen Elemente der Benutzungsoberfläche, die zur Steuerung des Werkzeugs und zur Erstellung der SDML-Dokumente benötigt werden.

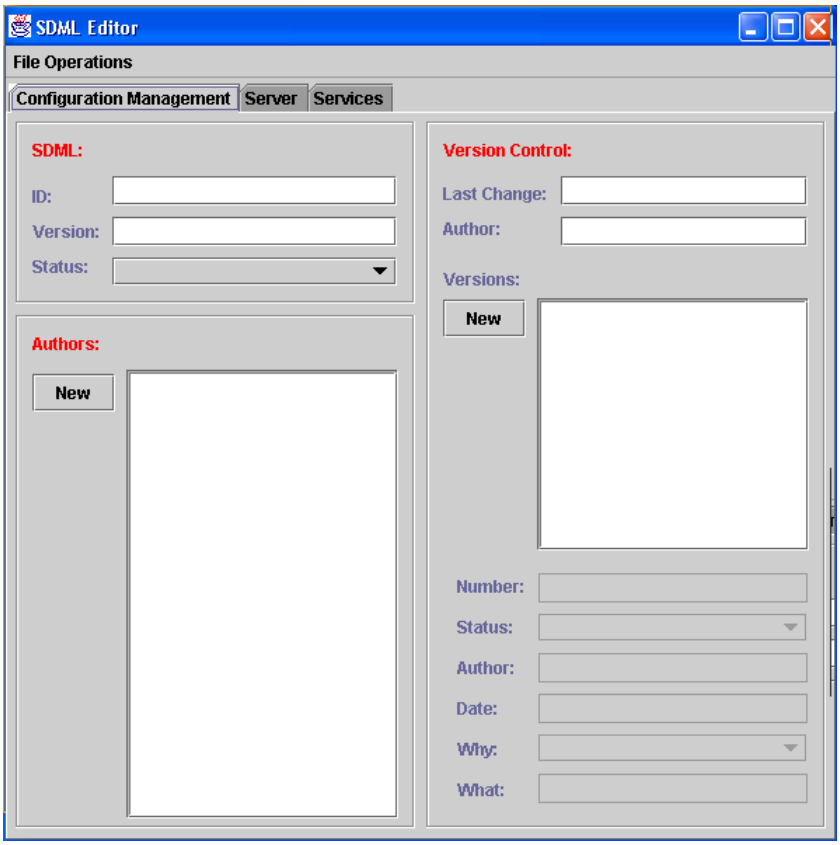

Abbildung 7.10: Benutzungsoberfläche des SDML-Editors

Das Werkzeug verfügt über ein Menü und drei Dialog-Tabs, die einen schnellen Zugriff auf die wichtigsten Funktionen des Werkzeuges ermöglichen. Mittels dieser Elemente der Benutzungsoberfläche ist eine intuitive Bedienung des Editors, wie z. B. das Öffnen bereits entwickelter SDML-Dokumente, das Anlegen neuer Dokumente, das Speichern von Eingabedaten des Anwendungsentwicklers oder die Generierung der Anwendungen möglich.

Die drei Dialog-Tabs dienen zur Beschreibung von Konfigurationsmanagement, Server-Definition und Services. Das Dialog-Tab „Service“ ist in zwei Bereiche unterteilt, einen für die Beschreibung der Services und einen für die Datenaufbereitung.

7.2.3.2 Software-Architektur

Maßgebend für den Entwurf der Software-Architektur sind folgende Anforderungen:

- *Unterstützung des Anwendungsentwicklers:* Dem Anwendungsentwickler soll ohne großen Einarbeitungsaufwand ein effizientes Arbeiten mit dem Editor ermöglicht werden.
- Funktionalität: Die Funktionen des Editors sollen stabil und fehlerfrei sein.
- Erweiterbarkeit: Da die SDML-Sprache weiter entwickelt wird, soll der Editor modular aufgebaut sein und leicht erweitert und modifiziert werden können.

Um die zuletzt genannten Anforderungen in Hinblick auf Erweiterbarkeit und Modifizierbarkeit des Werkzeuges zu erreichen, wurden weitere Entscheidungen gefällt, die nachfolgend benannt werden:

- Strikte Trennung zwischen Dokument (Daten) und Ansicht,
- ein der Struktur der SDML angepasstes Objekt-Modell,
- eine einheitliche Schnittstelle zur Darstellungskomponente,
- eine einheitliche Schnittstelle zur Erzeugung der SDML-Daten.

Der Editor wurde daher in zwei Komponenten gegliedert, die *View-Komponente* und die *Document-Komponente*. Die *View-Komponente* ist für die Visualisierung der Informationen zuständig. Sie besteht aus einem Dialog, der Daten aufnimmt und anzeigt. Die eingegebenen Daten werden in der *Document-Komponente* abgelegt. Sie stellt eine Objektsstruktur dar, welche dem Aufbau der SDML nachempfunden ist. Diese Objektsstruktur ist serialisierbar, d. h. sie kann auf permanenten Speichermedien gesichert und wiederhergestellt werden. Außerdem enthält jede Klasse die Funktionalität, einen String auszugeben, der die in ihr gekapselten Daten in SDML darstellt. Diese Funktionalität wird beim Erzeugen einer SDML-Datei aufgerufen.

Auf Grund positiver persönlicher Erfahrungen in vorangegangenen Projekten wurde die Programmiersprache Java als Implementierungssprache gewählt. Abbildung 7.11 zeigt die Softwa-

rearchitektur des SDML-Editors. Durch die strikte Trennung zwischen Daten und Ansicht einerseits und modular aufgebauter Benutzungsoberfläche andererseits wird eine Erweiterung des Editors gewährleistet. In den folgenden Abschnitten werden einzelne Bedienelemente und deren Aufgaben genauer betrachtet.

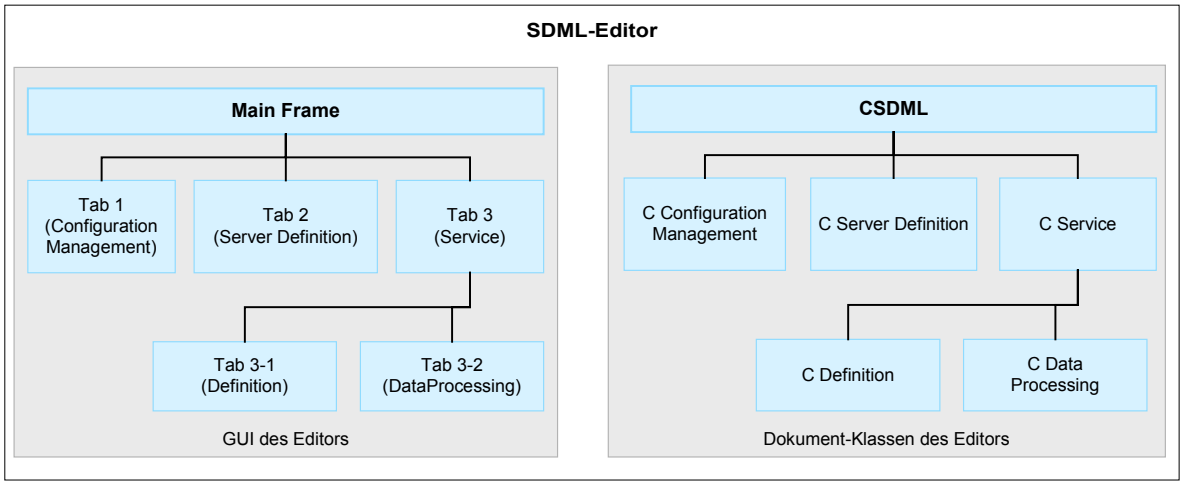

Abbildung 7.11: Statische Struktur des SDML-Editors

Bedienelemente des SDML-Editors

File Operations

Das Menü File Operations hat drei Funktionalitäten (siehe Abbildung 7.12). Mittels *Load file* kann eine vorher gespeicherte SDML-Datei geladen werden. Diese Dateien enthalten die in den Editor eingegebenen Daten. Die eingegebenen Daten können in einer SDML-Datei gespeichert werden (*Save file*). Mit Hilfe der Funktion *Create SDML file* kann eine SDML-Datei aus den in den Editor eingegebenen oder geladenen Dateien erzeugt werden.

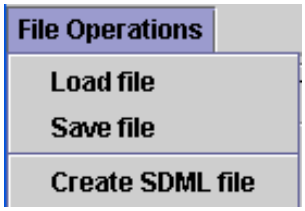

Abbildung 7.12: Menü File Operation

Configuration Management

Diese Ansicht stellt Informationen über die Version der SDML sowie über Autoren und Versionen dieser SDML-Datei dar (siehe Abbildung 7.13). Zunächst kann im Bereich SDML eine ID für das zu erstellende Dokument, die Versionsnummer und der Status eingegeben werden. Im

Bereich *Version Control* werden Informationen über letzte Änderungen eingetragen. Falls es sich um eine neue Version handelt, können die Informationen über diese neue Version im Feld *Versions* eingegeben werden. Das Ergebnis der eingegebenen Daten in Abbildung 7.13 führt zu einem SDML-Dokument, das in Abbildung 7.14 zu sehen ist.

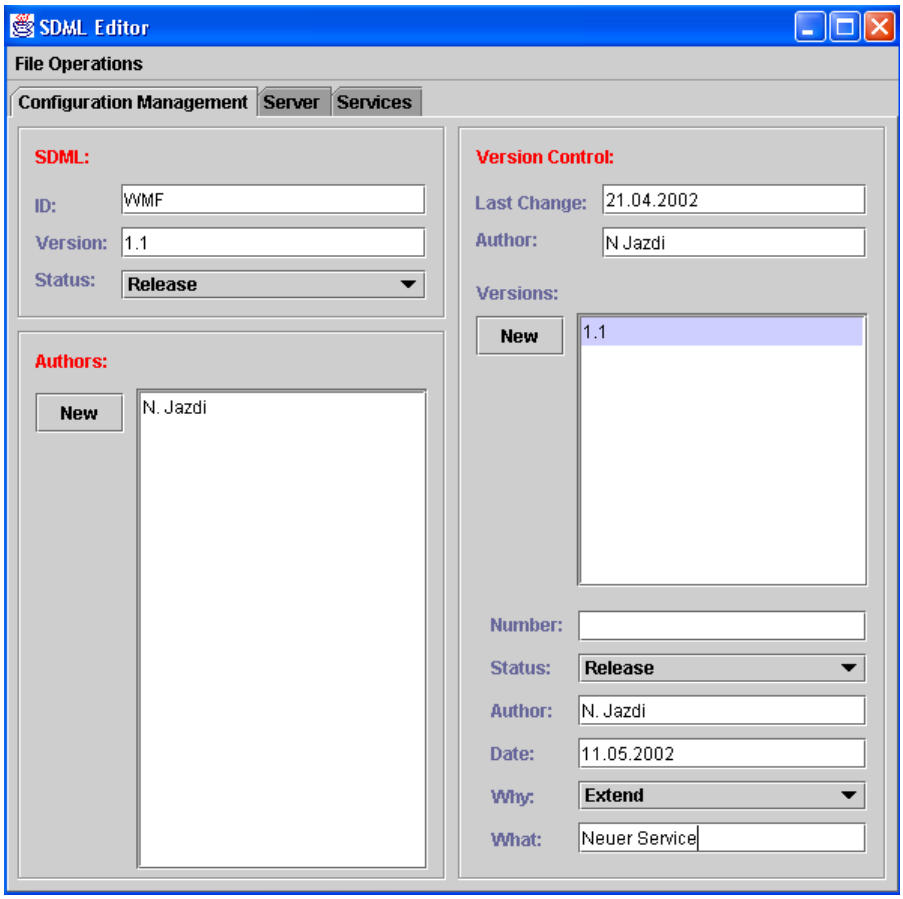

Abbildung 7.13: Dialogfeld für Konfigurationsmanagement-Daten

```
1:    <?xml version="1.0" encoding="UTF-8" ?>
2:    <sdml id="wmf" version="1.1" status="Release">
3:     <ConfigurationManagement>
4:      <ListOfAuthors>
5:      <Author name="N. Jazdi" />
6:      </ListOfAuthors>
7:     <VersionsControlling lastChange="21.04.2002" authorName="N.Jazdi">
8:      <Version number="" status="Release"
9:       authorName="N.Jazdi" date="11.05.2002"
10:      why="extend" what="Neuer Service">1.1</Version>
11:    </VersionsControlling>
12:    </ConfigurationManagement>
13:    </sdml>
```

Abbildung 7.14: Generiertes SDML-Dokument aus Abbildung 7.13

Server

Die Information über den Server ist ein wesentlicher Teil eines SDML-Dokuments, da an einen Fernzugriff-Server mehrere eingebettete Systeme angebunden werden können. Diese müssen eindeutig identifizierbar sein. Der Fernzugriff-Server muss außerdem über die Art der Verbindung zu jedem Gerät informiert werden. Die Ansicht *Server* im SDML-Editor bietet die Möglichkeit zur Eingabe dieser Informationen (siehe Abbildung 7.15).

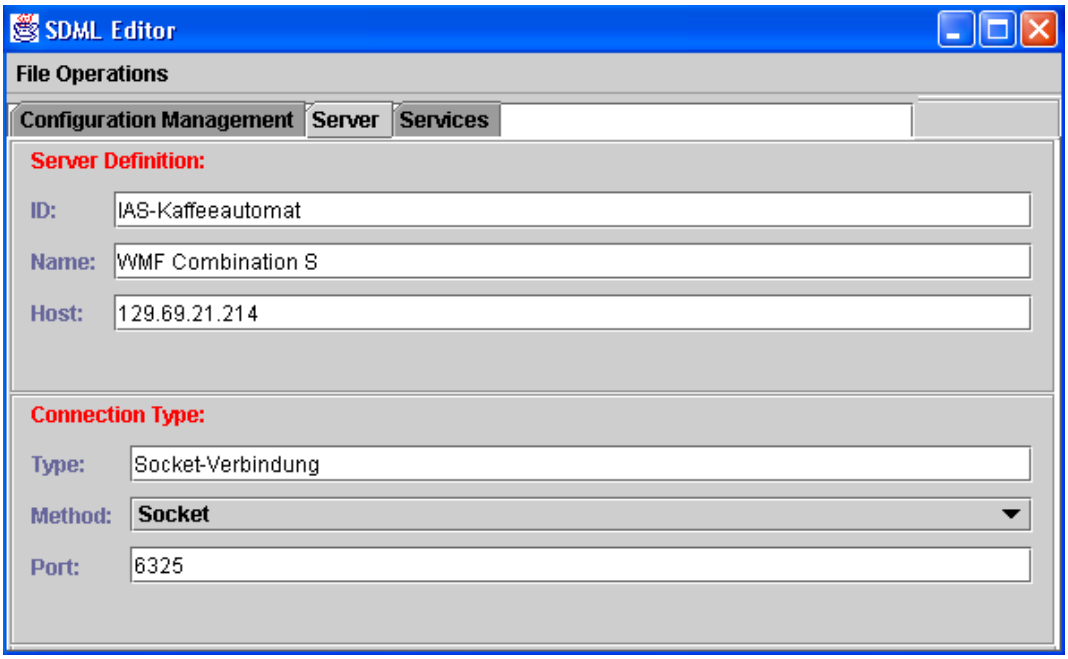

Abbildung 7.15: Dialogfeld für Server-Daten

Neben `ID` und `Name` des eingebetteten Systems muss die `Host`-Adresse (URL-Adresse) im Bereich *Server Definition* eingegeben werden. Der Bereich *Connection Type* ist für Informationen über die Art der Verbindung reserviert.

```
1:    <?xml version="1.0" encoding="UTF-8" ?>
2:    <sdml id="wmf" version="1.1" status="Release">

3:     <ServerDefinition id="IAS-Kaffeeautomat"
4:      name="WMF Combination S" host="129.69.255.214">
5:     <ConnectionType method="socket" port="6325">Socket-Verbindung
6:     </ConnectionType>
7:     </ServerDefinition>
8:    </sdml>
```

Abbildung 7.16: Generiertes SDML-Dokument aus Abbildung 7.15

Services

Die Ansicht *Service* dient zur Definition der Dienste eines eingebetteten Systems. Die Anzahl der Dienste ist unbegrenzt und kann beliebig wiederholt werden. Die Ansicht *Services* ist in zwei weitere Fenster (Sub-Panes), *Definition* und *Data-Processing*, unterteilt. Abbildung 7.17 zeigt die Ansicht *Services/Definition*.

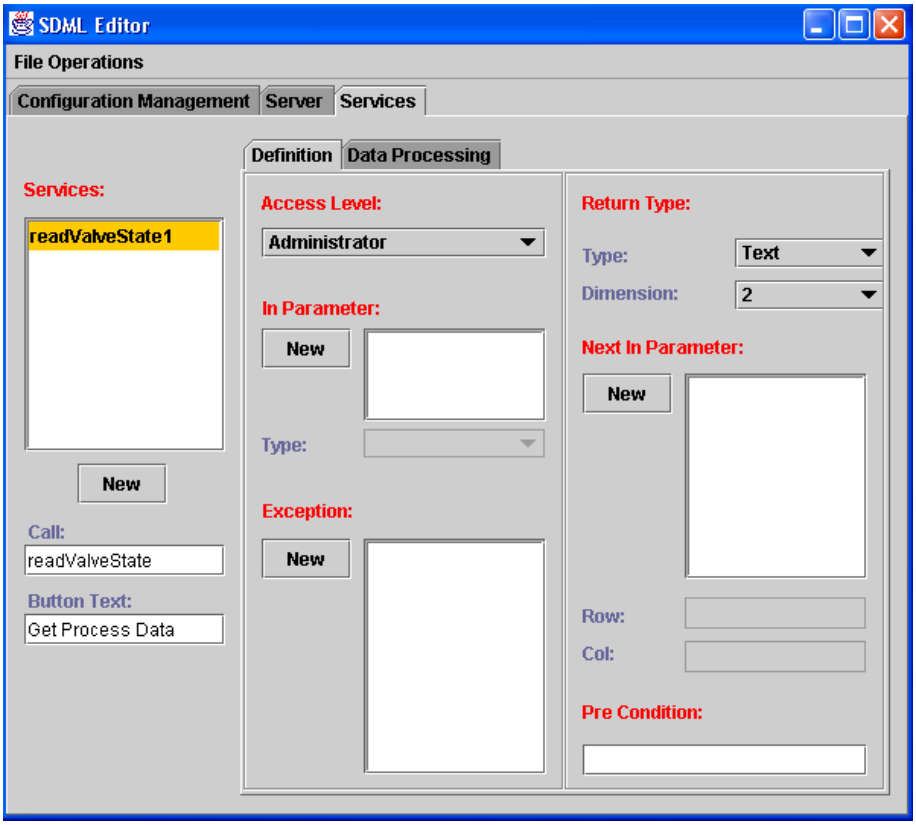

Abbildung 7.17: Dialogfeld für die Definition der Services

Im Feld *Services* kann mittels Auswahl-Button *New* ein neuer Service definiert werden. Im Textfeld *Call* wird angegeben, wie der Service genannt werden soll. Im Listenfeld *Access Level* kann entschieden werden, wer Informationen zu sehen bekommt. Derzeit ist eine Unterscheidung zwischen Gast, Anwender und Administrator vorgesehen. Im Beispiel aus Abbildung 7.17 wurde die Option Administrator ausgewählt, was bedeutet, dass nur die Administratoren diese Information abfragen dürfen. Im Bereich *Return Type* kann Typ und Dimension des Rückgabewerts bestimmt werden. Über *In Parameter* kann ein Parameter definiert werden, der für den Serviceaufruf benötigt wird. Ist im Feld *Pre Condition* ein Serviceaufruf als Vorbedingung festgelegt, so wird der Parameter aus dessen Rückgabewert ermittelt. Der Eintrag im Feld Exception veranlasst, dass beim Auftreten eines Fehlers während des Aufrufs eines Service eine Nachricht an den Benutzer gesandt wird. Abbildung 7.18 zeigt das generierte SDML-Dokument aus Abbildung 7.17.

```
1:    <?xml version="1.0" encoding="UTF-8" ?>
2:    <sdml id="wmf" version="1.1" status="Release">

3:     <ListOfServices>
4:      <Service>
5:       <Definition id="readValveState1" call="readValveState"
6:         buttontext="Get Process Data">
7:       <ReturnType type="text" dimension="2" />
8:       <Exception id="Fehler am Sensor" />
9:       <AccessLevel level="3" />
10:      </Definition>
11:     </Service>
12:    </ListOfServices>

13:   </sdml>
```

Abbildung 7.18: Generiertes SDML-Dokument aus Abbildung 7.17

Ein weiterer Unterbereich der Ansicht *Services* ist *Data-Processing*, welches für die Datenaufbereitung der Prozessdaten für verschiedene Clients zuständig ist. Abbildung 7.19 zeigt die Ansicht *Services/ Data-Processing*.

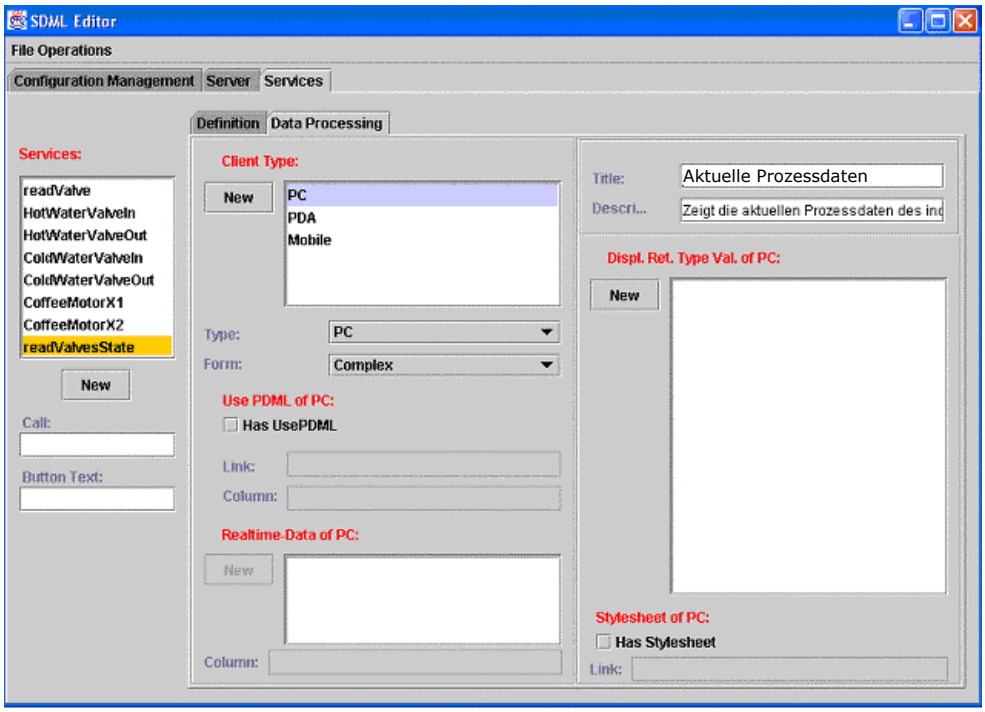

Abbildung 7.19: Dialogfeld für Datenaufbereitung

Mit Hilfe dieses Dialogfelds kann bestimmt werden, wie die angeforderten Prozessdaten bei den jeweiligen Clients aussehen sollen. Falls zur Darstellung der Prozessdaten bei einem Client ein

Stylesheet existiert, kann dies im Feld *Link* eingetragen werden. Abbildung 7.20 zeigt das generierte SDML-Dokument aus Abbildung 7.19.

```
1:    <?xml version="1.0" encoding="UTF-8" ?>
2:    <sdml id="wmf" version="1.1" status="Release">

3:     <DataProcessing>
4:      <Title>Aktuelle Prozessdaten</Title>
5:      <Description>Zeigt die aktuelle Prozessdaten
                     des industriellen Kaffeeautomaten
6:      </Description>
7:      <ClientType type="pc" form="complex">
8:      <UsePDML link="" col="" />
9:      <Stylesheet link="visualisierung.xsl" />
10:    </DataProcessing>

11:    </sdml>
```

Abbildung 7.20: Generiertes SDML-Dokument aus Abbildung 7.19

7.3 Anwender-Schicht

Der Anwender (Client) verwendet einen gängigen Browser zur Kommunikation mit dem Fernzugriff-Server. Der Verbindungsaufbau erfolgt mittels URL (Uniform Resource Locator). Der Nutzer gibt die URL-Adresse des Fernzugriff-Servers ein und gelangt damit zur Startseite dieses Servers. Um die gesamte Kommunikation zwischen Anwender (Client) und Fernzugriff-Server erläutern zu können, muss zunächst der Begriff *Session* definiert werden.

Begriffsdefinition *Session*

„Eine Sitzung (engl. Session, Anm. des Verf.) besteht aus einer Reihe von Request-Response-Transaktionen mit demselben Client.“ [NaMy00]

Bei der ersten Anfrage an den Fernservice-Server wird der Anwender aufgefordert, sich einzuloggen. Durch den Benutzernamen und das Passwort wird die Zugangsberechtigung des Anwenders ermittelt. Auf Grund dieser Berechtigung ergeben sich im weiteren Verlauf die Services, die für **diesen** Anwender zur Auswahl stehen. Die Auswahl der zur Verfügung stehenden Service ist vom jeweiligen Clienttyp abhängig. Hat der Anwender einen Service ausgewählt, erscheint das Ergebnis des Serviceaufrufs auf dem nächsten Screen[5]. Hier können dann weitere Service zur Auswahl stehen, die eventuell durch das gezeigte Ergebnis parametrisiert werden.

[5] Screnn = Bildschirmseite

Zudem hat der Anwender auf jeder Seite die Möglichkeit, die Session zu beenden oder ins Hauptmenü zurückzukehren. Abbildung 7.21 beschreibt den Ablauf einer Session.

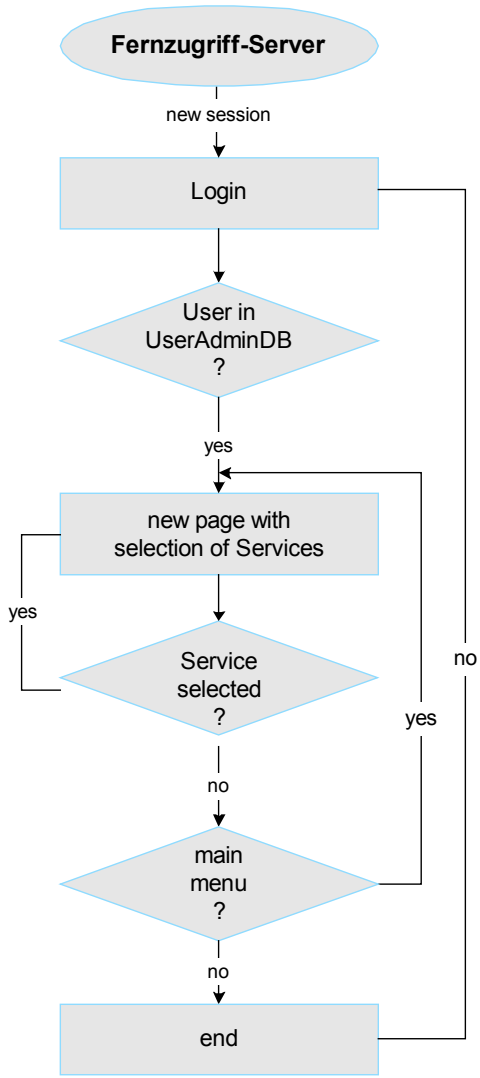

Abbildung 7.21: Ablauf einer Sitzung

Der Einsatz eines PDA als Client funktioniert genau wie bei einem PC, da für beinahe alle Geräte der aktuellen Generation von Personal Digital Assistants inzwischen einfache Browser vorhanden sind. Allerdings kennen diese Browser meist keine aktiven Elemente wie JavaScript, Applets oder ActiveX-Controlls, sondern verstehen ausschließlich HTML 2.0. Diese Einschränkung wirkt sich in einer reduzierte Möglichkeit der Darstellung der Information aus, sodass komplexe Bilder vermieden werden müssen.

Für Geräte mit sehr kleinen Displays, die nur etwa 10 bis 20 Wörter in 2 bis 5 Zeilen darstellen können, wie ein mobiles Telefon, müssen die Informationen im Allgemeinen extra aufbereitet werden, und zwar in einem stark gekürzten und wesentlich kompakteren Format. Da solche Geräte meist auch über geringere Rechen-Ressourcen verfügen als PCs oder Workstations, muss

außerdem ein Format und ein Protokoll verwendet werden, das möglichst effizient verarbeitet werden kann.

Der Anwender, der sich mittels eines mobilen Telefons mit dem FzS verbinden möchte, muss ein WAP-fähiges (Wireless Application Protocol) Gerät verwenden, das das WML-Protokoll (Wireless Markup Language) unterstützt. Im Folgenden werden die zwei Protokolle kurz erläutert.

WAP

1997 wurde das WAP-Forum gegründet (u. a. von Ericsson, Motorola und Nokia), das einen neuen Standard für die kabellose Kommunikation verschiedener Endgeräte (Palmtops, Mobiltelefone etc.) mit verschiedenen Betriebssystemen (Palm OS, Windows CE, EPOC) über unterschiedliche Netze (GSM, UMTS etc.) entwickeln sollte. Besonders wurde dabei auf die sichere Übertragung der Daten zwischen zwei Anwendern geachtet. Entwickelt wurde schließlich WAP, das inzwischen de facto Standard für die Kommunikation von mobilen Telefonen mit dem Internet bildet.

Bei der Entwicklung des WAP-Standards wurde besonders auf die eingeschränkten Möglichkeiten der mobilen Endgeräte Rücksicht genommen. Diese besitzen zugunsten längerer Akkulaufzeiten oftmals eine relativ langsame Rechenleistung und nur wenig Speicher. Auch bleibt auf den immer kleiner werdenden Geräten kaum noch Platz für ein Anzeige-Element. Jeder Punkt des kleinen Bildschirms ist also kostbar und muss möglichst effizient genutzt werden. Klassische WAP-Anwendungen kommen mit gerade mal vier Zeilen à 10 Zeichen aus. Je nach Endgerät hat der Anwender unterschiedliche Möglichkeiten, mit der Anwendung zu kommunizieren: Mal ist eine Stifteingabe vorgesehen, mal ist eine winzige Schreibmaschinen-Tastatur im Gehäuse zu finden und oftmals müssen sich Mobiltelefon-Besitzer mit 12 kleinen Tasten zufrieden geben. Auf all das galt es während der Entwicklung des Standards Rücksicht zu nehmen.

Die Anwendungen sollen jedoch nicht nur den kleinsten gemeinsamen Nenner unterstützen, sondern müssen „skalierbar“ sein. Das heißt, ein Anwender, der mit einem farbfähigen Endgerät via Breitband-Übertragung auf die Dienstleistung zugreift, muss sich nicht mit einer briefmarkengroßen Schwarzweiß-Oberfläche begnügen, sondern die Anwendungen lassen sich benutzerspezifisch anpassen. Des Weiteren greifen Dienste, die über den WAP-Standard Daten austauschen, nicht nur auf spezifische Möglichkeiten eines bestimmten Netzes zurück, sondern sind in ihrer Funktionalität von den darunter liegenden digitalen Netzen unabhängig.

Der Datenaustausch zwischen Kunden und Dienstleistungsanbieter erfolgt im WAP-Modell nicht direkt. Im Gegensatz zum „konventionellen“ Internetzugriff befinden sich beide Geräte in unterschiedlichen Netzen bzw. Netzwerksystemen. Ein WAP-Zugangsrechner (WAP-Gateway) übernimmt die Vermittlerrolle zwischen drahtgebundenem und kabellosem Netzwerk. Dabei erfolgt die Kommunikation zwischen Zugangsrechner und Dienstleistungs-Anbieter wie gewohnt

über das HTTP-Protokoll. Bei der drahtlosen Übertragung zum Endanwender kommt das WAP-Protokoll zum Einsatz. Ähnlich wie bei der Datenübertragung im Internet via TCP/IP besteht auch das WAP-Protokoll aus verschiedenen Schichten, also spezialisierten und gruppierten Merkmalen des Protokolls, die aufeinander aufbauen. Jede „Schicht“ ist dabei durch ihre eigenen Richtlinien, gewissermaßen „Unterprotokolle“, geprägt.

Auf das netzabhängige System, das nicht direkt mit WAP in Verbindung steht und sozusagen als „Träger-Medium“ der Kommunikation fungiert, setzt die Übertragungsschicht auf. Das „Datenpaket-Protokoll“ (Wireless Datagramm Protocol) übernimmt die Funktion des Vermittlers zwischen dem Träger und der weiteren Verarbeitung der Daten. Dabei wird auf die spezifischen Eigenschaften des Trägers eingegangen, zum Beispiel auf eine niedrige Bandbreite. Dazu wurde eine neue Sprache entwickelt, die den Anforderungen mobiler Endgeräte gerecht werden soll, die „drahtlose Markierungssprache“ WML, eine Ausprägung von XML. Die WML Sprache wird im Folgenden kurz beschrieben.

WML

Im Gegensatz zu HTML enthält WML[6] eine Art „Ereignismodell“, das dem Entwickler erlaubt, einfache Funktionen wie das Zurücknavigieren o.Ä. als Reaktion auf ein bestimmtes Ereignis (Drücken einer Taste, Ablauf einer Stoppuhr ...) zu definieren.

Diese Möglichkeiten sind jedoch stark eingeschränkt. Zum Beispiel lassen sich Berechnungen mit WML nicht durchführen. Dafür gibt es jedoch WMLScript, eine Sprache, mit der sich Programm-Fragmente in WAP-Seiten einbinden lassen. Dieses Programm wird auf dem Endgerät ausgeführt, um z. B. vor der Übertragung Formulardaten auf ihre Gültigkeit hin zu überprüfen. Da die Bandbreite der Netze jedoch sehr gering ist, wird WMLScript, wie übrigens WML-Seiten auch, direkt in Binärdarstellung zum Endgerät übertragen. Einziges Manko beim Einsatz von WMLScript ist die bisher fehlende Unterstützung durch Endgeräte.

Seiten, die durch die Markierungssprache WML beschrieben werden, halten ihre Informationen auf so genannten „Karten“, die jeweils einer Seite auf dem Anzeigeelement des Endgerätes entsprechen. Mehrere Karten werden in einem Karteikasten (engl. „deck“) zusammengefasst und jeder Kasten entspricht einer Datei im Dateisystem des Dienstleistungsrechners.

Konventionsgemäß enden die Namen dieser Dateien mit „.wml“. Da die Speicher der mobilen Geräte sehr eingeschränkt sind, muss man darauf achten, dass die gesamte Datei nicht größer als 1.400 Zeichen wird – ansonsten erscheint womöglich gar nichts beim Anwender. Aufgrund dieser Einschränkung soll besonders darauf geachtet werden, welche Fernservice-Funktionen den „Mobilen Telefon-Clients“ zur Verfügung gestellt werden.

---

[6] Wireless Markup Language

Abbildung 7.22 zeigt ein Beispiel für eine WML-Datei.

```
1:    <?xml version="1.0" encoding="UTF-8" ?>
2:    <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
      "http://www.wapforum.org/DTD/wml_1.1.xml">

3:    <wml>
4:      <card id="AktuelleProzessdaten" title="Prozessdaten des IAS-KA">
5:        <p> Ventil_Heisswasser offen! </p>
6:      </card>
7:    </wml>
```

Abbildung 7.22: Aufbau einer WML-Datei

Wie jedes XML-Dokument beginnt auch eine WML-Datei mit der obligatorischen Kopfzeile. In der ersten Zeile wird angezeigt, dass WML von der XML-Version 1.0 abstammt. Die zweite Zeile legt den Dokumententyp fest und spezifiziert den verwandten WML-Standard, hier 1.1. Untergeordnet befinden sich die einzelnen Karten des Karteikastens, jeweils in „card“-Tags eingebettet. Das Attribut „id“ stellt dabei eine eindeutige Bezeichnung der Karte innerhalb des Stapels und „title“ die Überschrift, die im Anzeigeelement des mobilen Gerätes eingeblendet wird, dar. Die Karten enthalten die Information, wobei wie in HTML die Texte jeweils in Abschnitte durch <p> markiert und eingeteilt werden.

Bei der Anzeige im Endgerät fällt auf, dass schon relativ viel Anzeigefläche verbraucht wurde, obwohl der Informationsgehalt der WAP-Seite noch nicht besonders groß ist. Bei der Entwicklung von Internet-Seiten für mobile Geräte steht demnach der Informationsgehalt im Vordergrund, das Design ist fast zu vernachlässigen. Daher haben Auszeichnungen wie *fett*, *kursiv* oder *unterstrichen* eine größere Bedeutung als bei herkömmlichen Internetseiten. Bei WML-Seiten kommen die eingeschränkten Anzeigefähigkeiten des Endgerätes hinzu: Kann ein Entwickler bei einem normalen Monitor immerhin noch von mindestens 640x480 Bildpunkten ausgehen, muss der Entwickler von WML-Seiten mit einer durchschnittlichen Auflösung von 60x40 Punkten auskommen. Auch ist die Implementierung je nach Endgerät sehr unterschiedlich. Dies beschränkt sich nicht nur auf eine variierende Anzeige, sondern die Bedienung der WAP-Anwendung ist zudem nicht eindeutig festgelegt.

Darstellungsprobleme werden bei HTML-Seiten oftmals durch den Einsatz von Grafiken umgangen. Auch WAP sieht eine Darstellung von Grafiken vor, allerdings wird nur das Format WBMP (Wireless Bitmaps, drahtlose pixelorientierte Grafiken) unterstützt. Dieses Format beinhaltet lediglich 1 Bit Farbtiefe, jeder Punkt kann also entweder nur schwarz oder weiß sein. Sinnvolle Größen für WBMP-Bilder sind ca. 32 auf 32 Bildpunkte, da größere Grafiken evtl. von den Endgeräten nicht mehr dargestellt werden können. Die Grafiken werden wie bei HTML über die <img>-Markierung eingebunden. Neben der einzubindenden Grafik muss hierbei auch noch ein Alternativtext angegeben werden, falls die Grafik nicht geladen werden kann. Die Dar-

stellung von Grafiken ist beim Zugriff auf das Internet vom Mobiltelefon aus also stark eingeschränkt.

Eine weitere gravierende Einschränkung bei der Entwicklung der WML-Seiten ist auch der geringe Speicher der mobilen Endgeräte. Eine Seite, die auf allen gängigen Geräten angezeigt werden soll, darf insgesamt nicht mehr als 1400 Zeichen groß sein. Die maximale Größe der Dateien variiert aber von Gerät zu Gerät, und der Entwickler hat es hier nicht wie im WWW mit nur zwei bis drei konkurrierenden Interpreter-Systemen zu tun, sondern mit über einem Dutzend.

8 Fallbeispiele

In den vorangegangenen Kapiteln wurde ein Verfahren zur Anbindung verschiedener eingebetteter Systeme an einen Fernzugriff-Server vorgestellt. Dadurch ist der Fernzugriff-Server in der Lage, mit den Geräten zu kommunizieren, auf gleiche Art und Weise deren Prozessdaten abzufragen und sie bei unterschiedlichen Clients in ein jeweils angepasstes Format zu überführen und darzustellen. Um das Verfahren zu evaluieren, wurde zunächst ein Fernzugriff-Server entwickelt. An diesen Fernzugriff-Server wurden zwei eingebettete Systeme, ein industrieller Kaffeeautomat und ein Aufzugsmodell angebunden, die am Institut für Automatisierungs- und Softwaretechnik (IAS) der Universität Stuttgart zur Verfügung stehen.

In diesem Kapitel wird zunächst die Realisierung des Fernzugriff-Servers erläutert. Danach werden die eingebetteten Systeme und die dafür entwickelten Komponenten vorgestellt. Anschließend wird das gesamte System und folgende drei Clients, ein PC, ein PDA und ein mobiles Telefon, vorgestellt. Zum Schluss werden die dabei gesammelten Erfahrungen diskutiert.

8.1 Systemarchitektur

Für die Evaluierung des Verfahrens wurde eine komplette Applikation (bestehend aus Ferndiagnose, Fernvisualisierung, Fernwartung und Fernbedienung) realisiert. Sie besteht aus zwei eingebetteten Systemen, dem Kaffeeautomaten und dem Aufzugsmodell, einem Fernzugriff-Server und mehreren Clients. Diese sind jeweils über das Internet miteinander verbunden. Abbildung 8.1 zeigt die Systemarchitektur der realisierten Applikation.

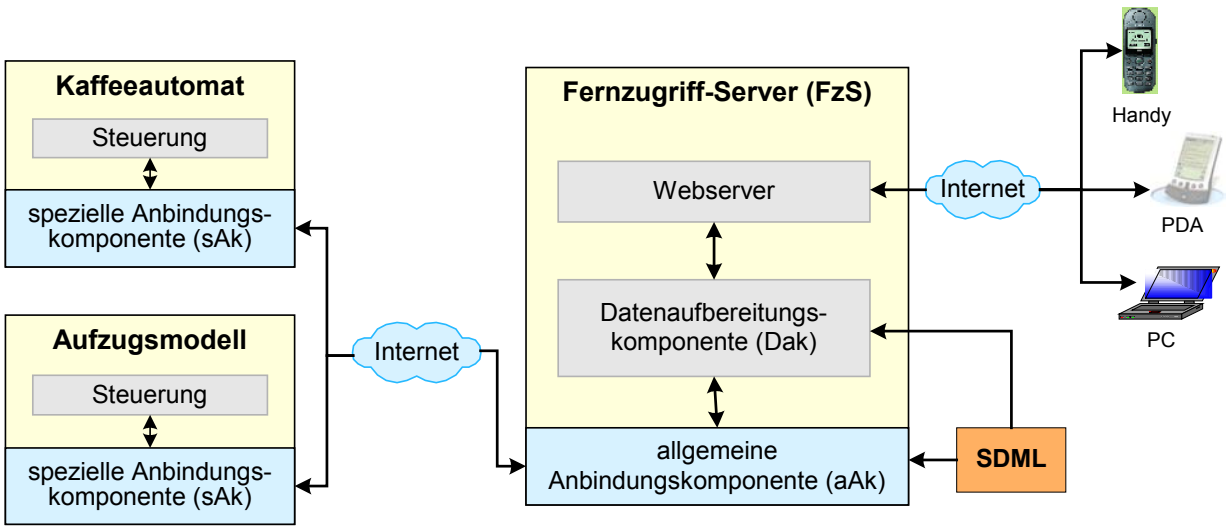

Abbildung 8.1: Systemarchitektur der Applikation

8.2 Beschreibung der Systemkomponenten

8.2.1 Clients

Als Clients kommen ein Standard-PC, ein Personal Digital Assistent (PDA) und ein mobiles Telefon zum Einsatz. Die Darstellung der Prozessdaten auf dem PC erfolgt mit Hilfe eines Webbrowsers. Der PC muss daher das Hyper Text Transfer Protokoll (HTTP) ab der Version 1.0 unterstützen. Auf PDA, ein Compaq-Gerät, muss ebenfalls ein Webbrowser installiert sein. Das mobile Telefon ist ein WAP-Handy, da nur dieses die W*ireless Markup Language* (WML) für die Präsentation der Prozessdaten unterstützt.

8.2.2 Fernzugriff-Server

Der Fernzugriff-Server besteht aus drei Komponenten: dem Webserver, der Datenaufbereitungskomponente und der allgemeinen Anbindungskomponente. Zudem verfügt er über zwei SDML-Dokumente, die aber kein notwendiger Bestandteil des Fernzugriff-Servers sind, jedoch für die Anbindung der eingebetteten Systeme benötigt werden. Sie wurden in dieser Realisierung auf dem Fernzugriff-Server zur Verfügung gestellt, können aber auch an einem anderen Ort im Internet abgelegt werden. Im Folgenden werden die Bestandteile des Fernzugriff-Servers erläutert.

Webserver

Der Webserver ist für die Kommunikation mit dem Webbrowser des Clients verantwortlich. Er nimmt die Anfragen des Clients entgegen und übermittelt diese über die Servlet-Engine *Tomcat* und das XSP-Servlet *Cocoon* an die Datenaufbereitungskomponente (Dak). Die generierten Antwort-Seiten des Fernzugriff-Servers werden dann vom Webserver an den anfragenden Client übermittelt. Abbildung 8.2 stellt diese Kommunikation grafisch dar.

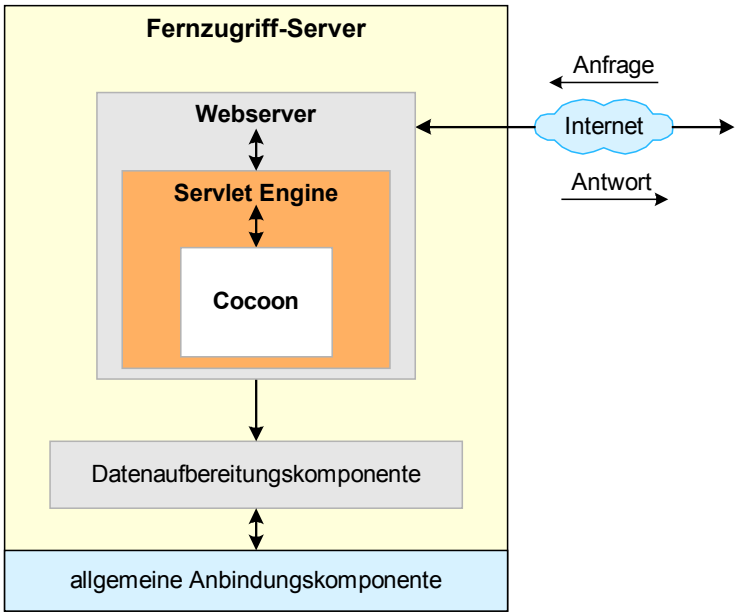

Abbildung 8.2: Webserver für die Kommunikation zwischen Client und Datenaufbereitungskomponente

Datenaufbereitungskomponente

Die Anfrage eines Clients mittels des Webservers wird durch die Datenaufbereitungskomponente (Dak) an die allgemeine Anbindungskomponente weitergeleitet. Diese veranlasst den Aufruf des entsprechenden Service bei der speziellen Anbindungskomponente des entsprechenden Geräts. Das Ergebnis dieses Serviceaufrufs wird anschließend von der Datenaufbereitungskomponente entsprechend der Beschreibung im SDML-Dokument für den anfragenden Client aufbereitet. Dieser Sachverhalt ist in Abbildung 8.3 dargestellt.

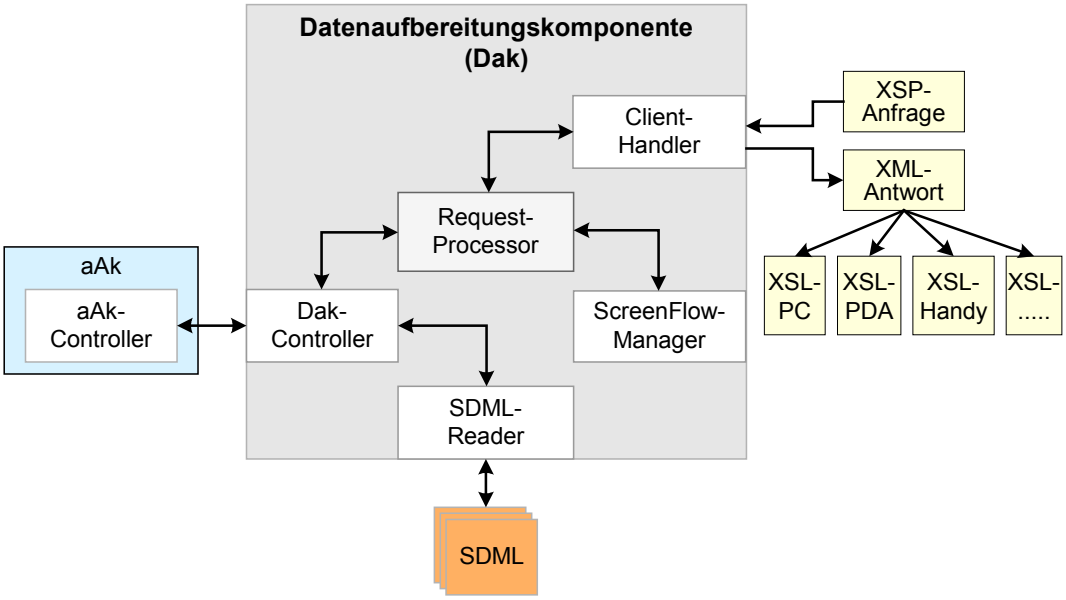

Abbildung 8.3: Aufbau der Datenaufbereitungskomponente

Die einzelnen Bestandteile der Datenaufbereitungskomponente haben folgende Aufgaben:

- Der `ClientHandler` übernimmt sämtliche Anfragen der Clients und fungiert als einziger Zugang zur Dak.
- Der `ScreenFlowManager` ermittelt aus der aktuellen Anfrage und den darin enthaltenen Parametern die nächste aufzurufende Seite.
- Der `RequestProcessor` ist das Herzstück der Datenaufbereitungskomponente. Er bearbeitet die über den ClientHandler kommenden Anfragen. Hier werden Korrektheit und Vollständigkeit der zugehörigen Parameter überprüft. In dieser Komponente steckt somit beinahe die gesamte Anwendungslogik für die Datenaufbereitung. Diese Komponente kann bei größeren Systemen bei Bedarf in weitere Komponenten untergliedert werden.
- Der `SDMLReader` liest das SDML-Dokument ein und ist für den kontrollierten Zugriff auf dessen einzelne Elemente zuständig.
- Der `DakController` stellt den kontrollierten Zugang zur allgemeinen Anbindungskomponente her.
- Der `aAkController` stellt den kontrollierten Zugang zur Datenaufbereitungskomponente her.

Allgemeine Anbindungskomponente

Die allgemeine Anbindungskomponente stellt die Verbindung über das Intranet/Internet mit einer speziellen Anbindungskomponente des eingebetteten Systems her. Die zentrale Zugangsstelle für die allgemeine Anbindungskomponente bildet der aAk-Controller. Hier sind alle Funktionalitäten abgelegt, die von dieser Komponente angeboten werden. Die allgemeine Anbindungskomponente unterstützt Verbindungen mittels Socket, RMI oder CORBA. Für jede dieser Verbindungsarten gibt es wiederum eine Komponente, die alle Funktionalitäten repräsentiert. Bei der Kommunikation übernimmt die aAk die Rolle eines Clients. Abbildung 8.4 zeigt die Architektur der allgemeinen Anbindungskomponente.

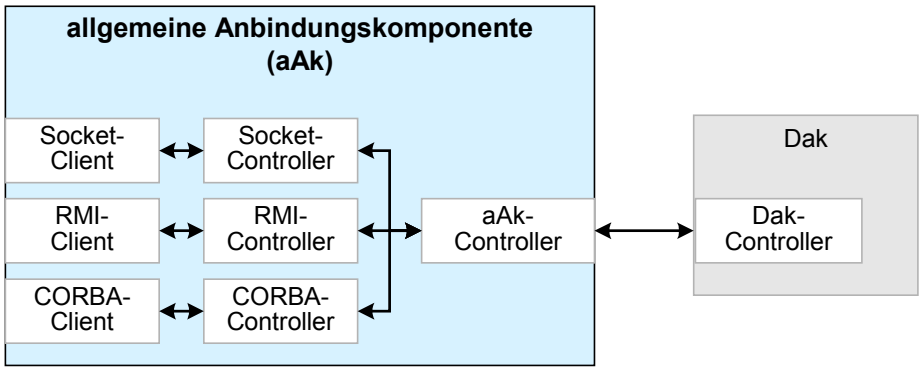

Abbildung 8.4: Architektur der allgemeinen Anbindungskomponente

SDML-Dokumente

Es gibt zwei SDML-Dokumente, die für den Kaffeeautomaten (*WMF_Combination_SDML.xml*) bzw. für das Aufzugsmodell (*Aufzug_SDML.xml*) verwendet werden. Sie beinhalten Informationen über die Art der Verbindung zum Gerät und die aufzurufenden Prozessdaten.

8.2.3 Automatisierungsgeräte

Als Automatisierungsgeräte werden zwei Geräte an den Fernzugriff-Server angebunden, der industrielle Kaffeeautomat und das Aufzugsmodell. Sie stellen die Prozessdaten bereit, die von den Clients abgefragt werden können. Im folgenden Abschnitt werden die Geräte und die zur Anbindung entwickelten spezifischen Komponenten beschrieben.

8.3 Eingebettete Systeme zur Bereitstellung der Prozessdaten

8.3.1 Fallbeispiel „Industrieller Kaffeeautomat“

Der industrielle Kaffeeautomat WMFcombiNationS ist ein Einzeltassenvollautomat für Espresso, Café Crème, Cappuccino, Milchkaffee, Latte macchiato und Filterkaffee. Außerdem ist eine Heißwasser- bzw. Dampfentnahme möglich. Es können sowohl Kaffeebohnen als auch gemahlener Kaffee verarbeitet werden, und über die Handzugabe ist die Zubereitung einer zusätzlichen Kaffeemehlsorte möglich. Die Bedienerführung der Maschine erfolgt über ein grafisches Touch-Screen-Display. Durch Druck auf bestimmte Felder auf dem Display können Funktionen der Maschine aktiviert bzw. Informationen abgerufen werden. Abbildung 8.5 zeigt die Frontseite des beschriebenen Kaffeeautomaten.

Abbildung 8.5: Frontseite des industriellen Kaffeeautomaten

Abbildung 8.6 gibt einen grundlegenden Einblick in den Aufbau des Kaffeeautomaten.

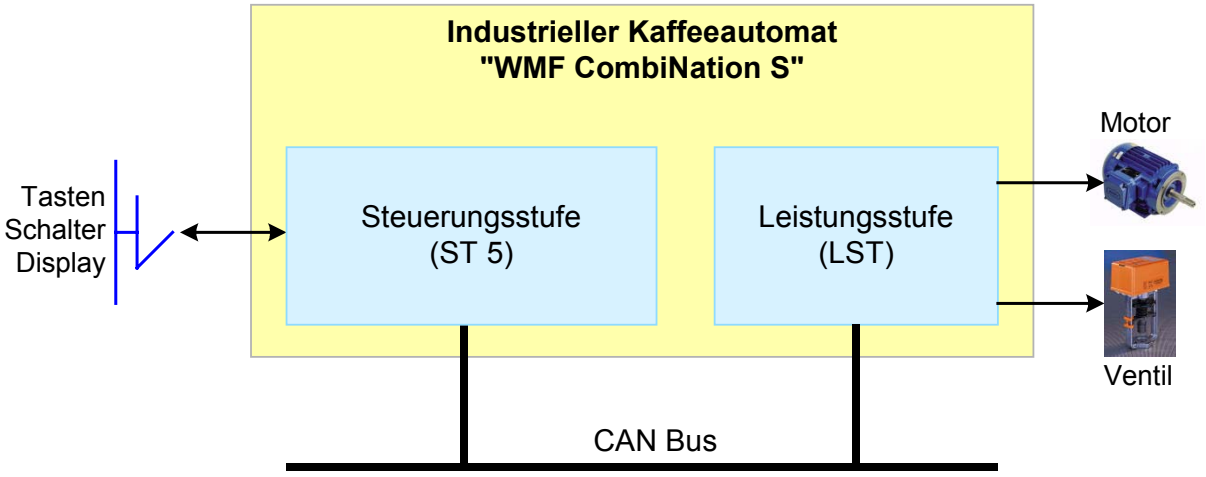

Abbildung 8.6: Struktureller Aufbau des Kaffeeautomaten

„Der Kaffeeautomat besteht aus zwei verschiedenen Komponenten und zwar aus einer 5V Steuerungsstufe (ST5) und aus einer 24V Leistungsstufe (LST). Die Steuerungsstufe enthält die eigentliche Steuerungssoftware und verwaltet die einzelnen Tasten, das Display und die Schalter der Anwendung. An diese Komponente sind alle Bedienungstasten des Kaffeeautomaten angeschlossen. Die Leistungsstufe verwaltet einzelne Aktoren der Hardware: dazu gehören in erster Linie Ventile und Motoren. Beide Komponenten kommunizieren über CAN-Bus (Controller

Area Network [Etsc00]) miteinander. Als Anwendungsprotokoll wird die so genannte CANopen-Spezifikation verwendet.

Das eigentliche Steuerungsprogramm des Kaffeeautomaten wird in der Steuerungsstufe ausgeführt. Dabei werden von der Steuerungsstufe aus einzelne Ventile über die Leistungsstufe aktiviert. Dies erfolgt über eine so genannte Master-Slave-Beziehung zwischen Steuerungsstufe und Leistungsstufe. Die Steuerungsstufe stellt den Master dar und überprüft in regelmäßigen Abständen den Zustand der einzelnen Aktoren über die Leistungsstufe. Die Leistungssufe ist der Slave und führt die Anweisungen des Masters aus.

8.3.1.1 Anbindung des Kaffeeautomaten an den Fernzugriff-Server

Ein Gesamtüberblick der Anbindung des industriellen Kaffeeautomaten an den Fernzugriff-Server ist in der Abbildung 8.7 dargestellt.

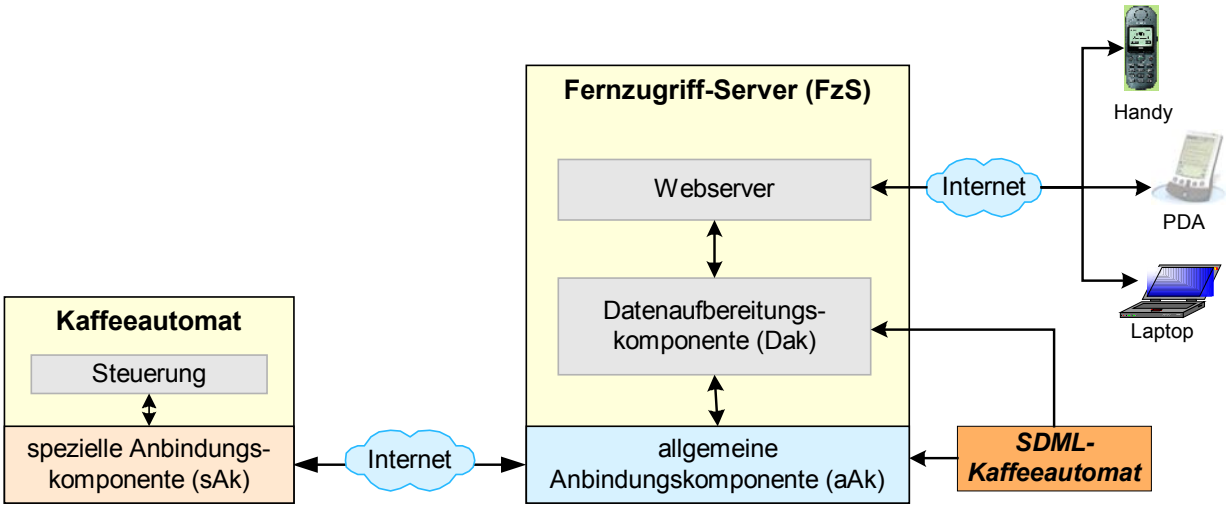

Abbildung 8.7: Anbindung des Kaffeeautomaten an den Fernzugriff-Server

Zur Anbindung des Kaffeeautomaten an das Internet wurde eine spezifische Anbindungskomponente implementiert, die einerseits Anfragen des Fernzugriff-Servers entgegennimmt und andererseits abgefragte Informationen dem FzS via Internet zur Verfügung stellt. Die Anbindung des Kaffeeautomaten an den Fernzugriff-Server und die Bereitstellung der Prozessdaten werden nachfolgend anhand eines einfachen Beispiels gezeigt. Zur Vereinfachung soll das eingebettete System durch einen so genannten Dienstserver, der nur einen einzigen Service zur Verfügung stellt, repräsentiert werden. Der Service liefert als Ergebnis das berühmte „Hello World“, das anschließend auf dem Client dargestellt wird. Abbildung 8.8 zeigt, wie dieser Dienstserver realisiert werden kann.

```
1:    public class HelloWorldDS
2:     {
3:       public HelloWorldDS()
```

```
4:        {}
5:
6:       public String HelloWorld()
7:        {
8:        return "Hello World";
9:        }
10:     public static void main( String[] args )
11:       {
12:        new HelloWorldDS();
13:       }
14:    }
```

Abbildung 8.8: Einfacher Dienstserver

Der Dienstserver besitzt genau eine Methode `HelloWorld`, die im Folgenden zum Service `HelloWorld` erweitert wird. Für den Fernzugriff auf diesen Service muss der Dienstserver durch die spezielle Anbindungskomponente erweitert werden. Hierfür wird eine Wrapper-Komponente speziell für diesen Dienstserver entwickelt, die für den richtigen Aufruf des Service sorgt. Das Ergebnis wird in der Weise aufbereitet, wie es die allgemeine Anbindungskomponente des Fernzugriff-Servers erwartet. In diesem Fall besteht die Wrapper-Komponente lediglich aus folgender Klasse (siehe Abbildung 8.9).

```
1:    // sAk imports
2:    import de.uni_stuttgart.ias.SCC.*;
3:    import java.util.Vector;

4:    public class HelloWorldDSWrapper implements SCCInterface
5:    {
6:    // Schnittstelle zum Dienstserver
7:    private HelloWorldDS dienstserver = null;
8:    public HelloWorldDSWrapper( HelloWorldDS dienstserver )
9:    {
10:   this.dienstserver = dienstserver;
11:   // Initialisierung und Starten des Servers
12:   SpecialConnection specialconnection =
13:   new SpecialConnection( this, false, "", 6326 );
14:   specialconnection.startServer();

15:   public Vector callService( String servicename, Vector inParameter )
16:   {
17:   if ( servicename.equals( "HelloWorld" ) )
18:   {
19:   return wrapHelloWorld( inParameter );
20:   }
21:   else
22:   {
```

```
23:   return null;
24:   }
25:   }
26:   public Vector wrapHelloWorld( Vector inParameter )
27:   {
28:   Vector result = new Vector();
29:   String helloworld = dienstserver.HelloWorld();
30:   Vector data = new Vector();
31:   data.addElement( helloworld );
32:   result.addElement( data );
33:   return result;
34:   }
35:   }
```

Abbildung 8.9: Wrapper-Komponente für den Dienstserver *HelloWorldDS*

Im Konstruktor erhält die Wrapper-Komponente eine Referenz auf den Dienstserver, um somit den eigentlichen Service aufrufen zu können. Außerdem wird im Konstruktor eine Instanz der Klasse `SpecialConnection` gebildet, die Teil der Fernzugriffskomponente der sAk ist. Da die Fernzugriffskomponente nicht angepasst werden muss, wurde sie in die Java-Bibliothek aufgenommen, die in dieser Arbeit entwickelt worden ist. So ist es möglich, die Fernzugriffskomponente durch lediglich zwei Zeilen Code zu instanzieren und zu starten. Für die Socketverbindung wird der Port auf 6326 festgelegt, über den eingehende Anfragen überwacht werden.

Damit die Fernzugriffskomponente auf die abstrakte Methode `callService` der Wrapper-Komponente zugreifen kann, muss die Methode implementiert und das Interface `SCCInterface` aus der Bibliothek eingebunden sein. Die Umsetzung des Serviceaufrufs `HelloWorld` und die Konvertierung des Ergebnisses erfolgt dabei in der Methode `wrapHelloWorld`. Im letzten Schritt muss der Dienstserver dahingehend erweitert werden, dass er die Wrapper-Komponente instanziert (siehe Abbildung 8.10).

```
1:    public HelloWorldDS()
2:    {
3:    new HelloWorldDSWrapper(this);
4:    }
```

Abbildung 8.10: Instanziierung der Wrapper-Komponente

Übrig bleibt noch die Erstellung eines SDML-Dokuments, in dem die Art der Verbindung, der Verbindungsaufbau und die Aufbereitung des Ergebnisses definiert sind.

Definition des Verbindungsaufbaus

In Abbildung 8.9 (Zeile 14) wurde der Port, auf den der Socket-Server der Fernzugriffskomponente der sAk hört, auf den Wert 6326 festgelegt. Zudem müssen dem Dienstserver noch die eindeutige ID `HelloWorldDS` und der Name `HelloWorldDS` zugeteilt werden. Der Name er-

scheint bei der Auswahl der Dienstserver auf der Startseite des Fernzugriff-Servers. Daraus ergibt sich folgende SDML-Definition (siehe Abbildung 8.11).

```
1:    <ServerDefinition id="HelloWorldDS" name="HelloWorldDS"
2:       host="localhost">
3:    <ConnectionType method="socket" port="6326"/>
4:    </ServerDefinition>
```

Abbildung 8.11: Definition des Verbindungsaufbaus

Definition des Service-Aufrufs

Ein Service ist definiert durch eine eindeutige ID und seinen Serviceaufruf (`Call`), damit die Wrapper-Komponente den eigentlichen Service aufruft. Des Weiteren wird der Text festgelegt, der auf dem Button zur Auswahl des Service, der beim Client angezeigt wird, erscheinen soll. Der Rückgabetyp des Services `HelloWorld` besteht aus einem einfachen Text und hat daher die Dimension 0. Die Dimension gibt an, ob der Service einen einzelnen Wert, eine Liste oder gar eine Matrix zurückliefert (siehe Abbildung 8.12).

```
1:    <Definition id="HelloWorld" call="HelloWorld" buttontext="Hello World">
2:    <ReturnType type="text" dimension="0"/>
3:    </Definition>
```

Abbildung 8.12: Definition des Serviceaufrufs

Definition der Ergebnisaufbereitung

Auf der Antwortseite des Serviceaufrufs kann ein Titel und eine kurze Beschreibung für den Benutzer angegeben werden. Für den Service HelloWorld wurde zudem noch festgelegt, dass das Ergebnis einem PC und einem PDA zur Verfügung gestellt wird und jeweils in Form eines einfachen Textes dargestellt wird (siehe Abbildung 8.13).

```
1:    <DataProcessing>
2:    <Title>Hello World</Title>
3:    <Description>Result of the very simple Dienstserver
4:      HelloWorldDS'.</Description>
5:    <ClientType type="pc" form="text"/>
6:    <ClientType type="pda" form="text"/>
7:    </DataProcessing>
```

Abbildung 8.13: Definition der Aufbereitung des Ergebnisses

8.3.2 Fallbeispiel „Aufzug“

Das Aufzugsmodell besitzt zwei Schächte, die gemeinsam vier Stockwerke bedienen. An jedem Einstiegspunkt sind für die möglichen Fahrtrichtungen Tasten angebracht, um den Aufzug anzu-

fordern. Diese Tasten können beleuchtet werden, um die Anforderung zu quittieren. Zusätzlich besitzt jedes Stockwerk für jeden Fahrkorb Signale, welche die aktuelle Fahrtrichtung des Fahrkorbs anzeigen. Im obersten und untersten Stockwerk stehen natürlich nur die Anforderungstasten und Kontrollsignale zur Verfügung, in deren Richtung eine Weiterfahrt möglich ist. Jede Einstiegsmöglichkeit in den Aufzug ist durch eine Tür gesichert, deren Zustand durch einen Sicherheitsschalter überwacht wird. Die Bedienelemente, die normalerweise in den Fahrkörben vorhanden sind, wurden zur Vereinfachung der Bedienung an der Frontseite des Aufzugsmodells angebracht. Dabei steht für jedes Stockwerk eine Anforderungstaste zur Verfügung, die den Fahrtwunsch durch Beleuchtung quittiert. Weiterhin ist ein Not-Aus-Schalter vorhanden, um den Fahrbetrieb im Notfall sofort anhalten zu können.

Abbildung 8.14 zeigt die Frontseite des beschriebenen Aufzugsmodells. Die aktuelle Position und Bewegung des Fahrkorbs wird mit Hilfe von vier Sensoren je Stockewerke erfasst, die innerhalb des Schachts angebracht sind. Jeder Fahrtrichtung sind zwei Sensoren zugeordnet – der erste ist für das rechtzeitige Abbremsen des Fahrkorbs zuständig, falls im betreffenden Stockwerk angehalten werden soll, und der zweite signalisiert die exakte Position, an welcher der Fahrkorb angehalten werden muss.

Abbildung 8.14: Frontseite des Aufzugsmodells

8.3.2.1 Anbindung des Aufzugsmodells an den Fernzugriff-Server

Abbildung 8.15 zeigt die Anbindung des Aufzugsmodells an den Fernzugriff-Server.

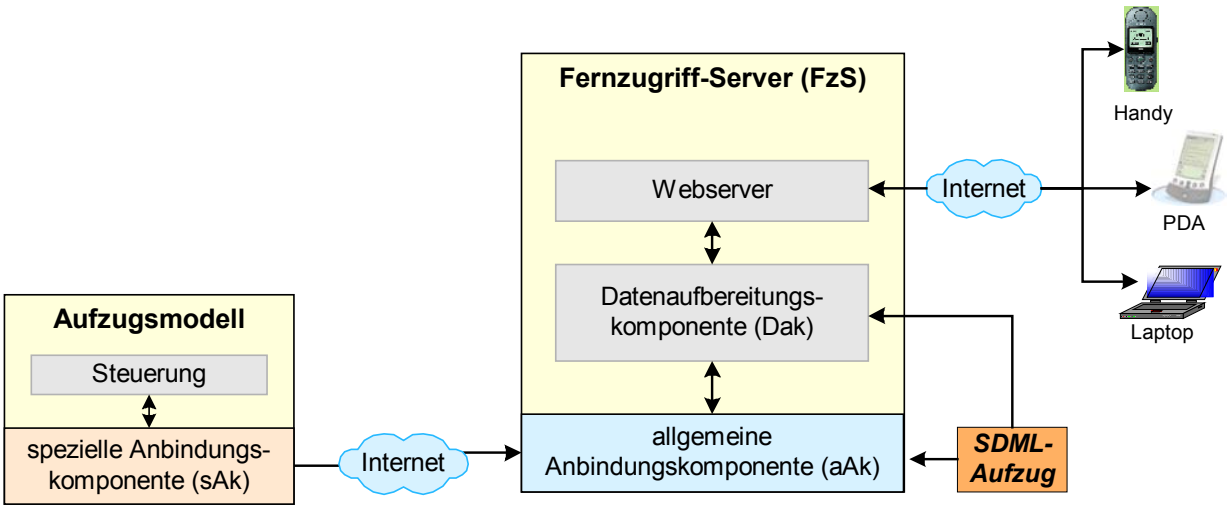

Abbildung 8.15: Anbindung des Aufzugsmodells an den Fernzugriff-Server

Zur Anbindung des Aufzugsmodells an den FzS wurde nach beschriebenem Prinzip in Abschnitt 8.3.1.1 eine spezifische Anbindungskomponente implementiert, die einerseits Anfragen des Fernzugriff-Servers entgegennimmt und andererseits abgefragte Informationen dem FzS über das Internet zur Verfügung stellt. Außerdem wurde eine SDML-Datei erstellt, die Informationen über die Art der Verbindung zum Aufzugsmodell (in diesem Fall eine Socketverbindung) und die aufzurufenden Prozessdaten beinhaltet. Der Zeitaufwand zur Erstellung spezieller Anbindungskomponenten und der SDML-Datei betrug ca. eine Mann-Woche. Hierbei nahm die Einarbeitung in die Steuerung des Aufzugsmodells einen nicht zu vernachlässigenden Anteil ein. Das bedeutet, dass der Implementierungsaufwand noch reduziert werden kann.

8.4 Abfrage und Präsentation der Prozessdaten

Der komplette Ablauf einer Session, beginnend mit der Anfrage eines Clients bis zur Präsentation der geforderten Prozessdaten, ist im Sequenzdiagramm der Abbildung 8.16 wiedergegeben. Eine Anfrageseite wird vom `ClientHandler` aufgenommen und an den `RequestProcessor` übergeben. Dieser ruft beim `DakController` den abstrakten Service `callService` mit den entsprechenden Parametern aus dem SDML-Dokument auf. Der `DakController` leitet den Service-Aufruf an die Komponenten `aAkController` der allgemeinen Anbindungskomponente weiter. Der Serviceaufruf wird übermittelt und der Rückgabewert an den `DakController` gegeben. Der `RequestProcessor` verarbeitet den Rückgabewert nach der Beschreibung im SDML-Dokument. Dabei werden nicht nur der Typ des anfragenden Clients berücksichtigt, sondern auch die Zugriffsrechte des Benutzers, der sich auf dem Gerät eingeloggt hat. Die ver-

arbeiteten Rückgabewerte werden anschließend vom `ClientHandler`, nachdem dieser vom `ScreenFlowManager` die nächste darzustellende Seite erfragt hat, in die Antwortseite eingebunden und mittels des Webservers an den Client gesandt.

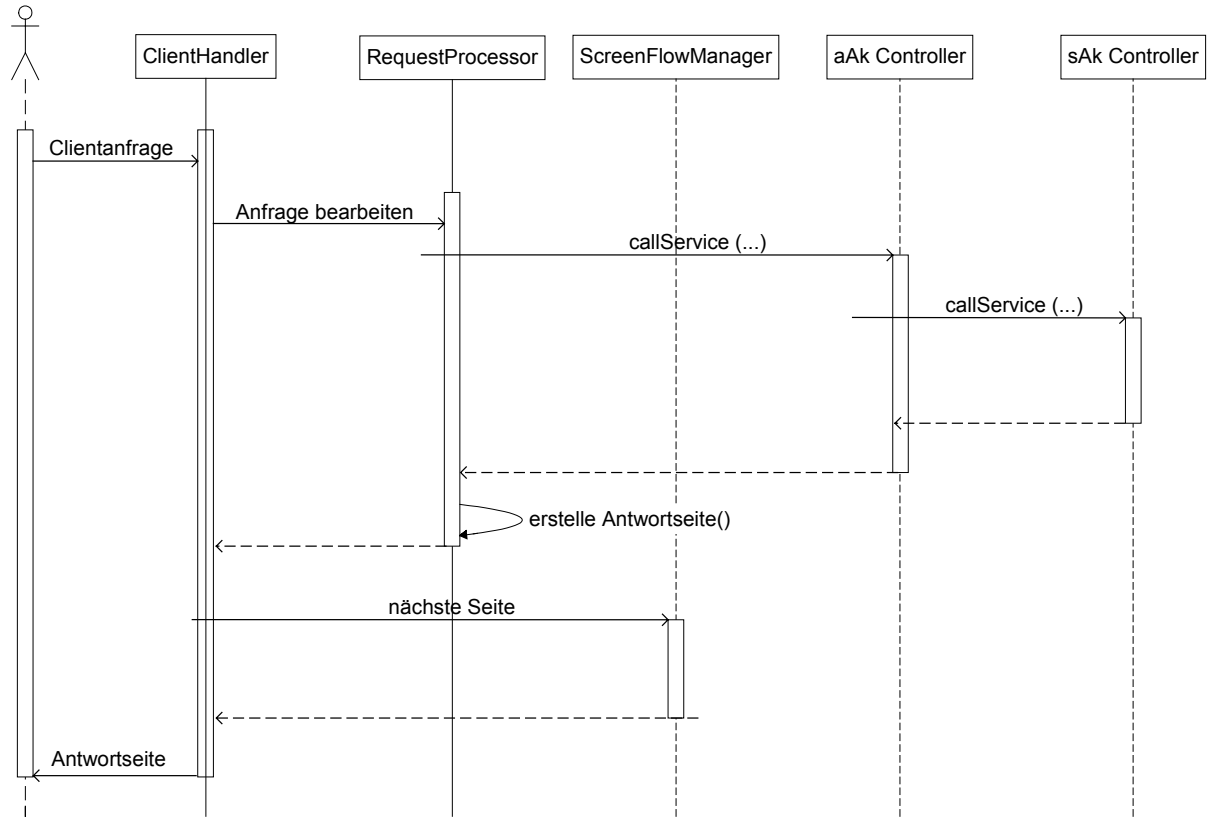

Abbildung 8.16: Sequenzdiagramm zur Bearbeitung einer Anfrage eines Clients

Wenn der Fernzugriff-Server gestartet ist, kann ein beliebiger Client mittels Browser eine Verbindung zu ihm herstellen. Auf der Startseite wird dem Benutzer nun eine Auswahl der für diesen Fernzugriff-Server zur Verfügung stehenden eingebetteten Systeme angeboten (siehe Abbildung 8.17).

Abbildung 8.17: Startseite des Fernzugriff-Servers

Nachdem der Benutzer ein Gerät ausgewählt und den Button *connect* gedrückt hat, erscheint auf der darauf folgenden Seite eine Auswahl der auf diesem Gerät zur Verfügung stehenden Services. Von diesen Services kann ein beliebiger ausgewählt werden. Jeder Service kann dabei weitere Services bereitstellen. Des Weiteren besteht auf jeder Seite die Möglichkeit, die Verbindung zum Gerät zu beenden bzw. zum Hauptmenü – der Auswahl der Services der ersten Ebene – zurückzukehren.

Wird zum Beispiel die WMF Combination ausgewählt, steht u. a. der Service *get process data* zur Verfügung. Dieser Service liefert die aktuellen Prozessdaten des Kaffeeautomaten. Abhängig vom Typ des Clients werden die gleichen Informationen unterschiedlich dargestellt (siehe Abbildung 8.18 und 8.19).

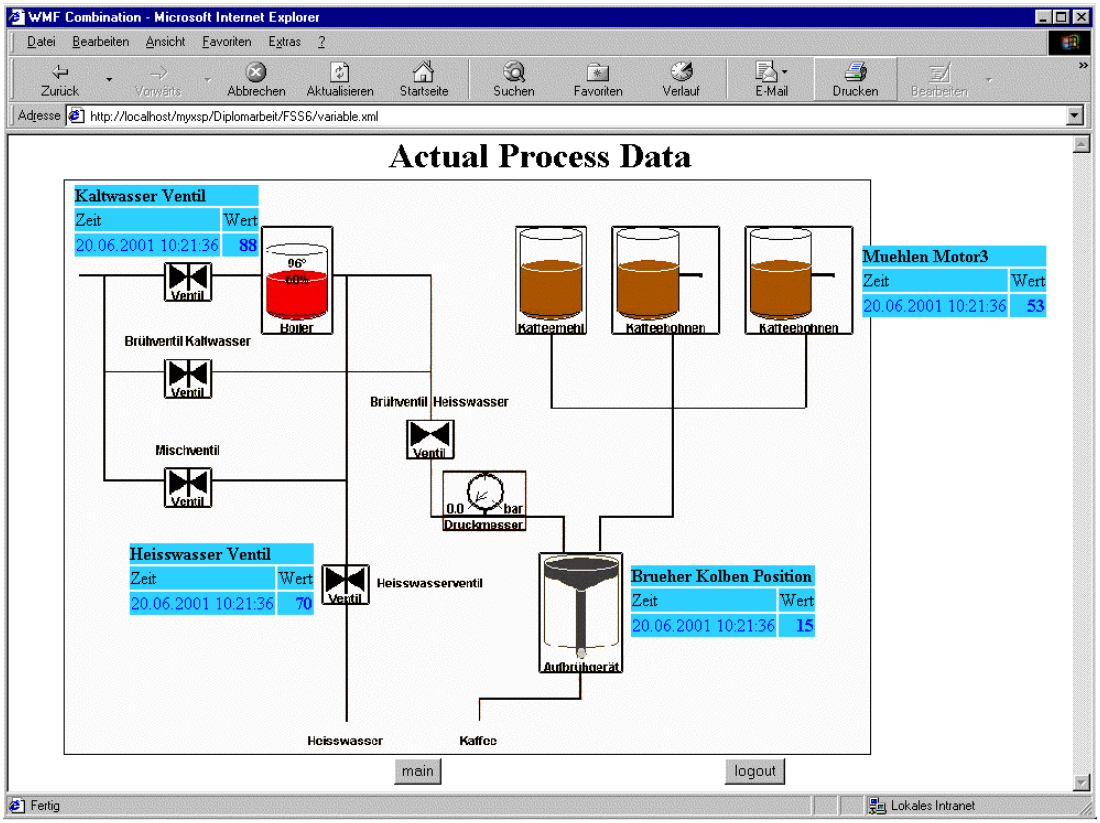

Abbildung 8.18: Aktuelle Prozessdaten des Kaffeeautomaten auf dem PC

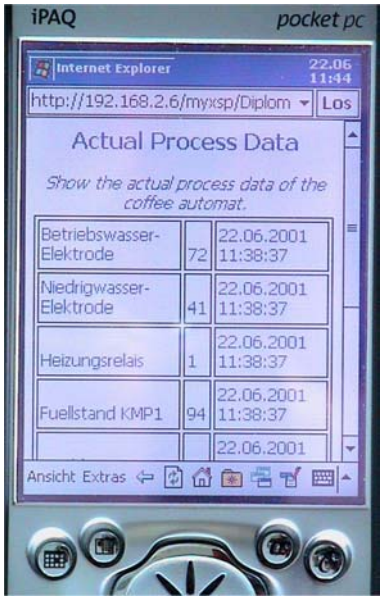

Abbildung 8.19: Aktuelle Prozessdaten des Kaffeeautomaten auf dem PDA

8.5 Ergebnisse und Erfahrungen

Nachdem der Prototyp, bestehend aus einem Fernzugriff-Server, zwei eingebetteten Systemen und drei Clients, und seine Funktionsweise beschrieben wurden, sollen in diesem Abschnitt die gesammelten Erfahrungen wiedergegeben werden.

Die Clients

Der wichtigste Punkt bei den Clients war, dass keine zusätzliche Software installiert werden muss. Durch den Einsatz eines gewöhnlichen Browsers sind sie sofort in der Lage, eine Verbindung mit dem Fernzugriff-Server herzustellen und dessen Dienste in Anspruch zu nehmen.

Der Fernzugriff-Server

Der Fernzugriff-Server wurde einmalig implementiert und ist in der Lage, beliebig viele eingebettete Systeme anzubinden und Lese- und Schreibaktionen durchzuführen. Um ein neues eingebettetes System an den Fernzugriff-Server anzubinden, muss lediglich eine Codezeile der Konfigurationsdatei hinzugefügt werden. Dies geschieht derzeit manuell, eine Unterstützung durch eine grafische Benutzungsoberfläche ist jedoch denkbar, damit sich der Anwender überhaupt nicht mehr mit Code auseinander setzen muss. Dazu kommt die Bereitstellung eines SDML-Dokuments, das dem Fernzugriff-Server Informationen über die Art der Verbindung zum Gerät und vorhandene Services gibt. Dieser Schritt wurde, wie in Kapitel 8 beschrieben, automatisiert. Vom gesamten Speicherplatzbedarf auf dem Fernzugriff-Server werden nur 8% für die allgemeine Anbindungskomponente, die Datenaufbereitungskomponente und alle Konfigurationsdateien verwendet. Die restlichen 92% werden für den Webserver benötigt. Beim Webserver gibt es einige Module wie Benutzungsanleitung und Dokumentation, die kein Bestandteil der Funktionalität der Software sind. Tabelle 8.1 gibt einen Überblick über die Speicherbelegung der einzelnen Module auf dem Fernzugriff-Server.

Tabelle 8.1: Speicherbelegung auf dem Fernzugriff-Server

Software	Speicherbedarf in MByte
Apache Webserver	**6,00**
• icons	0,084
• manual	3,210
• docs	0,065
• functions	2,100 (entspricht 35%)
Jakarta (Servlet Engine und Cocoon)	**20,400**
• functions	5,060 (entspricht 24 %)
• examples	0,380
• docs	14,800

Fernzugriff-Software	**2,316**
• Dak und aAk	2,300
• SDML-Dokumente	0,016

Wie aus Tabelle 8.1 ersichtlich, kann der Speicherbedarf beim *Apache Webserver* und beim *Jakarta* – 65% bzw. 76% – ohne weiteres optimiert werden, indem die Hilfedateien weggelassen werden.

Eingebettete Systeme

Für die Anbindung eingebetteter Systeme an den Fernzugriff-Server wurde pro Gerät eine spezielle Anbindungskomponente entwickelt. Das Gerät muss dabei zwei Funktionen zur Verfügung stellen: einerseits eine Zugriffsmöglichkeit auf seine Prozessdaten und andererseits einen Internet-Zugang. Abbildung 8.20 zeigt den Systemaufbau bei der Anbindung des Aufzugsmodells.

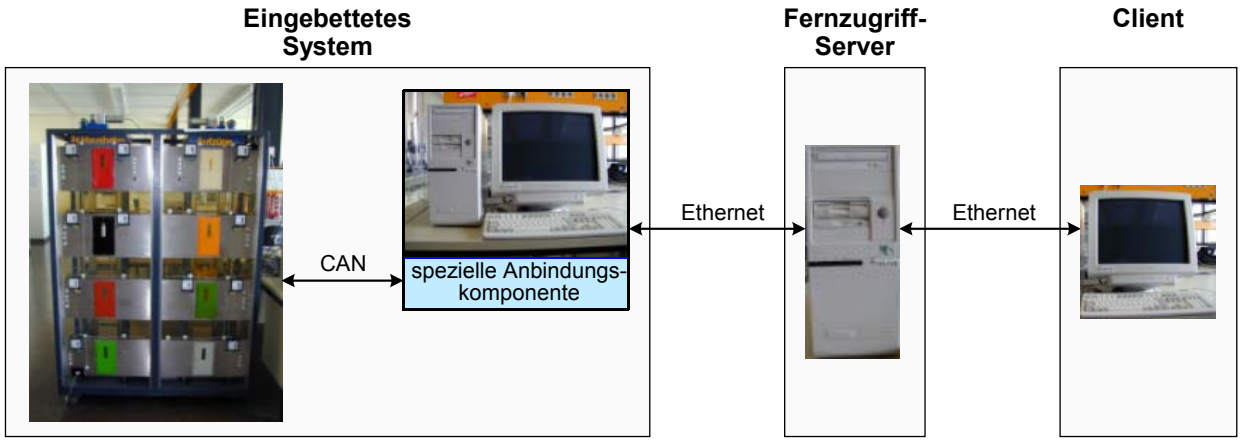

Abbildung 8.20: Systemaufbau bei der Anbindung des Aufzugsmodells

Die Hardware des Aufzugs wird über einen CAN-Feldbus mit einem PC verbunden. Für die Anbindung des PC an den CAN-Bus wird eine kommerzielle CAN-Einsteckkarte verwendet. Diese beinhaltet u. a. die benötigten Treiberbibliotheken. Auf der anderen Seite ist auf dem PC ein Webserver und eine Ethernet-Einsteckkarte installiert. Diese ermöglichen einen Internetzugang. Somit kann die spezielle Anbindungskomponente einerseits mittels Treiberbibliotheken der CAN-Einsteckkarte mit dem Aufzugsmodell und andererseits über den Webserver mit dem Fernzugriff-Server kommunizieren.

Beim industriellen Kaffeeautomaten wurde zunächst auf ähnliche Weise der Zugriff auf Prozessdaten und der Internet-Zugang realisiert. Abbildung 8.21 zeigt den Systemaufbau bei der Anbindung des Kaffeeautomaten.

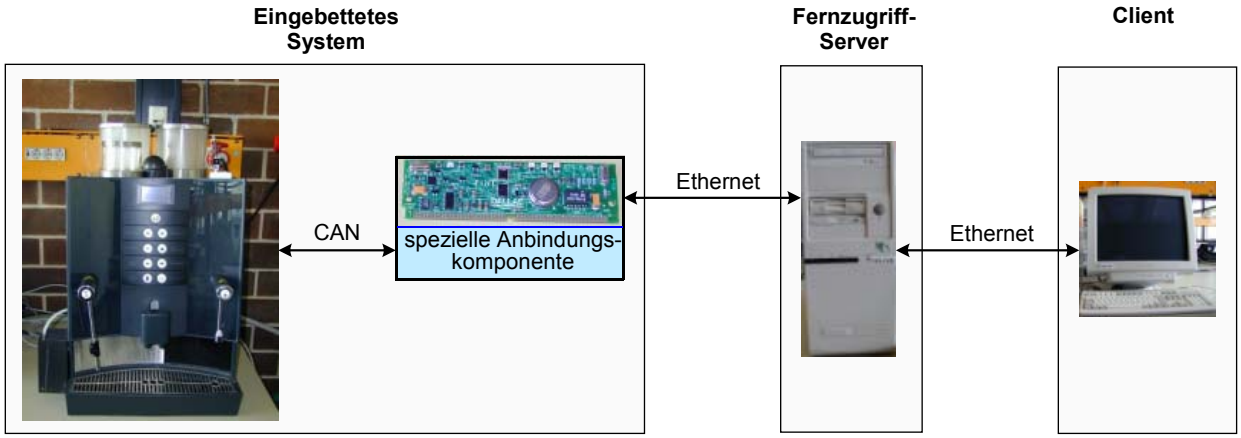

Abbildung 8.21: Systemaufbau bei der Anbindung des Kaffeeautomaten

In diesem Fall wurden die Kosten zur Anbindung des eingebetteten Systems an den Fernzugriff-Server reduziert. Hierzu wurde der PC durch ein TINI-Board ersetzt. TINI (Tiny Internet Interface) ist ein preisgünstiges (50 bis 70 €), kompaktes und Java-fähiges Mikrocontrollerboard, das unter anderem über eine Ethernet-Schnittstelle und einen integrierten CAN-Bus verfügt [Loo01]. Das TINI-Board ist eigentlich nur ein Referenzaufbau, um die Möglichkeiten des verwendeten Mikrocontrollers und TINI-Betriebssystems aufzuzeigen. Auf Grund seiner Architektur eignet sich das TINI-Board aber hervorragend, technische Prozesse um die Fähigkeit zu erweitern, Prozessdaten über lokale Netzwerke oder das Internet zu versenden und zu empfangen. Die Bauteile des TINI-Boards sind auf einer 72pin SIMM-Platine untergebracht. Abbildung 8.22 zeigt die Vorder- und Rückseite des TINI-Boards (Originallänge: 108 mm).

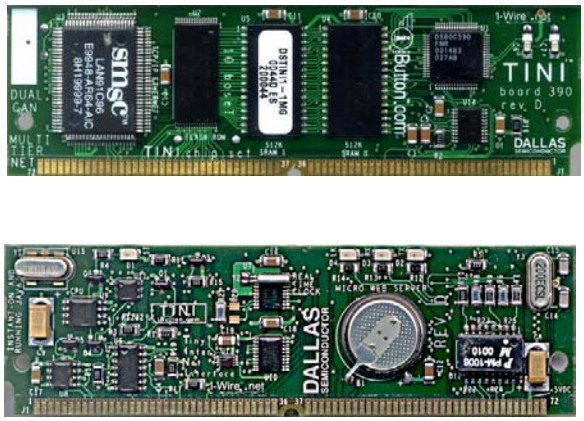

Abbildung 8.22: Vorder- und Rückseite des TINI-Boards

Da der Kaffeeautomat für die interne Kommunikation das CANopen-Protokoll verwendet, auf dem TINI-Board aber nur das Lowlevel-CAN-Protokoll implementiert ist, wurde ein CANopen-Interpreter auf dem TINI-Board implementiert. Die Speicherbelegung ist in Tabelle 8.2 wiedergegeben.

Tabelle 8.2: Speicherbelegung auf dem TINI-Board

Softwarekomponenten	**Speicherbelegung in kByte**
spezielle Anbindungskomponente	26,5
CANopen-API	32,0
Gesamte statische Speicherbelegung	**58,5**

Abbildung 8.23 stellt den Systemaufbau mittels TINI-Board grafisch dar.

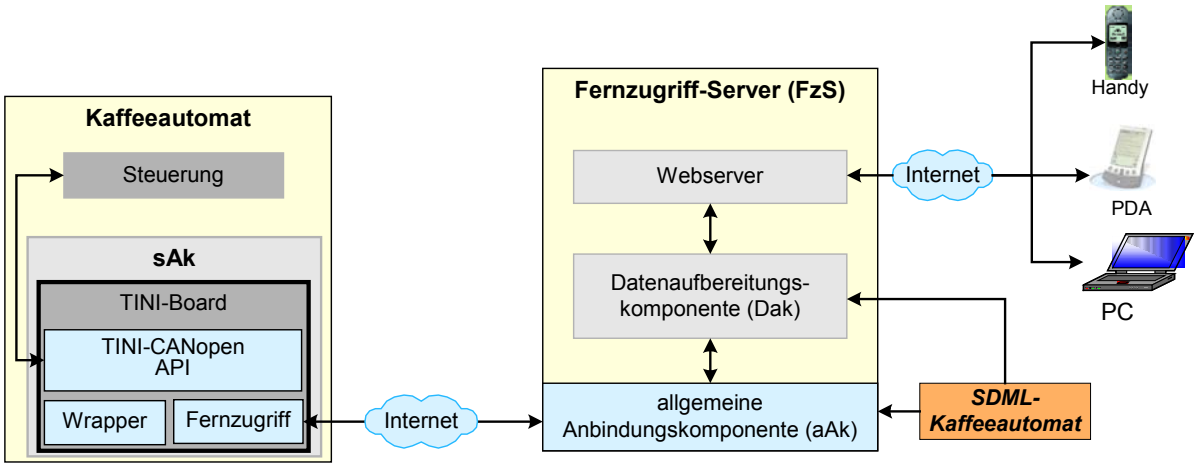

Abbildung 8.23: Systemaufbau bei der Anbindung des Kaffeeautomaten mittels TINI-Boards

Dies zeigt, dass das TINI-Board bei vollem Funktions-Umfang der Funktionalität der speziellen Anbindungskomponente den PC ersetzen kann. Auf Grund der wesentlich schwächeren Prozessorleistung von TINI im Vergleich zu einem konventionellen PC sind bei der Übertragung größerer Datenmengen allerdings zum Teil erhebliche Wartezeiten einzukalkulieren. Um eine genaue Aussage treffen zu können, wie diese Wartezeiten zustande kommen, wurden eine Reihe von Zeitmessungen einmal mit dem PC (für die Anbindung des Kaffeeautomaten) und einmal mit dem TINI-Board durchgeführt. Abbildung 8.24 zeigt die Auswertung von drei verschiedenen Zeitmessungen.

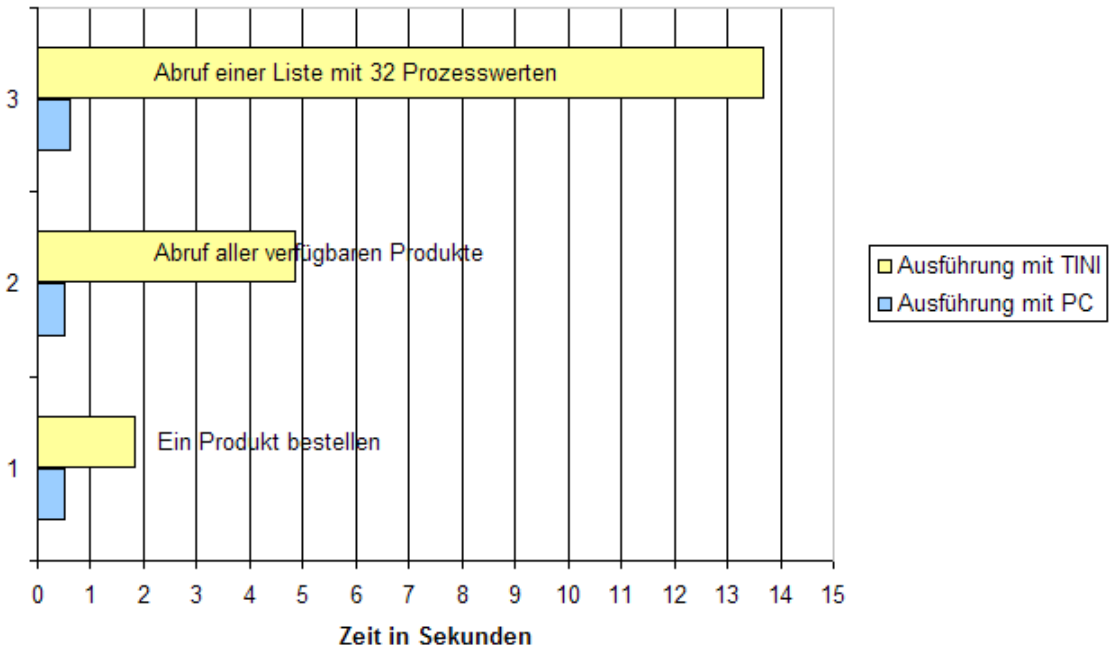

Abbildung 8.24: Auswertung von drei Zeitmessungen

Für die erste Messung (ein Produkt bestellen) wurde auf der Startseite (beim Client) die Funktion *Get Products* und anschließend *Tee* ausgewählt. Die Zeitspanne zwischen dem Betätigen des Knopfes und Einsetzen der Heißwasser-Ausgabe wurde gemessen. Nach Anzeige der Bestätigungsmeldung kehrt man mit der Schaltfläche *Main* auf die Hauptseite zurück und wiederholt die Messung insgesamt 5-mal. Nach gleichem Prinzip wurden die weiteren Messungen (Abruf aller verfügbaren Produkte und Abruf einer Liste mit 32 Prozesswerten) durchgeführt.

Da die Daten hauptsächlich in Form von Zeichenketten, also Strings, oder als Vector-Objekte versendet werden, wird ein höherer Rechenaufwand benötigt als bei einem atomaren Datentyp, z. B. Integer. Sie werden als Objekte mit Hilfe dynamischer Speicheranforderungen verwaltet. Da die Rechenleistung des TINI-Prozessors und der zur Verfügung stehende Speicher wesentlich geringer sind als bei einem PC, verbraucht die spezielle Anbindungskomponente auf dem TINI-Board mehr Zeit.

Fazit

Durch dieses Verfahren werden die rechenintensiven Applikationen auf den Fernzugriff-Server ausgelagert. Damit ist eine Minimierung der Hardware und Software, die für die Bereitstellung der Prozessdaten der Geräte notwendig sind, möglich. Die spezielle Anbindungskomponente soll von den Entwicklern eingebetteter Systeme erstellt werden. In diesem Fall fällt die Einarbeitungszeit in die Funktionsweise des jeweiligen Geräts weg. Damit kann der Aufwand von einer Mann-Woche gekürzt werden. Das Ziel soll die Reduzierung der Kosten des Geräts sein. Dies ist ein entscheidender Faktor für den Einsatz des Verfahrens im industriellen Umfeld.

In diesem Kapitel wurde am Beispiel einer kompletten Anwendung für die Steuerung und Übertragung der Prozessdaten eines industriellen Kaffeeautomaten bzw. eines Aufzugsmodells gezeigt, wie die Anbindung eines eingebetteten Systems an den Fernzugriff-Server funktioniert. Weiterhin wurde die Arbeit mit der Spezifikationssprache SDML in der Praxis erläutert. Es wurde gezeigt, dass durch eine einheitliche Schnittstelle zwischen eingebetteten Systemen und Fernzugriff-Server die Entwicklungskosten für die Realisierung von Fernservice-Applikationen wesentlich reduziert werden können. Im nächsten Kapitel werden abschließend die wesentlichen Merkmale des Verfahrens zur Realisierung einer universellen Fernservice-Infrastruktur für den Einsatz von Web-Technologien in eingebetteten Systemen zusammengefasst, bewertet und ein Ausblick auf mögliche Fortsetzung der Arbeit gegeben.

9 Schlussfolgerung und Ausblick

9.1 Bewertung des vorgestellten Verfahrens

In der vorliegenden Arbeit wurde ein Verfahren zur Realisierung einer universellen Fernservice-Infrastruktur für den Einsatz von Web-Technologien in eingebetteten Systemen erarbeitet. Das hier zugrunde liegende Konzept der Drei-Schichten-Architektur ermöglicht eine Auslagerung rechenintensiver Fernservice-Applikationen auf einen Fernzugriff-Server, der bei Bedarf auf mehrere Server verteilt werden kann. Die Realisierung einer allgemein gültigen Schnittstelle zwischen Fernzugriff-Server und eingebetteten Systemen ermöglicht eine einmalige Implementierung des Fernzugriff-Servers, an den beliebig viele eingebettete Systeme angebunden werden können. Durch den Einsatz von Standard-Technologien wie *Hypertext Transfer Protocol* (HTTP) zwischen Fernzugriff-Server und Clients ist der Anwender in der Lage, mit gängigen Browsern via Fernzugriff-Server auf Prozessdaten eines beliebigen Geräts zuzugreifen und diese auf dem Rechner grafisch zu betrachten, ohne dabei irgendeine Software installieren zu müssen. Durch den Einsatz von Browsern, mit denen wohl jeder Anwender vertraut sein dürfte, entfallen Schwierigkeiten, die neue Software in der Regel mit sich bringt, wie z. B. Installation, besondere Systemanforderungen und Einarbeitungszeit in die Bedienung. Das Konzept der Datenaufbereitung für unterschiedliche Typen von Clients wurde durch die entwickelten Datenaufbereitungskomponenten unter Verwendung der Beschreibungssprache XML und der damit verbundenen Technologien ermöglicht. Das vorgestellte Gesamtsystem eignet sich daher zur Darstellung verschiedenster Informationen für unterschiedliche Typen von Clients. Gerade die Möglichkeit, Daten für sehr unterschiedliche Plattformen bereitstellen zu können, gewährleistet absolute Ortsunabhängigkeit. Mit Hilfe der im Rahmen dieser Arbeit entwickelten Beschreibungssprache SDML ist eine Erweiterung des Fernzugriff-Servers um die Anbindung eines neuen eingebetteten Systems und die Beschreibung der Dienste des neuen Gerätes ohne großen Aufwand realisierbar. Außerdem ist auf Grund der Definition bestimmter Anwendergruppen eine anwenderspezifische Bereitstellung der Informationen möglich. Somit erhält jeder Anwender nur die für ihn vorgesehenen Daten, und die Informationspräsentation wird damit effektiver und übersichtlicher. Das Konzept unterstützt bei der Kommunikation zwischen eingebettetem System und Fernzugriff-Server neben *Socket*-Verbindungen, *Remote Method Innvocation* (RMI) und *Common Object Request Broker Architecture* (CORBA). Dieser Freiheitsgrad vergrößert die Palette der anzusprechenden Systeme erheblich.

Das Konzept bietet drei weitere Vorteile, die nachfolgend beschrieben werden.

Abstimmung auf Belange eingebetteter Systeme

Die Grundidee dieser Arbeit war die Auslagerung möglichst vieler Komponenten aus dem eingebetteten System, damit entstehende Kosten für die Webfähigkeit eines Gerätes möglichst gering gehalten werden können. Dieses Verfahren in Verbindung mit der speziellen Anbindungskomponente als einziger Komponente auf dem eingebetteten System erreicht dieses Ziel.

Herstellerunabhängigkeit

Das Konzept der speziellen Anbindungskomponente, welche die Funktion einer Adapterkomponente übernimmt, ermöglicht es, die Systemkomponenten unterschiedlicher Hersteller anzubinden und zu verknüpfen. Fernzugriff-Server und Fernservice-Applikationen sind nicht mehr an einen Hersteller bzw. eine Systemlinie gebunden. Dies bedeutet, dass beispielsweise eine einzige Diagnose-Applikation für unterschiedliche Geräte verschiedener Hersteller eingesetzt werden kann.

Keine Software- und Hardware-Änderungen des Gerätes

Das Verfahren fordert keine zusätzlichen Anforderungen von bestehenden eingebetteten Systemen, solange der Zugang zu den Prozessdaten, z. B. in Form eines Feldbusses, gegeben ist. Also kann das Gerät in seiner Konfiguration in gleicher Weise weiter betrieben werden wie bisher und die kompakte spezielle Anbindungskomponente wird in Form einer kleinen Blackbox an den Feldbus angekoppelt.

9.2 Grenzen des Verfahrens

Das vorgeschlagene Verfahren nutzt die Browser-Technologie zur Darstellung der Prozessdaten bei den Clients. Die Clients werden vom Fernzugriff-Server nicht aktiv über Änderungen am Prozessdatenbestand benachrichtigt. Zur Kommunikation zwischen Webserver des Fernzugriff-Servers und Webbrowser des Clients wird das HTTP-Protokoll eingesetzt, das in der jetzigen Form nur das Abrufen von Daten seitens der Clients erlaubt. Dies bedeutet, dass das Verfahren derzeit zur Überwachung eingebetteter Systeme noch nicht geeignet ist.

Durch die Einschränkungen der heutigen Netzwerk-Technologien zur Übertragung von Daten im Internet ist derzeit keine gesicherte Aussage zu den Übertragungszeiten möglich. Die Anforderung nach Echzeitverarbeitung kann somit nach dem heutigen Stand der Technik nicht erfüllt werden.

9.3 Ausblick auf weiterführende Arbeiten

Das im Rahmen der vorliegenden Arbeit vorgestellte Verfahren bietet noch Potenzial für eine Fortsetzung der Arbeit. Nachfolgend werden die wichtigsten Punkte erläutert.

- *Erweiterung der Kommunikation zwischen Fernzugriff-Server und Clients*: Wie bereits erwähnt, ist ein Fernzugriff-Server nicht in der Lage, aktiv Daten an Clients zu versenden. Ein nächster Schritt kann daher sein, das Verfahren zu erweitern, damit es auch für Überwachungsaufgaben eingesetzt werden kann. Die Clients könnten abhängig von der Dringlichkeit von Nachrichten zugeordnet werden. Somit könnten sie bei Eintreten bestimmter Alarme oder Ereignisse automatisch benachrichtigt werden.

- *Evaluierung und Erweiterung der SDML:* Durch weitere Evaluierung der SDML kann festgestellt werden, ob sie in einigen Punkten erweitert werden muss. Außerdem ist die Einführung von Logikelementen, die beim Fernzugriff-Server zur Abbildung der Abfolge der Service-Aufrufe ausgewertet werden können, ein weiterer notwendiger Schritt für die Weiterentwicklung der SDML.

- *Weiterentwicklung des SDML-Editors:* Der im Rahmen dieser Arbeit entwickelte Editor ermöglicht die Entwicklung von SDML-Dokumenten, ohne Code schreiben zu müssen. In der jetzigen Version fehlt jedoch eine Fehlerüberprüfung und Fehlerkorrektur während der Eingabe, mit der falschen Daten, wie z. B. fehlerhafte Port-Adresse etc., abgefangen werden könnten. Darüber hinaus könnte eine automatische Generierung von Meta-Daten erfolgen, wie das automatische Erstellen der Versionsnummer beim Anlegen einer neuen Version.

- *Verbesserung des TINI-Boards:* Der Einsatz des TINI-Boards zur Realisierung der speziellen Anbindungskomponente beim eingebetteten System gezeigte, dass eine kostengünstige Kopplung des Geräts an den Fernzugriff-Server auch mit den derzeitigen technischen Gegebenheiten möglich ist. Für eine Verbesserung der Laufzeit-Eigenschaften müsste die Datenübertragung zwischen TINI-Board und Fernzugriff-Server mit weniger Datenaufkommen verbunden sein.

- *Eigenentwicklung eines Web-Boards:* Die meisten Kosten bei der Anbindung eingebetteter Systeme ans Internet entstehen durch die Realisierung der Hardware und der dazugehörigen Software wie TCP/IP-Stack. Die Eigenentwicklung eines Web-Boards bietet die Möglichkeit zur Reduzierung der Kosten. Hierbei ist darauf zu achten, dass das zu entwickelnde Board möglichst modular und universell aufgebaut sein soll.

- *Standardisierung der SDML:* Dem Autor ist bis dato keine XML-basierte Beschreibungssprache in der Automatisierungstechnik bekannt. Aus diesem Grund ist es sehr wichtig, dass die Sprache weiter verbreitet und eine Standardisierung angestrebt wird.

Literatur

[Alda01] Khalid Al Dam: *Visualisierung der Prozessabläufe eines industriellen Kaffeeautomaten mit Java*, DA-1804, IAS, Stuttgart, 2001

[Balz99] Helmut Balzert: *Lehrbuch – Grundlagen der Informatik*, Spektrum, Akad. Verlag Heidelberg, Berlin, Oxford, 1999

[Balz99a] Heide Balzert: *Lehrbuch der Objektmodellierung: Analyse und Entwurf*, Spektrum, Akad. Verlag Heidelberg, Berlin, 1999

[BBHJ99] D. Barelmann, C. Bürger, St. Himstedt, R. Jamal: *OPC in der Praxis*, VDE Verlag, Berlin, 1999

[BeMi00] H. Behme, S. Mintert: *XML in der Praxis*, Addison Wesley, München, 2000

[BFM97] T. Berner-Lee, R. Fieldign, L. Masinter: *Uniform Resource Identifies (URI): Generic Syntax and Semantics,* RFC 1738, http://info.internet.isi.edu:80/in-notes/rfc/files, 1997

[BJS02] F. Bölstler, N. Jazdi, J. Seidelmann: *Konzept einer einheitlichen, herstellerunabhängigen Schnittstelle für den Zugriff auf Prozessdaten*, wt Werkstattstechnik online, Springer VDI Verlag, 2002

[Blac99] Uyless Black: *Internet-Technologien der Zukunft,* Addison-Wesley Longman Verlag, München, 1999

[Boge99] Marko Boger: *Java in verteilten Systemen: Nebenläufikeit, Verteilung, Persistenz,* dpunkt Verlag, Heidelberg, 1999

[Böls01] Frank Bölstler: *Modularer Aufbau eines PDA-basierten Konzeptes für den Zugriff auf Prozessdaten*, DA-1794, IAS, Stuttgart, 2001

[Boss00] Andreas Bossert: *Integration und Anpassung eines Webservers für den Einsatz in einem Kfz-Steuergrät*, DA-1760, IAS, Stuttgart, 2000

[BRA00] H. Buchner, G. Rauprich, W. Ahrens: *Was bringt XML in der Prozessleittechnik – Buzzword oder informationstechnischer Backbone für eCommerce, Planung und Betriebsbetreuung?,* atp, Heft 9, S. 51–62, Oldenbourg Verlag, 2000

[Bret00] McLaughlin Brett: *Java and XML*, Sebastopol, O'Reilly & Associates Inc., USA, 2000

[Bros92] Walter Brost: *Der Feldbus in der Maschinen- und Anlagentechnik*, Franzis Verlag, München, 1992

[Clar97] James Clark: *Comparison of SGML and XML,* http://www.w3.org/TR/NOTE-sgml-xml-971215, 1997

[Clar99] James Clark: *XSL Transformations (XSLT) version 1.0,* http://www.w3.org/TR/xslt, 1999

[Doss00] George M. Doss: *XML-Tipps*, Software & Support Verlag GmbH, Frankfurt a. M., 2000

[Down98] T. B. Downing: *Java RMI - Remote Method Invocation* John Wiley & Sons (C), USA, 1997

[Dude01] Duden, *Deutsches Universalwörterbuch*, Dudenverlag, Mannheim, Leipzig, Wien, Zürich, 1996

[Dujm01] Stjepan Dujmovic: *Anwendungsentwicklung mit Komponenten-Frameworks in der Automatisierungstechnik*, Dissertation am Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart, 2001

[Eber00] Stephan Eberle: *XML-basierte Internetanbindung technischer Prozesse,* Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart, 2000

[EbGö01] S. Eberle, P. Göhner: *Kostengünstige Internetanbindung eingebetteter Geräte mit verteilten Gerätehomepages,* Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart, 2001

[Ecks00] Robert Eckstein: *XML, kurz & gut*, O'Reilly Verlag, Köln, 2000

[Etsc00] Konrad Etschberger: *Controller Area Network: Grundlagen, Protokolle, Bausteine, Anwendungen*, Carl Hanser Verlag, 2. überarbeitete Ausgabe, München, Wien, 2000

[Fowl97] Martin Fowler *Analysis Patterns-Reusable Object Models*, Addison Wesley, Menlo Park, California, 1997

[Furr00] Frank J. Furrer: *Ethernet-TCP/IP für die Industrieautomation*, Hüthig-Verlag, Heidelberg, 2000

[Gieß01] Walter Gießler: *SIMATIC S7. SPS-Einsatzprojektierung und -Programmierung*, VDE Verlag, 2001

[GmAi00] GMD/AiS: *Autonomie und Adaptivität*, Tagung in Kloster Seeon im Februar/März 2000, http://ais.gmd.de/SEEON-2000/Abschluss-Christaller.pdf

[Goll99] Guido Gollmer: *Ansteuerung eines Modellprozesses per Internet in Java am Beispiel eines industriellen Kaffeeautomaten*, DA-1717, IAS, Stuttgart, 1999

[Gosp00] Robert Gospocic: *Fernbedienung und Diagnose eines industriellen Kaffeeautomaten über das Internet*, DA-1756, IAS, Stuttgart, 2000

[Gros02] William Grosso: *Java RMI*, O'Reilly & Associates Inc., Sebastopol, USA 2002

[Gutb02] Felix Gutbrodt: *Entwicklung eines Editors für SDML-Dokumente*, SA-1826, IAS, Stuttgart, 2002

[Gutb03] Felix Gutbrodt: *Sicherheit in der Automatisierungstechnik*, Forschungsarbeit am IAS, http://www.ias.uni-stuttgart.de/de/start_mitarbeit.html, 2003

[GVNG94] D.D. Gajski, F. Vahid, S. Narayan, J. Gong: *Specification and Design of Embedded System*, PTR Prentice Hall, USA, 1994

[GWBJ01] P. Göhner, Th. Wagner, F. Bölstler, N. Jazdi*: Einsatz von XML zur einheitlichen Übertragung von Prozessdaten*, atp-Automatisierungstechnische Praxis, 2001

[Haas01] Oliver Haase: *Kommunikation in verteilten Anwendungen,* Oldenbourg Verlag, München, Wien, 2001

[Harm98] Paul Harmon: *Components*, Component Development Strategies, Volume VIII No. 7, Cutter Consortium, USA, 1998

[Haus00] S. Haustein, M. Kroll: *Java-Entwicklung mit CLDC auf Kleinstgeräten*, Java Spektrum, S. 24–29, November/Dezember 2000

[Heat99] Steve Heath: *Embedded Sytems Design*, 3. Aufl., Oxford, Auckland, Boston, Johannesberg, Melborne, New Dehli, USA, 1999

[Holz01] Steven Holzner: *Inside XML*, New Riders Publishing, Indianapolis, USA, 2001

[HWR99] A. Hergenhan, Ch. Weiler, W. Rosenstiel, K. Weiß: *Internet-basierte eingebettete Systeme in der industriellen Automation*, At 7/99, Automatisierungstechnik 47, Theoretische Grundlagen, Methoden, Anwendungen, S. 305–313, Oldenbourg Verlag 1999

[Inter00] *Interbus-Club-Online*: http://www.interbusclub.com, 2000

[IwLa01] F. Iwanitz, J. Lange: *OLE for Process Control*, Hüthig Verlag, Heidelberg, 2001

[Jazd99] Nasser Jazdi: *Ansteuerung eines industriellen Kaffeeautomaten*, VDE-GMA Fachtagung Langen, VDI Bericht 1515: Industrielle Automation und Internet/Intranet-Technolog, 1999

[Jazd00] N. Jazdi, J. Konnertz, J. Traumüller: *Einsatz von Web-Technologien bei der Entwicklung moderner eingebetteter Systeme,* 12th International Workshop on Distributed and Parallel Embedded Systems (DIPES 2000), Illmenau, 2000

[Jazd01] Nasser Jazdi: *Component-based and Distributed Web Application for Embedded Systems*, International Conference on Intelligent Agents, Web Technology and Internet Commerce–IAWTIC, Las Vegas, USA, 2001

[Jazd02] Nasser Jazdi: *Distributed Web Applications for Embedded Systems*, International Conference „Software & Systems Engineering and their Applications“, CNAM, Paris, France, 2002

[Jeck00] Mario Jeckle: *Entwurf von XML Sprachen*, Java Spektrum, Heft 6, S. 56–60, 2000

[Kope97] Hermann Kopetz: *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Boston, Dordrecht, London, 1997

[Krau01] Andreas Kraut*: Warum die Automatisierungstechnik die IAONA braucht*, Jetter Journal, April 2001

[Krau01a] Holger Krauth: *Integration und Anpassung eines Embedded-PCs für die Ferndiagnose eines industriellen Kaffeeautomaten*, DA-1779, IAS, Stuttgart, 2001

[KrPo88] G. Krasner, S. Pope: *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smaltalk-80*, Journal of Object Oriented Programming (JOOP), pp. 26-49, August/September 1988

[Kühn01] Paul J. Kühn: *Skript zur Vorlesung Communication Networks I*, Institut für Kommunikationsnetze und Rechnersysteme, Stuttgart, 2001

[Kunz00] Christoph Kunz: *Steuerung und Visualisierung eines industriellen Kaffeeautomaten mit JavaBeans*, DA-1741, IAS, Stuttgart, 2000

[LaGö99a] Rudolf Lauber, Peter Göhner: *Prozessautomatisierung I*, 3. vollst. überarb. Auflage, Springer Verlag, Berlin-Heidelberg-New York, 1999

[LaGö99b] Rudolf Lauber, Peter Göhner: *Prozessautomatisierung II*, 1. Auflage, Springer Verlag, Berlin, Heidelberg, New York, 1999

[LiFä00] Bonito Liccardi, Georg Färber: *Template-basierte Entwicklung von eingebetteten Systemen für Produkte mittlerer Seriengröße*, atp-Automatisierungstechnische Praxis, Seiten 26-36, Juni 2000

[Loom01] Don Loomis: *The TINI Specification and Developer's Guide*, Addison-Wesley, New York, USA, 2001

[Megg98] David Megginson: *Structuring XML-Documents,* Upper Saddle River, Prentice-Hall, NJ, USA, 1998

[Midd00] Stefan Middendorf: *Von Bohnen und Bäumen: SAX2, JAXP, JDOM und mehr*, iX, 12/2000, S. 112–118

[MOV96] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone: *Handbook of Applied Cryptography,* CRC Press, 1996

[Müll02] Jürgen Müller: *Regeln mit SIMATIC. Praxisbuch für Regelungen mit SIMATIC S7 und SIMATIC PCS7*, MCD Verlag, 2002

[MüFä00] Alexander Münnich, Georg Färber: *Calculating Worst-Case Execution Times of Transactions in Databases for Event-Driven, Hard-Real-Time Embedded Systems*, Proceedings of the IEEE: International Database Engineering and Applications Symposium (IDEAS'00), Yokohama, Japan, September 2000

[Naed03] Martin Naedele: *IT Security for Automation Systems*, atp-Automatisierungstechnische Praxis, Ausgabe 5/2003, S 84-91

[NaMy00] A. Nakhimovsky, T. Myers: *Java XML Programming*, MITP-Verlag, Bonn, 2000

[Nato00] Ali Natour: *Erstellung eines OPC-Clients zum Datenaustausch zwischen verschiedenen Automatisierungsapplikationen*, DA-1748, IAS, Stuttgart, 2000

[OrHa97] R. Orfali, D. Harkey: *Client/Server Programming with Java and Corba*, John Wiley & Sons, USA, 1997

[Pösc01] Thomas Pöschmann: *MVC im Shop*, Java Magazin, 02/2001, S. 37–42

[Prof02] PROFIBUS International: *PROFInet–Technical Description,* http://www.profibus.com, Karlsruhe, 2002

[RaRa98] D. S. Ray, E. J. Ray: *Web-Publishing mit HTML 4*, Sybex Verlag, Düsseldorf, 1998

[Rock02] Rockwell Automation: *Ethernet TCP/IP als Steuerungsnetzwerk?,* http://www.ab.com/catalogs/b113/comm/ethernet.html*,* 2002

[Rose99] Wolfgang Rosenstiel: *Entwurfsmethoden für eingebettete Systeme*, it+ti – Informationstechnik und Technische Informatik, 2/99, S. 5-6

[Schn00] B. Schneier: *Secrets and Lies – Digital Security in a Networked World*, Wiley, 2000

[Schn94] Gerhard Schnell: *Bussysteme in der Automatisierungstechnik,* Vieweg & Sohn, Braunschweig, Wiesbaden, 1994

[Schn99] Eckehard Schnieder: *Methoden der Automatisierung*, Vieweg & Sohn, Braunschweig, Wiesbaden, 1999

[Schu00] Kay Schulz: *Java professionell programmieren*, Springer Verlag, Berlin, Heidelberg, 2000

[Spen99] Paul Spencer: *Professional XML Design and Implementation*, Wrox, USA, 1999

[Star00] Gernot Starke: *Java auf dem Palm-Pilot*, Java Spektrum, Mai/Juni 2000, S. 18–25

[Stla98] Simon St. Laurent: *XML A Primer*, MIS Press, Forster City/Ca, USA, 1987

[VaGi01] Frank Vahid, Tony D. Givargis: *Embedded System Design: A Unified Hardware/Software Introduction*, John Wiley & Sons, USA, 2001

[Vigd98] Mark Vigder: *An Architecture for COTS Based Software Systems*, National Research Council of Canada, 1998

[Wagn01] Thomas Wagner: *Konzeption für die Übertragung von Prozessdaten bei Fernservice-Applikationen*, DA-1782, IAS, Stuttgart, 2001

[Wede02] Michael Wedel: *Untersuchung und Vergleich der Leistungsfähigkeiten der Übertragungstechnologien RMI und XML*, SA-1820, IAS, Stuttgart, 2002

[Willi00] John C. Williams: *Harbor Research, Whitepaper*, http://www.harborresearch.com, San Francisco, USA, 2002

[Zelt01] Holger Zeltwanger: *CANopen*, VDE Verlag, 2001

Lebenslauf

Persönliche Daten:

20.08.1960	geboren in Mashhad / Iran

Schulbildung:

1966 - 1971	Grundschule in Mashhad / Iran
1971 - 1977	Gymnasium in Mashhad / Iran, Abschluss Abitur

Studium:

1977 - 1980	Studium der Medizin an der Universität Teheran / Iran, kein Abschluss
1987 - 1989	Studienkolleg an der Universität Karlsruhe, Zulassung zum Studium in Deutschland
1989 - 1997	Studium der Elektrotechnik an der Universität Stuttgart, Studienrichtung: Theoretische Nachrichtentechnik
30.06.1997	Abschluss als Diplom-Ingenieur

Berufstätigkeit:

Seit 01.07.1997	Wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart