Agentenbasiertes Monitoring von Netzressourcen auf Basis von Webservices

Florian Müller

15. November 2005

Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund

Diplomarbeit

Agentenbasiertes Monitoring von Netzressourcen auf Basis von Webservices

von
cand. inform. Florian Müller

Aufgabenstellung und Betreuung:
Prof. Dr. Stefan Fischer und Dipl.-Wirt.-Inf. Torsten Klie

Braunschweig, den 15. November 2005

Erklärung

Ich versichere, die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Braunschweig, den 15. November 2005

Kurzfassung

Durch ein anhaltend steigendes Datenaufkommen haben sich Kommunikationsnetze weit verbreitet. Für die Zuverlässigkeit und Qualitätssicherung müssen diese Netze kontrolliert werden. Dies erfolgt über ein Monitoring der beteiligten Ressourcen und wird gegenwärtig computergestützt über das Simple Network Management Protocol (SNMP) durchgeführt. Da dieses Protokoll im Bereich des Ressourcenmonitorings den heutigen Anforderungen nicht mehr gewachsen ist, liegt die Idee in der Entwicklung eines neuen Monitoringsystems. Weil das neue System in heterogenen Umgebungen arbeiten soll, setzt es auf eine moderne interoperable Kommunikationstechnologie, den Webservices. In dem neuen Monitoringsystem ist jeder Ressource eine eigene Monitoringkomponente zugeordnet, welche das Monitoring übernimmt. Diese Komponente nimmt externe Befehle entgegen, die das Monitoringverhalten konfigurieren, und verschickt Benachrichtigungen über Monitoringereignisse.

Abstract

Due to an increasing amount of data, communication networks have spread. For reliability and quality assurance these networks have to be controlled. This is made by monitoring involved resources and is currently realized computer based with the Simple Network Management Protocol (SNMP). Since this protocol does not meet today's requirements for the part of resource monitoring, the idea is to develop a new monitoring system. This new system uses a modern interoperable communication technology based on web services, as it shall work in heterogeneous environments. In the new monitoring system one private monitoring component, which takes over control of the monitoring process, is assigned to each resource. This component accepts external instructions to configure the monitoring performance and sends messages about monitoring events.

[Hier wird später die Aufgabenstellung eingefügt.]

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
ATM	Asynchronous Transfer Mode
BEEP	Blocks Extensible Exchange Protocol
BER	Basic Encoding Rules
CMIP	Common Management Information Protocol
CMOT	CMIP over TCP
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DOM	Document Object Model
DTD	Document Type Definition
EGP	Exterior Gateway Protocol
FTP	File Transfer Protocol
GUID	Global Unique Identifier
HEMS	High Level Entity Management System
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force

IP	Internet Protocol
ISO	International Organization for Standardization
J2SE	Java 2 Standard Edition
JAR	Java-Archiv
JSP	Java Server Pages
JVM	Java Virtual Machine
MIB	Management Information Base
OID	Objekt-ID
PDU	Protocol Data Unit
RFC	Request for Comments
RPC	Remote Procedure Call
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SGMP	Simple Gateway Monitoring Protocol
SMI	Structure of Management Information
SMIv1	Structure of Management Information Version 1
SMIv2	Structure of Management Information Version 2
SMP	Simple Management Protocol
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SNMPv1	Simple Network Management Protocol Version 1
SNMPv2	Simple Network Management Protocol Version 2
SNMPv3	Simple Network Management Protocol Version 3
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol[1]

[1] Ab der Version 1.2 ist SOAP keine Abkürzung mehr, sondern ein eigenständiger Name.

SSH	Secure Shell
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WWW	World Wide Web
XML	Extensible Markup Language
XSD	XML Schema Definition

1 Einleitung

In den letzten Jahren ist der elektronische Datenaustausch in Unternehmen und auch im privaten Bereich immer mehr zu einem wichtigen Faktor geworden. Ein stetig steigendes Datenaufkommen hat zu einer weiten Verbreitung von Kommunikationsnetzen geführt. Dieses enorme Wachstum hat zur Folge, daß es zunehmend schwieriger wird, die Zuverlässigkeit von Kommunikationsnetzen zu kontrollieren. Jedoch ist die Kontrolle und Überschaubarkeit für den effizienten und gesicherten Betrieb großer und komplexer Netzwerke unabdingbar. Während beispielsweise die Kontrolle kleiner Computernetzwerke (u. a. Datendurchsatz, Dienstgüte, Verfügbarkeit, Zuverlässigkeit) noch von einzelnen Personen umgesetzt werden kann, ist dies in großen Netzwerken so nicht mehr möglich.

In Netzwerken mit mehreren hundert oder tausend Komponenten sind Werkzeuge erforderlich, um die o. g. Anforderungen zu erfüllen. Diese Werkzeuge bezeichnet man im allgemeinen als Netzwerkmanagementwerkzeuge. Da in großen Kommunikationsnetzen viele verschiedene heterogene Systeme zusammenarbeiten, müssen ebenfalls die Werkzeuge des Netzwerkmanagements interoperabel sein. Mit Hilfe von werkzeuggestütztem Netzwerkmanagement ist es erreichbar, weiterhin die geforderte Kontrolle und Übersichtlichkeit in komplexen Netzwerken zu behalten.

Seit vielen Jahren wird für das Netzwerkmanagement das *Simple Network Management Protocol (SNMP)* eingesetzt. Dieses Protokoll hat sich bewährt und im Laufe der Zeit zahlreiche Überarbeitungen erfahren. Jedoch hat es vom Konzept her für heutige Anwendungen im Bereich des Netzwerkmanagements einige Schwächen. Daher gibt es das Bestreben, eine komplette Neuentwicklung eines Netzwerkmanagementkonzepts voranzutreiben.

Diese Arbeit beschäftigt sich daher mit der Neuentwicklung eines Systems für das Monitoring von Netzwerkressourcen. Das neue System setzt dabei auf moderne Kommunikationstechnologien auf und wickelt seine gesamte Kommunikation über Webservices ab.

Das nachfolgende 2. Kapitel gibt zuerst einen kurzen Überblick über die *Entwicklung des Netzwerkmanagements* und behandelt anschließend, als Grundlage dieser Diplomarbeit, das *SNMP*. Nach der Beschreibung von SNMP folgt ein kleiner Exkurs zu einem weiteren Netzwerkmanagementansatz.

Kapitel 3 beschreibt zunächst eine Einführung in das Thema *verteilte Systeme*. Danach folgt eine genaue Abhandlung von *Webservices* und der damit verbundenen Technologien.

Kapitel 4 handelt über den *Entwurf eines agentenbasierten Monitoringsystems* und beschreibt Ansätze, wie ein solches Monitoringsystem aufgebaut werden könnte. Weiter zeigt dieses Kapitel einen Vergleich zwischen einer bestehenden Monitoringtechnologie und dem agentenbasierten Ansatz mit Webservices.

Das folgende Kapitel 5 definiert, aufbauend auf dem Entwurf des vorherigen Kapitels, eine genaue Spezifikation eines Monitoringsystems.

Das 6. Kapitel stellt eine Prototypimplementierung einer Manager- und Agentenanwendung vor. Diese verwenden die in dem vorherigen Kapitel definierte Spezifikation.

Kapitel 7 zeigt, welche Testmethoden während oder nach der Implementierung des Prototyps angewendet wurden, um das korrekte Verhalten der Implementierung zu belegen.

Kapitel 8 bewertet das entwickelte Monitoringsystem, und Kapitel 9 enthält eine Zusammenfassung dieser Arbeit und gibt einen Ausblick auf zukünftige Erweiterungen.

Hinweis: Das Symbol » ↵ « in abgedruckten Programmcodeausschnitten oder Kommandozeilen kennzeichnet einen nicht vorhandenen Zeilenumbruch. So gekennzeichnete Zeilen sind nur aufgrund des Dokumentenlayouts umgebrochen und müssen in einer Zeile stehen.

2 Netzwerkmanagement

Dieses Kapitel gibt zuerst einen kurzen Überblick über das Netzwerkmanagement und dessen Entstehungsgeschichte und behandelt dann ausführlich *SNMP* und *Benachrichtigung durch Agenten in SNMP*. Die Benachrichtigung durch Agenten bildet die Grundlage für die Untersuchung von Webservices als Eignung für ein agentenbasiertes Monitoring in Kapitel 4 dieser Diplomarbeit. Weiterhin erläutert dieses Kapitel die Zusammenhänge der Einzelteile des SNMP-Rahmenwerks und die Kommunikationsbeziehungen zwischen Manager und Agenten in einem managebaren Netzwerk. Im Anschluß von SNMP gibt es eine kleine Einführung in weitere Netzwerkmanagementansätze.

2.1 Was ist Netzwerkmanagement

Der Begriff *Netzwerkmanagement* ist nicht eindeutig definiert, aber allgemein läßt sich sagen, daß dies ein Oberbegriff für die Steuerung und Überwachung von Netzwerken darstellt. Die International Organization for Standardization (ISO) hat das Netzwerkmanagement für ihre eigenen Entwicklungen in fünf Kategorien unterteilt [ISO89]. Diese Unterteilung wurde auch von anderen Herstellern im großen und ganzen akzeptiert und ist in Tabelle 2.1 beschrieben. Diese Aufgabenbereiche arbeiten nicht unabhängig voneinander, sondern interagieren auch miteinander. Beispielsweise erfordert ein erkannter Ausfall einer Netzkomponente im *Fehlermanagement* gegebenenfalls eine Umkonfiguration einer anderen Komponente des Netzwerks durch das *Konfigurationsmanagement* ([Sta96], [Sch99]).

2.2 Historie

Als sich das Internet, vormals ein amerikanisches Forschungsnetz für Universitäten und andere Forschungseinrichtungen (ARPANET), im Laufe der 80er Jahre sprunghaft ausbreitete und sukzessive immer mehr heterogene Komponenten daran beteiligt waren, wurden Lösungen gesucht, um das Management in dem Verbund von Netzwerken zu vereinfachen. Demzufolge arbeiteten mehrere Gruppen von Entwicklern unterschiedliche Ansätze zur Vereinfachung des Netzwerkmanagements aus. Dies waren u. a. *HEMS (High Level Entity Management System)* [PT87], *CMOT (CMIP over TCP)* [WB89] und *SGMP (Simple Gateway Monitoring Protocol)* [DCFS87]. Nach einer ausführlichen Diskussion über die Vor- und Nachteile der einzelnen Managementkonzepte deutete sich an, daß sich HEMS wahrscheinlich nicht durchsetzen wird. Die SGMP-Spezifikation entwickelte sich weiter fort und firmiert kurze Zeit später

Kategorie	Beschreibung
Fehlermanagement	Fehlererkennung, Fehlerisolation und Fehlerkorrektur in Netzwerkkomponenten
Abrechnungsmanagement	Erzeugen von Kosteninformationen, Kontingentüberwachung und Verbrauchsstatistiken
Konfigurationsmanagement	Identifikation, Konfiguration und Verwaltung von Konfigurationsinformationen von Netzwerkkomponenten
Leistungsmanagement	Statistiken, Daten über Systemleistung und Vorgehen zur Leistungsverbesserung
Sicherheitsmanagement	Verwaltung von Zugriffskontrolle, Schlüsselgenerierung und -verteilung und Erfassung von Sicherheitsereignissen

Tabelle 2.1: Aufgabenbereiche des Netzwerkmanagements.

unter dem Namen *SNMP*. Die Weiterentwicklung des CMOT verlief dagegen nicht so erfolgreich, so daß sich das SNMP-Konzept letztendlich als einziges durchsetzen konnte. Die Bezeichnung *SNMP* ist hier ein wenig irreführend, da SNMP keineswegs nur eine Bezeichnung für ein Protokoll ist. Vielmehr kennzeichnet es ein vollständiges Managementkonzept. SNMP wurde 1990 in den drei Dokumenten *SMI (Structure of Management Information)* [RM90], *MIB*[1] *(Management Information Base)* [MR90] und *SNMP* [CFSD90] genormt und wird heute als Standard-Managementkonzept in TCP/IP-Netzwerken eingesetzt [JS93].

2.3 SNMP

SNMP wird seit seiner Standardisierung im Jahre 1990 bis zum heutigen Zeitpunkt in vielen verschiedenen Komponenten, die an Netzwerken beteiligt sind, eingesetzt. Die Gruppe der Anwendungen erstreckt sich im Hardwarebereich von Bridges, Switches über Router bis hin zu DSL-Modems und zahlreichen anderen Geräten. Weiterhin wird SNMP auch im Softwarebereich wie z. B. in Webservern, Proxies, Datenbankservern usw. benutzt. Ein wichtiger Grundgedanke bei der erstmaligen Entwicklung des SNMP war die Einfachheit. SNMP soll nur geringe Anforderungen an die Ressourcen eines Gerätes oder Dienstes stellen, damit diese den größten Teil ihrer Kapazität für ihre eigentliche Aufgabe verwenden können. Dies erfordert einen einfachen Protokollaufbau und einfache Operationen zum Austausch von Managementinformationen. Ansonsten soll das System so flexibel sein, daß es mit wenigen Operationen, auch zukünftig, verschiedenste Geräte und Dienste ohne Änderungen am Protokoll ansteuern kann.

[1] Eine erweiterte Version der MIB ist 1991 als MIB-II erschienen. Siehe dazu auch [MR91].

2.3.1 Historie

Über die Zeit hinweg wurde SNMP selbst weiterentwickelt und verbessert, so daß später mit SNMPv2 (Simple Network Management Protocol Version 2) und SNMPv3 (Simple Network Management Protocol Version 3) weitere Versionen des SNMP folgten. Die erste Version wird zur Unterscheidung mit den Folgeversionen fortan SNMPv1 genannt.

Ein großer Mangel in SNMPv1 sind fehlende Sicherheitsmechanismen. SNMPv1 kann den Zugriff auf SNMP-Agenten nur durch einen Gruppennamen steuern, welcher zudem noch unverschlüsselt im Netzwerk übertragen wird. Somit existiert keine Möglichkeit, das Abhören von SNMP-Nachrichten, insbesondere des Gruppennamens, zu verhindern. Daraus folgt, daß es ebenso unmöglich ist, die Authentizität der Quelle einer Managementnachricht zu überprüfen. Ein Angreifer könnte beispielsweise durch Abhören von SNMP-Nachrichten den Gruppennamen in Erfahrung bringen und diesen anschließend in eigenen Nachrichten zur Manipulation des Netzwerkmanagements verwenden. Insofern beschloß die IETF (Internet Engineering Task Force) im Dezember 1991, das Thema Sicherheit als einen wichtigen Kernpunkt bei der Weiterentwicklung festzulegen. Für die Verbesserung der Sicherheit wurden daraufhin mehrere Dokumente verfaßt, welche in einer zukünftigen Version von SNMP einfließen sollen. Angesichts weiterer Mängel bezüglich der Leistung und Funktionalität entstand unter dem Namen *SMP (Simple Management Protocol)* ein neuer Entwurf des SNMP-Rahmenwerks. Dem folgend zeichnete sich im Juli 1992 ein Konsens in der Internetgemeinde ab, die Weiterentwicklung von SMP in Verbindung mit den sicherheitsbezogenen Dokumenten als SNMPv2 fortzuführen.

Bis Anfang 1993 erarbeiteten zwei Arbeitsgruppen einen gemeinsamen Vorschlag für einen SNMPv2-Standard. Das dort beschriebene Sicherheitsmodell *SNMP-Parties* basiert auf Authentifizierungs- und Verschlüsselungsmechanismen sowie Zugriffskontrollisten. Die darin enthaltenen Sicherheitsmechanismen wurden von der Industrie und den Anwendern allerdings nie richtig angenommen, so daß diese 1996 in einer überarbeiteten Version wieder fallengelassen wurden. Auch eine Neuintegration von Sicherheitsfunktionen blieb erfolglos, so daß SNMPv2 letztendlich ohne neue Sicherheitsfunktionen veröffentlicht wurde [CMRW96a]. Infolge dessen bildeten sich mehrere Splittergruppen, die verschiedene Sicherheitskonzepte verfolgten. Zum einem das bereits vorgeschlagene Konzept der *SNMP-Parties (party-based)* unter *SNMPv2p* und zum anderen das bekannte gruppenbasierte Konzept unter *SNMPv2c*. Obendrein entstanden mit *SNMPv2u* und *SNMPv2** zwei weitere Konzepte. SNMPv2u basiert auf Benutzerrechten *(user-based)*, und SNMPv2* ist eine Kombination aus Teilen von SNMPv2p und SNMPv2 ([Sta96], [JS93], [Koz05]).

Einige der funktionalen Erweiterungen von SNMPv2 ([CMRW96d], [CMRW96c], [CMRW96b]) sind im folgenden:

- Manager-zu-Manager-Kommunikation mittels PDU *InformRequest*
- Übertragung größerer Informationsmengen mittels PDU *GetBulkRequest*
- Erweiterung der SMI

- Einführung einer SNMPv2-MIB, u. a. zur Überwachung der SNMP-Kommunikation und Konfigurationsabfrage von Manager und Agenten
- Einführung von *Notifications* (SNMPv2-Trap)[2]

1996 begann ein neuer Ansatz für die Weiterentwicklung von SNMP, welcher 1998 in SNMPv3 verwirklicht wurde. Wichtige Punkte waren dabei neue funktionale Erweiterungen und – aufgrund der »fehlerhaften« Sicherheit in SNMPv2 – ein einheitliches Sicherheitskonzept. Funktionale Erweiterungen sind u. a. die Unterstützung von sicheren Schreiboperationen auf Variablen (Transaktionen) [Pos02]. Für SNMPv3 existieren mehrere standardisierte Sicherheitsmodelle. Eines basiert dabei auf Benutzerrechten wie in SNMPv2u [BW02]. Das in [WPM02] definierte Sicherheitsmodell garantiert die Sicherheit dagegen auf Basis verschiedener Sichten.

Des weiteren existiert seit Dezember 2002 ein experimenteller Entwurf für die Verwendung des verbindungsorientierten TCP (Transmission Control Protocol)[3] [Sch02]. Die Verwendung von TCP gegenüber dem verbindungslosen UDP (User Datagram Protocol) hat einen entscheidenden Vorteil: Fehlerhaft übertragene oder verloren gegangene Daten werden automatisch protokollseitig erneut verschickt. So kann es möglicherweise schwerwiegende Konsequenzen für ein Netzwerk haben, wenn wichtige Meldungen eines Agenten an einen Manager (*Trap*, s. Kapitel 2.3.3.2), welche dessen Reaktion verlangen, über UDP nicht korrekt zugestellt werden.

2.3.2 Modell

Das SNMP-Modell definiert mit *Manager* und *Agent* zwei verschiedene Komponenten in einem Netzwerk, welche für die Durchführung des Netzwerkmanagements zuständig sind. In der Regel kommuniziert ein Manager mit einem oder mehreren Agenten durch Austausch von Nachrichten. Die Kommunikation läuft dabei in Form eines *Request-Reply-Protokolls* ab. D. h., der Manager sendet eine Informationsabfrage oder einen Konfigurationsaufruf an einen Agenten. Dieser versucht daraufhin die gewünschte Operation auszuführen und liefert eine Antwort an den Manager zurück. Die Abfrage- bzw. Konfigurationsinformationen werden auf einem Agenten in sogenannten *managebaren Objekten* gehalten. In Ausnahmefällen kann der Agent auch von sich aus tätig werden und dem Manager Nachrichten schicken. Diese Nachrichten werden als *Traps* bezeichnet. Eine Verdeutlichung der Kommunikation zwischen Manager und Agenten liefert die Abbildung 2.1.

2.3.2.1 SMI

Ziel der SMI ist es, Elemente und Regeln zur Beschreibung von managebaren Objekten zu definieren. Der Zugriff auf die managebaren Objekte findet über eine virtuelle

[2] Traps (s. Abschnitt 2.3.3.2) werden seit der Einführung von SNMPv2 Notifications genannt und werden in der MIB gegenüber Traps in SNMPv1 anders definiert.

[3] Die Verwendung von TCP als Übertragungsprotokoll in SNMP beschränkt sich nicht nur auf SNMPv3, sondern kann mit jeder Version von SNMP verwendet werden.

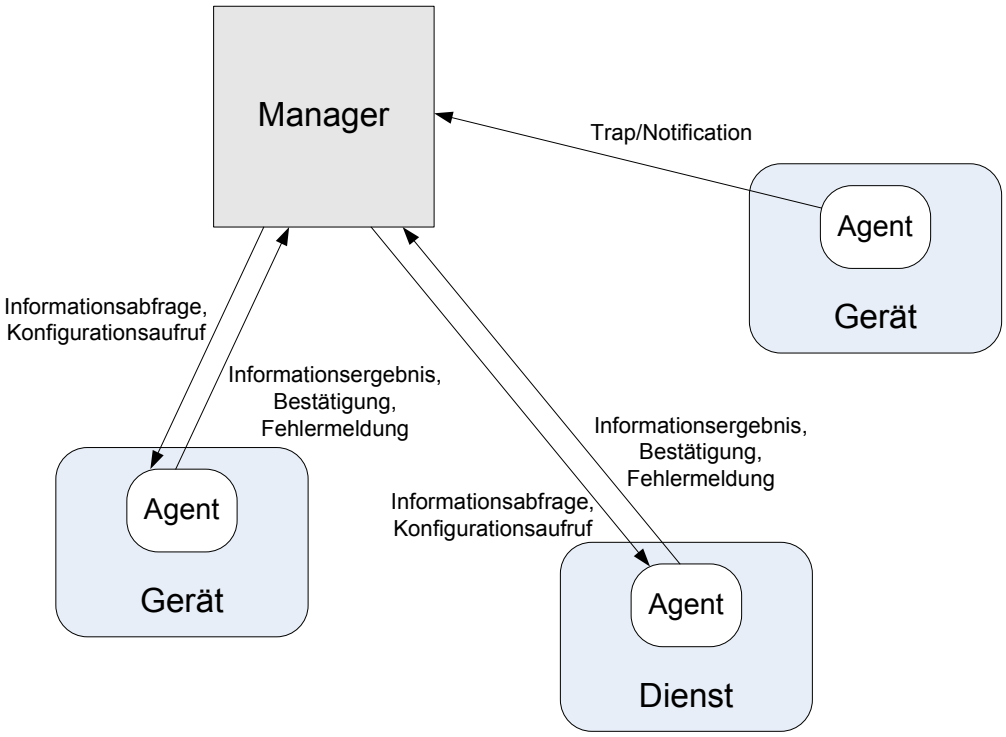

Abbildung 2.1: Kommunikationsablauf zwischen Manager und Agenten.

Datenbasis, als MIB bezeichnet, statt, in welcher alle vorkommenden Objekttypen beschrieben sind. Die Objekttypen werden in der MIB mittels ASN.1[4] (Abstract Syntax Notation One) dargestellt. Konkret besteht jeder Objekttyp aus einem *Namen*, einer *Syntax* und einer *Kodierung*. Da ASN.1 ziemlich umfangreich ist, schränkt SMI die Gruppe der zur Verwendung erlaubten ASN.1-Konstrukte ein [RM90].

Der Name eines jeden Objekttyps besteht aus einer eindeutigen Objekt-ID (OID). Alle Objekttypen sind fest in einer Baumstruktur organisiert, wobei diese die Blätter (Endknoten) des Baumes bilden. Jeder Knoten (ausgenommen der Wurzelknoten) des Baumes besitzt eine Nummer, neben den numerischen Bezeichnern gibt es aber teilweise auch noch alphanumerische Äquivalente (s. Abbildung 2.2). Gleiche Nummern dürfen auch mehrfach vergeben werden, solange jedoch alle Knoten eines direkt übergeordneten Knotens eindeutig durchnumeriert werden. Durch diese Durchnumerierung der Knoten existiert im Baum eine lexikographische Ordnung. Nach dieser kann jeder Objekttyp in einem Blatt durch einen eindeutigen Pfad adressiert werden. Die Objekt-ID eines Objekttyps ergibt sich nun aus der Aneinanderreihung der Nummern – durch Punkte getrennt – entsprechend des Pfades von der Wurzel zu einem Endknoten. Die Objekt-ID kann statt nur aus den Nummern auch aus den alphanumerischen Bezeichnern oder beiden bestehen ([RM90], [JS93]).

Die Syntax definiert die zugehörige Datenstruktur zu den Objekttypen. Die Datentypen

[4] ASN.1 ist ein Standard zur abstrakten Beschreibung von Datentypen, die Beschreibung ist dabei unabhängig von Rechnerarchitekturen. Nach ASN.1 spezifizierte Daten können mit verschiedenen Verfahren (Encoding Rules) serialisiert werden, um diese als Bitstrom übertragen zu können.

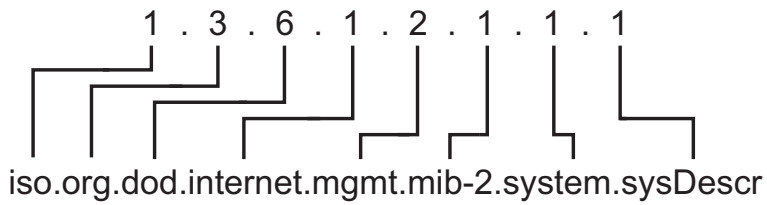

Abbildung 2.2: Objekt-ID eines Objekttyps.

sind in drei Gruppen, den Basisdatentypen (*primitive types*), den Konstruktionsdatentypen (*constructor types*) und den von Basistypen abgeleiteten Datentypen (*defined types*), unterteilt. Die Basisdatentypen sind ASN.1-Typen *INTEGER*, *OCTET STRING*, *OBJECT IDENTIFIER* und *NULL*. In der zweiten Gruppe befinden sich die Datentypen *SEQUENCE* für die Konstruktion von Listen und *SEQUENCE OF* für Tabellen. In der letzten Gruppe, den von Basistypen abgeleiteten Datentypen, befinden sich *NetworkAddress*, *IPAddress*, *Counter*, *Gauge*, *TimeTicks* und *Opaque* ([RM90], [JS93]).

Die Kodierung der nach ASN.1 spezifizierten Daten erfolgt in den BER (Basic Encoding Rules). Nach der Kodierung können die Daten einer Objektinstanz serialisiert über das Managementprotokoll übertragen werden.

Weiterhin ist für die managebaren Objekte auch noch ein Zugriffsmodus festgelegt. Dies ist notwendig, da einige Variablen nur gelesen oder beschrieben, andere wiederum gelesen und beschrieben werden dürfen. Auf einige Variablen ist der Zugriff überhaupt nicht gestattet, d. h., sie dürfen weder gelesen noch beschrieben werden. Dieser Zugriffsmodus hat eine besondere Bedeutung und wird bei Objekttypen verwendet, die wesentlich sind, aber auf die nicht zugegriffen werden darf. Dies trifft etwa auf Listen und Tabellen zu, da hier Objekttypen definiert werden, die andere Objekttypen gruppieren und ausnahmsweise keine Blätter in der Baumstruktur sind. Auf diese Objekttypen darf definitionsgemäß nicht zugegriffen werden, da diese nur den Inhalt von Listen und Tabellen bestimmen[5]. Für jede Variable ist immer einer der insgesamt vier Zugriffsmodi *lesen*, *schreiben*, *lesen und schreiben* und *kein Zugriff* festgelegt.

Eine neue Version der SMI ist im April 1999 (*Structure of Management Information Version 2 (SMIv2)* [MPS99a] und *Textual Conventions for SMIv2* [MPS99b]) als Standard herausgekommen. Diese Version wird *SMIv2*, die ursprüngliche SMI dementsprechend *SMIv1* genannt.

2.3.2.2 Objekttypen und Objektinstanzen

Die unter Konvention der SMI beschriebenen Objekttypen stellen nicht die real auf einem Agenten existierenden managebaren Objekte dar. Objekttypen sind vielmehr mit Objektklassen bei objektorientierten Programmiersprachen vergleichbar und bestimmen nur die Struktur eines Objekts. Die realen Abbildungen der Objekttypen auf

[5] In der MIB, die zusammen mit SNMPv1 standardisiert wurde, ist der Zugriffsmodus der inneren Objekttypen für Tabellen aufgrund eines Designfehlers auf *lesen und schreiben*, in der MIB-II dagegen korrekt auf *kein Zugriff* festgelegt.

einem Agenten sind die Objektinstanzen, welche dann die eigentlichen Daten (Wertzuweisung an ein Objekt) enthalten. Von einem Objekttyp können auf einem Agenten eine oder mehrere Instanzen vorhanden sein. Existiert nur eine Instanz, so wird diese als Skalar aufgefaßt, mehrere Instanzen sind dagegen immer in Listen oder mehrspaltigen Tabellen angeordnet. Je nach Anzahl der Instanzen werden unterschiedliche Suffixe an die Objekt-ID angehängt. Existiert von einem Objekttyp nur eine Instanz (Skalar), so wird einfach das Suffix *.0* (s. Abbildung 2.3) an die Objekt-ID angehängt. Auf den in Abbildung 2.2 bezogenen Objekttyp *sysDescr* zur Angabe der Systembeschreibung ergibt sich die Objekt-ID zu *1.3.6.1.2.1.1.1.0* bzw. deskriptiv angegeben zu *iso.org.dod.internet.mgmt.mib-2.system.sysDescr.0*. Existieren von einem Objekttyp dagegen mehrere Instanzen auf einem Agenten, so ist die Adressierung der Instanzen, aufgrund der Organisation in Listen oder mehrspaltigen Tabellen, komplexer. Das Instanzsuffix (eindeutiger Index für eine Zeile in einer Liste oder Tabelle) für die Objekt-ID kann dabei verschieden komplex aufgebaut sein. In einer einfachen Variante könnte dies ein fortlaufender Index sein, in einer komplexeren Variante dagegen ein Variablenwert wie z. B. eine IP-Adresse. Bei mehrspaltigen Tabellen kann sich die Bildung des Instanzsuffixes auch über mehrere Variablenwerte erstrecken. Die Bildung ist dabei von der jeweiligen Liste bzw. Tabelle abhängig.

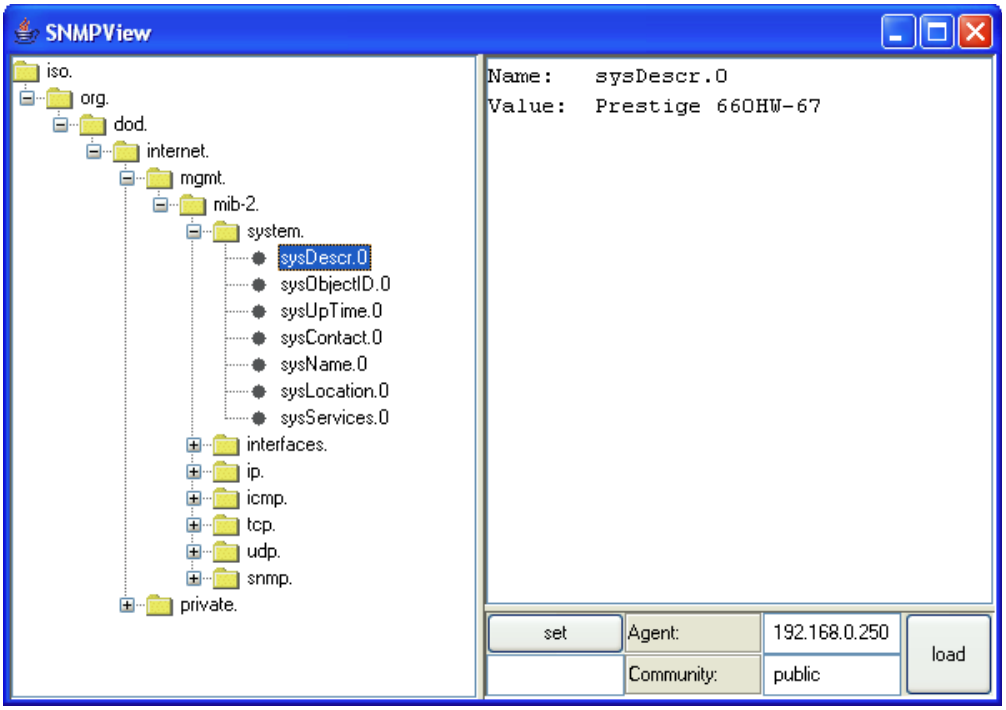

Abbildung 2.3: Abfrage von Objekten mittels MIB-Browser [Oec].

2.3.2.3 MIB

Bei der MIB handelt es sich um ein Datenmodell, welches sämtliche zu verwaltende Objekte eines SNMP-Agenten beschreibt. Die MIB definiert dagegen nicht, wie

die Objektdaten in einem Agenten oder Manager gespeichert werden. Die Quelltexte der MIBs liegen in Textdateien vor und sind nach Konvention der SMI aufgebaut. Der Inhalt der MIB besteht aus den Beschreibungen der Objekttypen. Je nach Anwendungszweck gibt es verschiedene MIBs, die auf die jeweiligen Geräte und Dienste zugeschnitten sind. Eine ganz bekannte und bereits erwähnte MIB ist die in [MR91] genormte MIB-II. Diese MIB findet oftmals im Internet mit Geräten wie Router, Bridges usw. Verwendung.

Die MIB-II hat bis zum heutigen Zeitpunkt zahlreiche Überarbeitungen und Erweiterungen erfahren, diese sind in den Dokumenten *SNMPv2 Management Information Base for the Internet Protocol using SMIv2* [McC96a], *SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2* [McC96b], *SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2* [McC96c], *The Interfaces Group MIB* [MK00] und *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)* [Pre02] definiert.

Der folgende Quelltext ist ein Ausschnitt aus der MIB-II [Pre02]:

```
sysDescr OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..255))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "A textual description of the entity.  This value
        should include the full name and version
        identification of the system's hardware type,
        software operating-system, and networking
        software."
    ::= { system 1 }

sysObjectID OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The vendor's authoritative identification of the
        network management subsystem contained in the
        entity. This value is allocated within the SMI
        enterprises subtree (1.3.6.1.4.1) and provides an
        easy and unambiguous means for determining 'what
        kind of box' is being managed.  For example, if
        vendor 'Flintstones, Inc.' was assigned the
        subtree 1.3.6.1.4.1.424242, it could assign the
        identifier 1.3.6.1.4.1.424242.1.1 to its 'Fred
        Router'."
    ::= { system 2 }
```

Bis heute hat die IETF über 200 verschiedene MIBs mit über 13.000 Variablen und mehr als 200 Traps und Notifications veröffentlicht. Organisationen wie IEEE (Institute of Electrical and Electronics Engineers) oder ATM-Forum (Asynchronous Transfer

Mode) haben zudem MIBs für ihre eigenen Technologien erstellt. Und außerdem haben Hersteller verschiedenster Geräte die bereits veröffentlichten MIBs noch einmal mit zusätzlichen Definitionen erweitert [Sch05]. Die ganze Vielfalt an verschiedenen MIBs gibt einen Hinweis darauf, wie erfolgreich sich SNMP in der Industrie durchsetzen konnte.

2.3.2.4 Manager

Der Manager ist eine Verwaltungseinheit des Netzwerkmanagements und dient der Kontrolle und Überwachung eines Netzwerkes. Dazu sendet der Manager Informationsanfragen oder Konfigurationsaufrufe an die Agenten im Netzwerk. Der Manager muß vor dem Datenaustausch mit dem Agenten jedoch wissen, welche Daten er mit diesem austauschen kann. Diese Informationen sind in einer MIB (s. Abschnitt 2.3.2.3) abgelegt. Um mit einem Agenten korrekt kommunizieren zu können, muß dem Manager die MIB des jeweiligen Agenten bekannt sein. Oftmals kommuniziert ein Manager auch mit mehreren Agenten gleichzeitig. Aufgrund der gelieferten Informationen eines Agenten kann der Manager beispielsweise die Umkonfiguration eines anderen Agenten veranlassen. In einem konkreten Beispiel sähe das folgendermaßen aus: Erkennt der Manager durch die Abfrage des Agenten eines Routers, daß dieser durch hohes Datenaufkommen überlastet ist, so kann der Manager andere Router umkonfigurieren und einen Teil des Datenaufkommens über diese umleiten.

Mit der in SNMPv2 eingeführten Manager-zu-Manager-Kommunikation ist es in einfacher Weise möglich, ein Managersystem durch mehrere Manager in einer Hierarchie aufzubauen. Dadurch lassen sich auch große Netzwerke komfortabel verwalten. Zum Beispiel ist jeder Manager der untersten Hierarchieebene für ein bestimmtes Netzsegment zuständig. Für das Management über Segmentgrenzen hinweg sind dagegen die Manager der darüberliegenden Hierarchieebene verantwortlich (s. Abbildung 2.4).

2.3.2.5 Agent

Der Agent ist oftmals Bestandteil eines managebaren Gerätes oder Dienstes im Netzwerk. Die Kombination eines Agenten mit einem Gerät bzw. Dienst wird auch als *Managementknoten* bezeichnet. Gelegentlich enthalten Geräte oder Dienste keine Agenten. Damit diese Komponenten auch in das Netzwerkmanagement integriert werden können, übernehmen gesondert ins Netzwerk eingegliederte Agenten (*Proxy-Agenten*) deren Überwachung und Steuerung (s. Abildung 2.5). Die eigentliche Aufgabe eines Agenten ist es, Informationen eines Gerätes oder Dienstes zu erfassen und einem Manager auf Anfrage zur Verfügung zu stellen. Weiterhin verwaltet der Agent auch Konfigurationsparameter eines Gerätes oder Dienstes und konfiguriert diese um, sofern ein Manager eine Umkonfiguration über diese Parameter veranlaßt. Diese Informationen werden in der MIB des Agenten abgelegt. In ganz dringenden Fällen kann der Agent auch von sich aus Nachrichten an den Manager schicken. Diese Nachrichten werden in Eigenregie verschickt, da sie dem Manager möglichst zügig zugestellt werden müssen. Ansonsten könnte der Manager diese Informationen nur durch Abfrage des Agenten bekommen, was je nach Abfrageintervall zu geringen oder größeren Verzögerungen

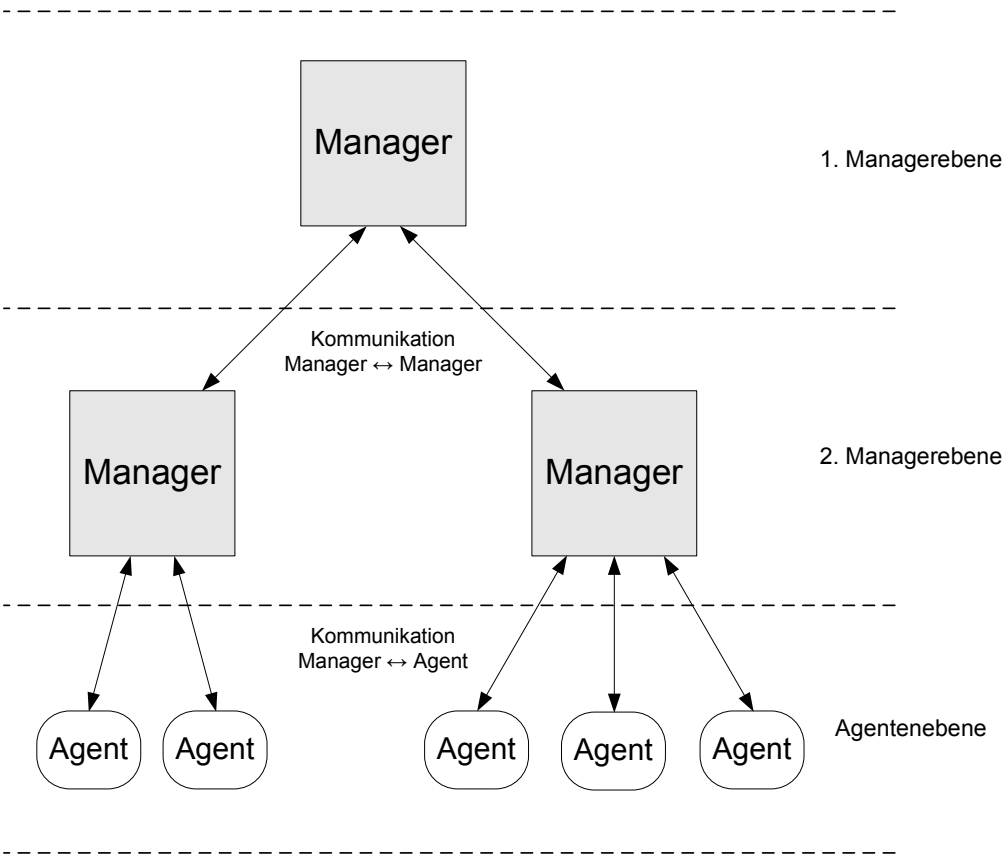

Abbildung 2.4: Netzwerkmanagement mit Managerhierarchie.

führen würde. Ein konkretes Beispiel könnte hierzu so aussehen: Ein Switch bemerkt in einem Netzwerk, daß eine Kabelverbindung unterbrochen wurde und meldet diesen Vorgang schnellstmöglich dem Manager. Daraufhin kann ein Administrator den Fehler innerhalb kürzester Zeit beheben, indem er die Verbindung wiederherstellt. Das Netzwerkmanagement trägt so zur Qualitätssicherung des Netzwerks bei.

2.3.2.6 Operationen

Für den Datenaustausch zwischen Manager und Agent definiert SNMPv1 verschiedene Operationen, die es ermöglichen, auf die Variablen eines managebaren Objekts zuzugreifen. Zur Ausführung der Operationen sind noch weitere wichtige Parameter wie Objekt-ID und Gruppenname[6] (community name) notwendig. Die folgenden vier Operationen bewerkstelligen die Kommunikation:

- get
- get-next

[6]Der Gruppenname teilt sämtliche SNMP-Komponenten in einem Netzwerk in Gruppen ein. Bei ihm handelt es sich um eine kurze Zeichenfolge. Wenn Manager und Agent miteinander kommunizieren wollen, so müssen sie den gleichen Gruppennamen aufweisen.

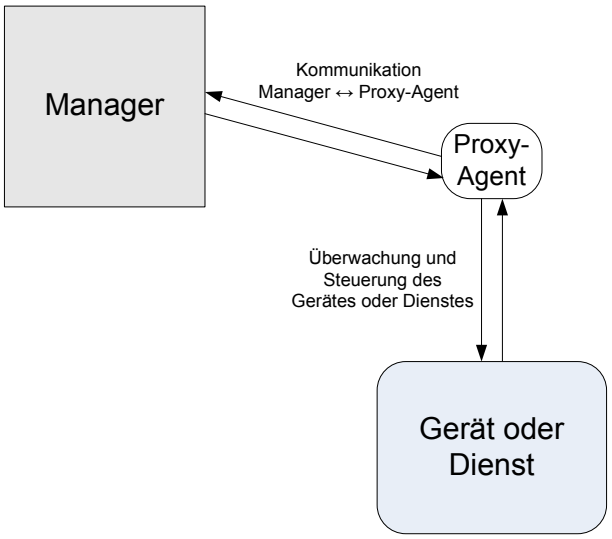

Abbildung 2.5: Netzwerkmanagement über Proxy-Agenten.

- set
- trap

Die Operation *get* bewirkt den direkten Zugriff auf eine Instanz eines Objekttyps, welche durch die Objekt-ID mit Instanzsuffix angegeben wurde. Da das Instanzsuffix, wie in Abschnitt 2.3.2.2 aufgezeigt, bei Listen und Tabellen nicht unbedingt bekannt ist, existiert hierfür die Operation *get-next*. Ihre Aufgabe ist es, die erste folgende Objektinstanz der übergebenen Objekt-ID zurückzuliefern. *Set* ändert hingegen Werte von Objektinstanzen, und mit *trap* meldet der Agent wichtige Ereignisse an den Manager.

Die Operationen bilden mit ihren zugehörigen Parametern eine Einheit, welche als PDU (Protocol Data Unit) bezeichnet wird. Da Anfragen auch beantwortet werden müssen, gibt es neben den PDUs zu den Operationen noch eine weitere PDU für die Antworten der Anfragen. Demnach existieren folgende fünf PDUs:

- get-request
- get-next-request
- get-response
- set-request
- trap

2.3.3 Kommunikationsmethoden

Die Kommunikation zwischen Manager und Agent findet durch zwei verschiedene Methoden statt. Die beiden Methoden verfolgen verschiedene Anwendungsgebiete,

zum einen das regelmäßige Abfragen des Agenten durch den Manager und zum anderen den umgekehrten Weg mit der Benachrichtigung des Managers durch den Agenten.

2.3.3.1 Regelmäßiges Abfragen des Agenten (Polling)

Das Abfragen der Agenten durch einen Manager ist die Standardmethode im Netzwerkmanagement durch SNMP. Der Manager fragt die zu überwachenden Variablen in einem zuvor festgelegten Intervall ab und entscheidet je nach Variablenwert, welche weiteren Aktionen auszuführen sind. Durch das ständige Abfragen kann Netzwerkverkehr entstehen, welcher das Netzwerk unnütz belastet, wodurch dieses anderen Netzwerkteilnehmern für die Datenübertragung nicht mehr zur Verfügung steht. Für Variablen, welche Statistikdaten liefern, sind kurze Abfrageintervalle je nach Statistik angemessen. Für Variablen, deren Werte sich selten ändern, sind kurze Abfrageintervalle dagegen ungeeignet. Ein falsch definiertes Abfrageintervall könnte das Netz zudem so stark belasten, daß ein normaler Datentransfer nur noch eingeschränkt möglich ist. Der Manager wäre dann selbst ursächlich dafür, daß dieser in die Netzwerkkonfiguration eingreifen müßte, um den normalen Datenverkehr aufrecht zu erhalten.

2.3.3.2 Benachrichtigung durch Agenten (Trap/Notification)

In ganz dringenden Fällen kann der Agent nicht warten, bis der Manager nach Ablauf seines Abfrageintervalls die Variable abruft, welche die dringende Nachricht enthält. In so einer Situation wird der betroffene Agent von sich aus aktiv und schickt eine Benachrichtigung an den Manager, so daß dieser sofort alle erforderlichen Maßnahmen treffen kann. Hier muß der Agent aber zwischen wichtigen und unwichtigen Situationen differenzieren und selbst entscheiden können, in welchen Fällen er Nachrichten verschickt. Ein willkürlicher Nachrichtenversand durch den Agenten würde das Netzwerk wie in der ersten Abfragemethode ebenfalls mit unnützem Datenverkehr belasten, so daß die eigentlichen Nutzdaten der anderen Netzwerkteilnehmer wieder nur eingeschränkt übertragen werden können. SNMPv1 definiert die in Tabelle 2.2 abgedruckten Typen von Traps. Traps werden mit einem der abgedruckten Typen nach Konvention der SMI in der MIB definiert [Ros91].

2.4 NETCONF-Protokoll

Neben SNMP als vollwertiges Managementkonzept befindet sich mittlerweile ein weiterer Ansatz in der Entwicklung, welcher ein einfaches Netzwerkmanagement realisieren soll. Hierbei handelt es sich um das *NETCONF-Protokoll* [Enn05] mit einem XML-basierten Nachrichtenaustausch.

Im Jahr 2003 wurde von der IETF eine neue Arbeitsgruppe *Network Configuration (NETCONF)* zugelassen, deren Aufgabe es ist, einen neuen Standard im Bereich Netzwerkmanagement zu schaffen. NETCONF soll dabei einen einfachen Mechanismus zum Managen von Netzwerkgeräten bereitstellen. Der Datenaustausch erfolgt wie in

Trap	Beschreibung
coldStart	Neuinitialisierung des Agenten, Objekte seiner MIB könnten sich geändert haben
warmStart	Neuinitialisierung des Agenten, Objekte seiner MIB haben sich nicht geändert
linkDown	Verbindung zu einer Kommunikationsschnittstelle wurde unterbrochen
linkUp	Verbindung zu einer Kommunikationsschnittstelle wurde wiederhergestellt
authenticationFailure	Authentifizierung einer empfangenen Protokollnachricht ist fehlgeschlagen
egpNeighborLoss	Verbindung zu einem anderen Knoten über das EGP (Exterior Gateway Protocol) wurde unterbrochen
enterpriseSpecific	Herstellerspezifische Ereigniskennung

Tabelle 2.2: Typen von Traps in SNMPv1.

SNMP mittels eines *Request-Reply-Protokolls*. Weiterhin verläuft der Datenaustausch zwischen Manager und Agenten über einen Mechanismus auf Basis eines Remoteprozeduraufrufs[7] (*Remote Procedure Call (RPC)*), welcher zuvor in *XML (Extensible Markup Language)* [BPSM+04] kodiert wird. Für den Ablauf eines Remoteprozeduraufrufs kodiert ein Manager den Bezeichner einer Remoteprozedur mit weiteren Parametern, die später an die Remoteprozedur übergeben werden, in XML. Danach schickt er diese Nachricht an einen Agenten. Dieser wertet daraufhin die XML-Daten aus und ruft die angegebene Prozedur mit Übergabe der Parameter auf. Der Rückgabewert der Prozedur wird von dem Agenten ebenfalls wieder in XML kodiert und an den Manager zurückgeschickt.

Ein wesentlicher Aspekt des NETCONF-Protokolls ist, daß die Funktionalität des Managementprotokolls möglichst die nativen Funktionen eines Gerätes widerspiegeln soll. Daraus ergeben sich zwei wesentliche Vorteile. Zum einen bleiben die Implementierungskosten gering, da keine aufwendigen Funktionen für die Informationsabfrage und Konfiguration implementiert werden müssen. Zum anderen sind neu implementierte Funktionen der Geräte sofort nutzbar, da sie über RPC ansprechbar sind. Neben dem Hauptaspekt gibt es weitere Punkte, die bei der Entwicklung von NETCONF eine entscheidende Rolle spielen:

- Abfragemechanismen zur Unterscheidung von Konfigurations- und Informationsdaten
- Zugriff auf sämtliche Konfigurationsdaten über ein einziges Protokoll
- Programmierschnittstelle (Application Programming Interface (API))

[7] Ein Remoteprozeduraufruf bezeichnet im weiteren Sinne einen entfernten Funktionsaufruf in verteilten Systemen. Die Schnittstelle eines RPC-Mechanismus abstrahiert insoweit, daß kein Unterschied zwischen entfernten und lokalen Funktionsaufrufen erkennbar ist.

- Textuelle Datenrepräsentation zur einfachen Datenmanipulation
- Unterstützung für die Integration existierender Methoden zur Benutzerauthentifizierung
- Unterstützung für die Integration existierender Konfigurationsdatenbanken
- Unterstützung für Transaktionen im Netzwerk
- Unabhängigkeit vom Transportprotokoll
- Unterstützung für asynchrone Benachrichtigungen

NETCONF ist auch bezüglich der Trägerprotokolle sehr flexibel, da es kein bestimmtes voraussetzt und deshalb mit verschiedenen einsetzbar ist. Diese Flexibilität hat den Vorteil, daß NETCONF zukünftig auf Geräten mit unterschiedlichen Protokollen einfach nutzbar ist. Die Trägerprotokolle müssen allerdings einige Voraussetzungen erfüllen, dies sind unter anderem Authentifizierung, Wahrung der Datenintegrität und Abhörsicherheit der Daten. Zur Zeit erlauben mehrere Ausarbeitungen der Arbeitsgruppe die Nutzung von NETCONF über drei Protokolle. Diese sind im einzelnen:

- BEEP (Blocks Extensible Exchange Protocol) [LC05]
- SSH (Secure Shell) [WG05]
- SOAP [God05]

Die Verwendung von SOAP [Mit03] als Trägerprotokoll für NETCONF ist unter dem Gesichtspunkt *Webservices* ein interessantes Thema, da Webservices gleichermaßen SOAP zur Kommunikation einsetzen. Somit ist es unter Umständen realisierbar, das Netzwerkmanagement einzelner Dienste oder Geräte als Webservice anzubieten. Besonders interessant ist die Erfahrung, ob die Benachrichtigung von Managern durch Agenten über Webservices praktikabel machbar ist. Diese Kommunikationsmethode ist der Ansatz für den in dieser Diplomarbeit untersuchten Hauptgegenstand *Agentenbasiertes Monitoring von Netzressourcen auf Basis von Webservices*.

3 Verteilte Systeme und Webservices

Nach einer kurzen Einführung in verteilte Systeme zeigt dieses Kapitel den Zusammenhang zwischen verteilten Anwendungen und Webservices auf. Daneben werden die verschiedenen Technologien erläutert, auf die Webservices aufsetzen und den Datenaustausch in heterogenen verteilten Systemen abwickeln.

3.1 Einführung in verteilte Systeme

Ein verteiltes System charakterisiert eine Menge von (heterogenen) Hardware- oder Softwarekomponenten, die sich verteilt auf vernetzten Computern befinden und durch den Austausch von Nachrichten miteinander kommunizieren und sich so koordinieren. Alle beteiligten Komponenten bilden mit ihrer Funktionalität zusammen ein Gesamtsystem. Die Motivation für den Aufbau eines Systems als verteiltes System besteht primär in der Ausnutzung von Nebenläufigkeit und der gemeinsamen Verwendung von Ressourcen. Bei der Nebenläufigkeit arbeiten Teile eines verteilten Systems unabhängig voneinander und können parallel ablaufen. Dies hat oftmals den Zweck, die Leistung einer Anwendung zu steigern und die Last gleichmäßig zu verteilen. Bei der gemeinsamen Verwendung von Ressourcen spielen dagegen Zeit- und Kostenfaktor eine übergeordnete Rolle. Ressourcen können Software wie Anwendungen und Anwendungsdaten, aber auch Hardware wie Drucker, DSL-Anschluß u. ä. sein. Komponenten stellen ihre Ressourcen entweder für andere zur Verfügung oder umgekehrt, Komponenten benutzen bereitgestellte Ressourcen. Die gemeinsame Ressourcennutzung kann deswegen erhebliche Einsparungen an Kosten und Zeit mit sich bringen. Dies führt zum Beispiel dazu, daß Daten nur einmal vorgehalten werden müssen, dadurch entfällt die sonst zeitaufwendige Synchronisation der Bestände untereinander. Im Bereich der Hardware können Drucker, Scanner usw. von mehreren Anwendern gleichberechtigt benutzt werden, hier entfallen dann die Anschaffungskosten für weitere Geräte. Es gibt aber auch noch weitere Gründe, ein System als verteiltes System auszulegen. Durch eine redundante Integration von Ressourcen kann die Ausfallsicherheit des Systems erhöht werden. Fällt eine Ressource aus, kann das System ungestört weiterarbeiten, da vergleichbare Ressourcen die Aufgaben der anderen übernehmen können. Weiterhin skaliert ein verteiltes System besser als ein Einzelsystem. Wird das System steigenden Leistungsanforderungen nicht mehr gerecht, so können weitere Ressourcen integriert und unter Berücksichtigung einer gleichmäßigen Lastverteilung genutzt werden.

Das größte verteilte System bildet zweifelsfrei das weltweite Internet mit einem großen Zusammenschluß von Computernetzwerken. Programme werden auf den am Internet angeschlossenen Computern ausgeführt und kommunizieren untereinander durch den

Austausch von Nachrichten. Im Internet befinden sich eine etliche Anzahl bereitgestellter Dienste (z. B. Informations- und Kommunikationsdienste). Je nach Anwendungsgebiet haben die Dienste eine unterschiedliche Zielgruppe. Auf Anwender zugeschnittene Dienste legen oftmals viel Wert auf eine visuelle Darstellung. Ein solcher Dienst könnte gegebenenfalls als Webanwendung realisiert sein. In diesem Fall wird die Anwendung auf einem Webserver ausgeführt, und der Anwender kann dann mit einem Webbrowser darauf zugreifen. Genauso gibt es im Internet aber auch Dienste, die nicht für den Anwender gedacht sind und für die automatische Verarbeitung durch Computer statt der visuellen eine genau strukturierte Darstellung fordern. Diese Dienste finden meist in Unternehmen Anwendung und stellen Daten für Geschäftsprozesse (z. B. Datenbanken) zur Verfügung. Der Zugriff auf Dienste mit strukturierter Datendarstellung ist gut über Webservices realisierbar.

3.2 Middleware

Die Kommunikation zwischen Anwendungen ist eine komplexe Aufgabe, da auszutauschende Daten von der Quelle an das richtige Ziel gelangen und dort korrekt interpretiert werden müssen. Hierfür ist erstens ein einheitliches Datenformat essentiell, welches die Anwendungen verstehen, und zweitens spielen eine Menge weiterer Faktoren eine wichtige Rolle. Diese sind im einzelnen die Programmiersprache der Anwendung, das verwendete Betriebssystem und gegebenenfalls noch dessen Netzwerkprotokolle. Um die Anwendungsprogrammierer von allen diesen Details des Datenaustauschs zu entlasten, werden diese in einer gesonderten Softwareschicht – als Middleware bezeichnet – gekapselt. Die Middleware bietet Dienste zum Datenaustausch zwischen Anwendungen oder auch zum entfernten Prozeduraufruf an. Aufgrund der Inanspruchnahme der Dienste verbirgt die Middleware die Komplexität der Infrastruktur und Heterogenität der zugrundeliegenden Netzwerke, Betriebssysteme und Programmiersprachen. Zwei überaus bekannte Middlewaretechnologien sind einerseits das von Microsoft ursprünglich für Windows-Betriebssysteme entwickelte *DCOM (Distributed Component Object Model)* [Mica] und andererseits das plattformunabhängige *CORBA (Common Object Request Broker Architecture)* [OMG].

Mit einfachen Worten ausgedrückt, übergibt eine Anwendung die Daten, die sie übertragen möchte, an die Middleware. Diese verpackt sie daraufhin in ein einheitliches Datenformat. Anschließend verschickt die Middleware die Daten mit Hilfe des Betriebssystems an die Middleware der Zielstelle. Das geschieht entweder über einfache Interprozeßkommunikation, falls die Anwendungen auf demselben Rechner ausgeführt werden, oder über ein Netzwerk oder ähnliche Transportwege, falls die Anwendungen auf verschiedenen Rechnern laufen. Die Middleware der Zielstelle dekodiert die empfangenen Daten und übergibt sie an die Zielanwendung.

Der Fall eines Remoteprozeduraufrufs sieht so aus, daß eine Anwendung lokal eine Prozedur aufruft, die aber nicht im eigenen Bereich zu finden ist. Die Middleware fängt diesen Prozeduraufruf ab und kodiert ihn samt Parameter in einen fest strukturierten Datenblock. Dieser Datenblock wird dann mit den o. g. Betriebsmitteln an die

Middleware der Zielstelle übermittelt. Diese dekodiert darauf die Daten des Datenblocks und ruft die angegebene Prozedur mit den zugehörigen Eingabeparametern auf. Nach Abarbeitung der Prozedur nimmt sie mögliche Rückgabewerte entgegen und kodiert diese wieder in einen fest strukturierten Datenblock. Der Datenblock wird danach an die Middleware der Gegenstelle übermittelt. Diese dekodiert den Datenblock und übergibt den Rückgabewert an die Anwendung, die den Prozeduraufruf veranlaßt hat. Wie bereits in Unterkapitel 2.4 erwähnt, erscheint der Remoteprozeduraufruf für die Anwendung völlig transparent, da die Prozedur nur lokal aufgerufen wird, die Middleware den Aufruf aber unbemerkt abfängt [CDK02].

3.3 Webservices

Webservices sind Anwendungen, die in einem Netzwerk zur Verfügung stehen und gegenwärtig zum Austausch von Anwendungsdaten über Netzwerke verwendet werden. Sie bilden inzwischen einen wesentlichen Bestandteil unserer Kommunikationsinfrastruktur und sind an sich auf zusätzliche Rechner ausgelagerte Prozeduren eines Softwaresystems, auf die über ein Netzwerk zugegriffen werden kann. Webservices dienen im allgemeinen dazu, wie im vorherigen Abschnitt bereits angedeutet, im Internet bereitgestellte Anwendungsfunktionalität für andere Anwendungen verfügbar zu machen. Deren Einsatz ist jedoch nicht zwingend auf das Internet beschränkt. Demnach übernehmen Webservices auch die Rolle einer Middleware, indem sie die Komplexität des Netzes vor der Anwendung verbergen.

Webservices orientieren sich an der *SOA (Service Oriented Architecture)* und sind hauptsächlich auf betriebswirtschaftliche Anwendungen und Lösungen im Internet ausgerichtet. Hinter dem Konzept von SOA steckt der Ansatz, Funktionen durch voneinander unabhängige Dienste anzubieten. Für die Dienste existieren zuvor fest definierte Schnittstellen, über welche sie aufgerufen werden können. So bieten auch Webservices ihre Funktionalität als Gruppe von Diensten an, die Anwendungen über definierte Schnittstellen in Anspruch nehmen können.

Für den Begriff Webservices gibt es mittlerweile recht unterschiedliche Definitionen, die im Kern das gleiche aussagen, sich aber in den Details unterscheiden. Die Organisation W3C (World Wide Web Consortium) hat auch eine Definition [ABFG04] des Begriffs herausgebracht. Das W3C ist für die Standardisierung von Webtechnologien zuständig und standardisiert neben anderen Technologien auch Webservices.

> »A Web service is a software system identified by a URI [BLFM98], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.«

Die deutsche Übersetzung der o. g. Definition:

> »Ein Webservice ist ein durch ein URI identifiziertes Softwaresystem, dessen öffentliche Schnittstellen mit XML definiert und beschrieben werden. Die Definition der Schnittstellen kann von anderen Softwaresystemen aufgefunden werden. Diese Systeme können dann nach Konvention der Schnittstellendefinition über XML-Nachrichten, die über Internetprotokolle transportiert werden, miteinander interagieren.«

Nach genauerer Betrachtung der obigen Definition sind Webservices Anwendungen, die fest definierte Schnittstellen haben. Diese Schnittstellen werden in einer bestimmten Sprache definiert, welche XML heißt. Die Schnittstellendefinitionen werden anderen Anwendungen irgendwie zugänglich gemacht, dabei spielt die Art und Weise hier keine Rolle. D. h., andere Anwendungen können die festgelegten Schnittstellendefinitionen der Webservices aufrufen und als Basis für die Kommunikation verwenden. Bei der Kommunikation über die Schnittstellen tauscht der Webservice mit den anderen Anwendungen Nachrichten aus, die auch auf der Sprache XML basieren. Jeder Webservice ist zudem noch eindeutig identifizierbar. Die Identifizierung geschieht dabei über ein URI[1] (Uniform Resource Identifier), welcher auch im WWW (World Wide Web) verwendet wird.

3.4 Historie der Webservices

Die Technologie der Webservices ist nicht wie bei vielen anderen Technologien akademischer Natur, sondern in der Industrie entstanden. Wie im Abschnitt 3.1 beschrieben, gibt es im Internet sowohl für den Anwender als auch für reine Geschäftsprozesse gedachte Dienste. Zuerst gab es nur die anwenderorientierten Webanwendungen, die häufig über den Webbrowser bedienbar waren. Mit der Zeit entdeckten auch die Firmen das Internet für sich. Im Vordergrund standen die globale Vernetzung der Firmen und Zweigstellen von Firmen untereinander und die Senkung von Kosten. Für eine einwandfreie Kommunikation mußte daher ein einheitliches Datenformat zur Kommunikation zwischen heterogenen Systemen und eine Abstrahierung der Software vom Netzwerk geschaffen werden. Genau diese Punkte erfüllt die in Unterkapitel 3.2 beschriebene Middleware. Da die Protokolle von CORBA und DCOM aber zu komplex und fehleranfällig waren, begann 1997 die Entwicklung eines neuen Protokolls auf Basis von ASCII (American Standard Code for Information Interchange). Das neu entwickelte Protokoll trägt den Namen *SOAP*. SOAP ist heute der Angelpunkt von Webservices, da über dieses Protokoll die Kommunikation abläuft und der Prozeduraufruf realisiert wird. Nach der Entwicklung des Protokolls entwarfen die an der Entwicklung beteiligten Firmen einen Verzeichnisdienst, um Webservices öffentlich bekannt zu machen und überall lokalisieren zu können. Der Verzeichnisdienst für Webservices heißt *UDDI (Universal Description, Discovery and Integration)*. Damit der Nachrichtenaustausch stattfinden kann, müssen die Schnittstellen zuvor eindeutig definiert werden, dies geschieht über die *WSDL (Web Service Description Language)*

[1] Ein Uniform Resource Identifier ist eine Zeichenfolge zur Identifizierung einer abstrakten oder physikalischen Ressource. Er beinhaltet keine Informationen über den Typ der Ressource und gibt auch kein Zugriffsprotokoll vor.

[BL05]. Diese Sprache basiert wie die Kommunikation auf XML und gibt die notwendigen Elemente zur Beschreibung der Schnittstellen vor [EF03].

Die Grundlage von Webservices bilden genau genommen die Standards XML, SOAP, WSDL und UDDI, wobei SOAP und WSDL selbst wieder auf XML aufsetzen. Eine nähere Beschreibung der Webservicekomponenten erschließt sich aus den folgenden Abschnitten dieses Kapitels.

3.5 XML

Die Extensible Markup Language ist eine standardisierte Sprache zur Generierung maschinenlesbarer Dokumente. XML stammt von der Sprache SGML (Standard Generalized Markup Language) ab, bzw. genauer genommen ist XML eine Teilmenge davon. XML-Dokumente bestehen aus Elementen, Attributen und Wertzuweisungen. Die Elemente dienen der Strukturierung von Dokumenten und grenzen unterschiedliche Daten voneinander ab. XML gibt eine strenge Struktur für die Daten vor, damit Maschinen diese auf einfache Weise verarbeiten können. Die Elemente werden in XML durch *Tags* definiert. Die Tags werden im folgenden Verlauf noch genauer erläutert. Neben den bereits erwähnten Bestandteilen können Dokumente auch Verarbeitungsanweisungen und Kommentare in Form von Tags enthalten, die aber zur Unterscheidung von strukturierenden Tags speziell gekennzeichnet sind.

XML stellt eine generelle Anforderung an die Dokumente, alle beschriebenen Dokumente müssen *wohlgeformt* sein. Wohlgeformt bedeutet, daß sämtliche durch XML spezifizierte Syntaxregeln in den Dokumenten eingehalten werden müssen. Weiterhin sind wohlgeformte Dokumente immer in einer Baumstruktur aufgebaut. Die Organisation in einer Baumstruktur ergibt sich durch die Anordnung der Tags im Dokument. Eine genauere Betrachtung der Tags in 3.5.1.1 verdeutlicht dies.

3.5.1 Tags

Tags sind Schlüsselwörter zur Strukturierung von XML-Dokumenten, sie bestehen aus einem eindeutigen Bezeichner, welcher keine Leerzeichen enthalten darf. Damit Tags von den eigentlich zu strukturierenden Daten unterscheidbar sind, werden diese durch spitze Klammern umschlossen.

```
<Tag>
```

Die Bedeutung von Tags ist nicht von vornherein festgelegt, da der XML-Standard nur definiert, was Tags sind und nicht deren Bedeutung vorschreibt. Die Interpretation der Tags bleibt allein den verarbeitenden Anwendungen überlassen.

Datenstrukturierende Tags unterscheiden sich zwischen öffnenden, schließenden und leeren Tags. Allgemein gehört zu jedem öffnenden ein schließendes Tag. Der Unterschied zwischen einem öffnenden und schließenden Tag besteht in einem Schrägstrich, der dem Bezeichner des schließenden Tags vorangestellt wird.

```
</Tag>
```

Ein leeres Tag unterscheidet sich von den anderen insofern, daß der Schrägstrich nicht vor, sondern hinter dem Bezeichner steht.

```
<Tag/>
```

3.5.1.1 Elemente

Elemente sind eine Kombination eines öffnenden mit einem entsprechenden schließenden Tag, die Bezeichner beider Tags müssen übereinstimmen. Das öffnende Tag steht im Dokument unterdessen immer vor dem zugehörigen schließenden Tag. Zwischen einem öffnenden und schließenden Tag können Daten stehen, welche gegliedert werden sollen. Das folgende Beispiel zeigt ein Element mit eingeschlossenen Daten, eine Anwendung könnte die Daten als Typ eines Gerätes auffassen.

```
<Typ>Router</Typ>
```

Es ist aber nicht zwingend erforderlich, daß Elemente Daten einschließen. Für die Interpretation des obigen Beispiels kann der Typ eines Gerätes unbekannt oder einfach nicht angegeben sein.

```
<Typ></Typ>
```

Leere Elemente können aber auch einfacher durch ein einziges Tag, dem leeren Tag ausgedrückt werden. XML sieht vor, daß ein öffnendes Tag mit direkt folgendem schließenden Tag durch das dem Bezeichner entsprechende leere Tag ersetzt werden kann. Leere Elemente dürfen aber nicht einfach weggelassen werden, da dieses die Gültigkeit von XML-Dokumenten beeinflussen kann. Das obige Element durch das leere Tag ausgedrückt:

```
<Typ/>
```

Anhand der Kombination der unterschiedlichen Tagtypen ergibt sich in Verbindung mit einer Verschachtelung der Tags der Dokumentaufbau in einer Baumstruktur. Die Verschachtelung von Tags ist grundsätzlich erlaubt, solange einige Regeln eingehalten werden. Tags dürfen insofern verschachtelt werden, daß ein öffnendes und entsprechendes schließendes Tag bzw., falls keine Daten zwischen den beiden Tags stehen, ein äquivalentes leeres Tag von anderen öffnenden und schließenden Tags umschlossen werden. Oder anders ausgedrückt, schließende Tags müssen in umgekehrter Reihenfolge zu den öffnenden Tags stehen, vergleichbar mit der Klammerung in mathematischen Formeln. Ein Beispiel für eine korrekte Schachtelung von Tags ist:

```
<Geraet>
  <Typ/>
  <Revision>
    <Hauptrevision>1</Hauptrevision>
    <Unterrevision>42</Unterrevision>
  </Revision>
</Geraet>
```

Wird der obige Ausschnitt aus einem XML-Dokument als Baumstruktur dargestellt, dann bilden die Elemente die Knoten des Baumes (rechteckige Felder), die Daten befinden sich in den elliptisch dargestellten Blättern (s. Abbildung 3.1).

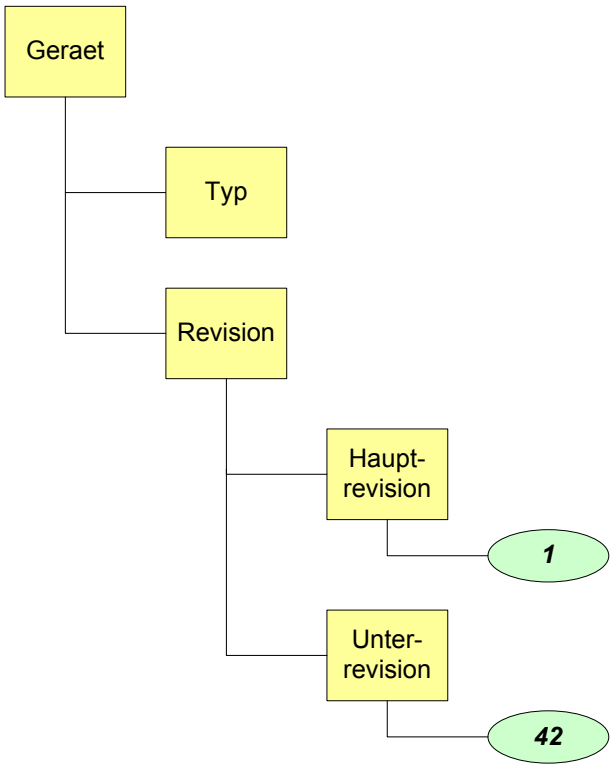

Abbildung 3.1: Baumstruktur der XML-Elemente.

3.5.1.2 Attribute

Elemente können nicht nur Daten oder andere Elemente einschließen, sondern auch mit Attributen verknüpft werden. Die Attribute stehen in dem öffnenden oder leeren Tag eines Elements und werden durch Leerzeichen von dem Tagbezeichner abgetrennt. Sind in einem Element mehrere Attribute enthalten, so sind diese ebenfalls durch Leerzeichen voneinander zu trennen. Attribute treten immer in der Form `Attributname = "Wertzuweisung"` auf, dabei ist zu beachten, daß die Wertzuweisung immer in Anführungszeichen steht. Einfache Daten können auch als Attribut eines äußeren Elements angegeben werden, statt als innere Elemente des äußeren Elements zu stehen. Auf das obige Beispiel bezogen, kann die Revisionsnummer eines Gerätes beispielsweise als Attribut angegeben werden.

```
<Geraet Revision="1.42">
  ...
</Geraet>
```

Als leeres Tag sieht »Geraet« so aus:

```
<Geraet Revision="1.42"/>
```

3.5.1.3 Verarbeitungsanweisungen und Kommentare

Als spezielle Tags können Dokumente auch Verarbeitungsanweisungen und Kommentare enthalten. Verarbeitungsanweisungen und Kommentare treten als alleinstehendes Tag auf, d. h., es gibt kein öffnendes und schließendes Tag. Verarbeitungsanweisungen sind durch je ein Fragezeichen hinter der öffnenden und vor der schließenden spitzen Klammer gekennzeichnet. Eine wichtige Verarbeitungsanweisung ist der XML-Prolog, der die Versionsnummer des verwendeten XML-Standards angibt, wahlweise mit zusätzlichen Attributen.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
```

Kommentare beginnen dagegen mit der Zeichenfolge `<!--` und enden mit `-->` und können sich über mehrere Zeilen erstrecken.

```
<!--Dies ist ein Kommentar-->
```

3.5.2 Namensraum

Namensräume (*engl. namespaces*) bieten eine Möglichkeit, die Bezeichner von Elementen und Attributen in XML-Dokumenten eindeutig zu kennzeichnen. Element- und Attributnamen werden mit einem Namensraum verknüpft, dieser wird wiederum eindeutig durch ein URI identifiziert. Auch wenn ein Namensraum als URI angegeben wird, besagt das nicht, daß sich dahinter ein Dokument befinden muß, vielmehr steht die Eindeutigkeit des URIs im Vordergrund. Allerdings kann der URI nebenbei auch einem Verweis zu einem Dokument mit weiteren Informationen, welches zum Beispiel die verwendeten XML-Elemente mit Beschreibung enthält, dienen. Namensräume sind hilfreich zur Vermeidung von Verarbeitungsproblemen von Elementen und Kollisionen. Betrachtet man XML-Dokumente für Anwendungen, dessen Inhalt für mehrere Softwaremodule definiert ist, kann es unter Umständen vorkommen, daß die verschiedenen Module gleich bezeichnete Tags verwenden. Diese doppelt benannten Elemente können zu Erkennungsproblemen führen und zwangsläufig, wegen ihrer Doppeldeutigkeit, Kollisionen hervorrufen. Die Lösung dieses Problems führt zu einer Verwendung von Namensräumen für jedes Softwaremodul, welche die Eindeutigkeit der Tagbezeichnungen wiederherstellt.

3.5.2.1 Deklaration

Namensräume können auf zwei verschiedene Weisen deklariert werden, je nach Deklaration kommt ihnen dann eine unterschiedliche Bedeutung zu. Die Deklaration erfolgt nur über reservierte Attribute in Elementen, einerseits mit dem Attributnamen `xmlns` oder andererseits über einen beliebigen Attributnamen in Kombination mit dem Präfix `xmlns:`. Der Attributname in der zweiten Variante darf hinter dem Präfix nicht

leer sein und muß mindestens ein Zeichen enthalten. Außerdem darf der Attributname nicht mit dem Wort `xml` beginnen, da alle diese Bezeichner für die Verwendung von XML oder dessen verwandten Spezifikationen reserviert sind. Als Wertzuweisung an das Attribut dient ein URI zur eindeutigen Identifizierung des Namensraums.

Ist der Attributname wie in der ersten Variante mit `xmlns` angegeben, dann legt dieser einen *voreingestellten Namensraum* im Geltungsbereich fest. Ist der Attributname dagegen wie in der zweiten Variante angegeben (*spezieller Namensraum*), so bedeutet dies eine Verknüpfung des Namensraums mit dem Attributnamen als dessen Bezeichner. Dieser kann Elementen und Attributen im Geltungsbereich zugewiesen werden, indem sein Bezeichner als Präfix in Elementen und Attributen angegeben wird. Präfixe werden in Elementen und Attributen durch einen Doppelpunkt von dem Bezeichner getrennt. Beispiel eines voreingestellten und speziellen Namensraums:

```
<Geraet xmlns="http://netman.xt/tech/ns">
  ...
</Geraet>
```

oder

```
<Geraet xmlns:nmw="http://netman.xt/tech/spec_ns">
  ...
</Geraet>
```

3.5.2.2 Geltungsbereich

Der Namensraum gilt für das Element, in dem er deklariert ist, und alle in diesem Element eingeschlossenen Elemente. Eine Ausnahme bildet eine Überschreibung des Namensraums mit gleichem Bezeichner in einem inneren Element. Ein voreingestellter Namensraum bezieht sich auf alle Elemente, denen kein Präfix für die Verknüpfung mit einem speziellen Namensraum vorangestellt ist. Die Wertzuweisung kann bei Angabe eines voreingestellten Namensraums auch leer bleiben. In diesem Fall werden die Elemente ohne Namensraumpräfix so angesehen, als wäre ihnen kein Namensraum zugewiesen. Obendrein gilt es zu beachten, daß voreingestellte Namensräume nicht direkt für Attribute in Elementen gelten, mehr dazu in Abschnitt 3.5.2.3. Die Wertzuweisung (URI) an einen speziellen Namensraum sollte gegenüber der eines voreingestellten Namensraums nicht leer sein, da die Elemente mit dem entsprechenden Präfix dann auch so angesehen werden müßten, als wäre ihnen kein Namensraum zugewiesen. Dies kann dann bei Elementen wieder zu Verarbeitungsproblemen und Kollisionen führen.

In dem folgenden Beispiel liegen die Elemente `Geraet`, `Typ`, `Seriennummer` und `Standort` im voreingestellten Namensraum des Elements `Geraet`. Die Elemente `Hersteller` und `Herstellername` liegen dagegen nicht im voreingestellten Namensraum von `Geraet`, da dieser durch einen anderen voreingestellten Namensraum überschrieben wird. Das Element `Name` liegt im speziellen Namensraum `nmw` von `Geraet`, da mit diesem eine Verknüpfung über das Präfix im Element besteht.

```
<?xml version="1.0"?>

<Geraet xmlns="http://netman.xt/tech/ns"
   xmlns:nmw="http://netman.xt/tech/spec_ns">
  <Typ>Router<Typ>
  <nmw:Name>Prestige 660HW-67</nmw:Name>
  <Hersteller xmlns="http://netman.xt/tech/xx">
    <Herstellername>Zyxel</Herstellername>
  </Hersteller>
  <Seriennummer>0064326</Seriennummer>
  <Standort/>
</Geraet>
```

3.5.2.3 Namensräume und Attribute

Wie bereits erwähnt, gelten voreingestellte Namensräume nur für Elemente, nicht aber für deren Attribute. Wie doppelt bezeichnete Elemente zu Kollisionen führen, so betrifft dies auch gleich bezeichnete Attribute eines Elements. Zur Wahrung der Eindeutigkeit müssen die Attribute ebenso mit einem speziellen Namensraum gekennzeichnet werden, da nur diese auf Attribute anwendbar sind. Die folgenden zwei Beispiele aus [BHL99] veranschaulichen das Kollisionsproblem in Zusammenhang mit je einem voreingestellten und speziellen Namensraum.

In dem ersten Beispiel haben die Attribute im Element `bad` gleiche Bezeichner oder liegen im selben Namensraum und kollidieren daher miteinander:

```
<!-- http://www.w3.org is bound to n1 and n2 -->
<x xmlns:n1="http://www.w3.org"
   xmlns:n2="http://www.w3.org">
  <bad a="1"     a="2"/>
  <bad n1:a="1"  n2:a="2"/>
</x>
```

Das zweite Beispiel zeigt im ersten Auftreten des Elements `good` zwei verschieden bezeichnete Attribute. Die Attribute des zweiten Auftretens von `good` tragen zwar den gleichen Bezeichner, liegen aber nicht im selben Namensraum. Da der voreingestellte Namensraum für das erste Attribut nicht gilt, liegt dieses Attribut deswegen in keinem Namensraum. Hier liegt keine Kollision vor:

```
<!-- http://www.w3.org is bound to n1 and is
   the default -->
<x xmlns:n1="http://www.w3.org"
   xmlns="http://www.w3.org">
  <good a="1"     b="2"/>
  <good a="1"     n1:a="2"/>
</x>
```

3.5.3 Schemasprachen

Neben dem generellen Aufbau von XML-Dokumenten fordern verarbeitende Anwendungen, das XML-Vokabular auf eine Gruppe von Elementen und Attributen einzuschränken und den Kontext zwischen mehreren Elementen und zwischen Elementen und Attributen festzulegen. D. h., Elemente dürfen nur an bestimmten Stellen im Dokument und bestimmte Attribute nur in bestimmten Elementen auftauchen. Daneben ist es sinnvoll, den Inhalt von Elementen und Attributen auf bestimmte Datentypen zu beschränken. Diese Einschränkungen sind notwendig, damit nur bestimmte Daten in einer vorgegebenen Anordnung in den XML-Dokumenten gespeichert werden. Die Einschränkung der Datentypen ist für Webservices besonders wichtig, da die Remoteprozeduren zuvor festgelegte Datentypen für Parameter und Rückgabewerte erwarten. Diese Einschränkungen werden mit einer Schemasprache in einem gesonderten Dokument formuliert, auf die vom XML-Dokument verwiesen wird. Zwei bekannte Schemasprachen sind:

- DTD (Document Type Definition) [BPSM+04]
- W3C-XML-Schema ([FW04], [TBMM04], [BM04])

Beide Sprachen weisen eine unterschiedliche Mächtigkeit auf. Das XML-Schema hat gegenüber der DTD einige Vorteile. Einerseits sind deren Schemadokumente XSD (XML Schema Definition) selbst in XML und nicht in einer eigenen Sprache verfaßt. Andererseits kann ein XML-Schemadokument die Struktur eines XML-Dokuments wesentlich strikter festlegen als eine DTD. Neben diesen beiden Sprachen existiert noch eine weitere Schemasprache:

- Relax NG [ISO02]

Diese Schemasprache ist ähnlich aufgebaut wie ein XML-Schema und verwendet in den Schemadokumenten gleichermaßen XML. Ein Dokument, welches wohlgeformt ist und die durch eine Schemasprache festgelegte Struktur aufweist, heißt *gültig*. Für eine ausführliche Beschreibung der drei Schemasprachen sei auf weiterführende Literatur verwiesen.

3.5.4 Parser

Ein Parser ist ein Syntaxprozessor für die automatische Verarbeitung von Eingabedaten. Der Parser erhält in diesem Zusammenhang ein XML-Dokument als Eingabedaten und analysiert dessen Syntax. Anhand der Bezeichner der Elemente übergibt der Parser die Daten, die durch diese Elemente eingeschlossen sind, an die dafür zuständigen Programmodule (s. Abbildung 3.2).

Für die Verarbeitung von XML-Daten gibt es zwei Parser, die grundverschiedene Techniken einsetzen:

- DOM (Document Object Model)
- SAX (Simple API for XML)

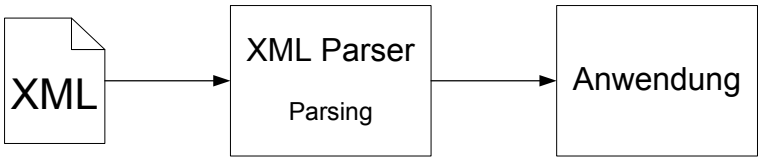

Abbildung 3.2: Parsen von XML-Dokumenten.

Beide Techniken haben ihre Vor- und Nachteile, so daß ihr Einsatz in unterschiedlichen Anwendungsgebieten liegt. Die größten Unterschiede der beiden Techniken liegen in der Verarbeitung der Daten. DOM liest zunächst das komplette Dokument ein und generiert einen Objektbaum, in dem die Beziehungen zwischen Eltern-, Geschwister- und Kindelementen erhalten bleiben. Da das ganze Dokument eingelesen wird, geht diese Technik mit einem großen Speicherverbrauch einher. SAX arbeitet dagegen ereignisbasiert. Die Programmodule können sich für diverse Ereignisse registrieren, die dann während der sequentiellen Abarbeitung der Elemente ausgelöst werden. Diese Technik hat gegenüber DOM den Vorteil eines niedrigen Speicherverbrauchs, da nie das ganze Dokument im Speicher vorgehalten werden muß. Der Nachteil äußert sich gegenüber DOM dann, wenn die für die Verarbeitung benötigten Informationen über das Dokument verstreut sind. Dies ist zeitaufwendig.

3.5.5 Validierung

Während des Parsens von XML-Dokumenten können diese bei Bedarf gegen das mit ihnen verknüpfte Schemadokument auf ihre Gültigkeit überprüft werden (s. Abbildung 3.3). Tritt während des Parsens eine Verletzung der Gültigkeit auf, so wird dies der verarbeitenden Anwendung gemeldet, die dann entsprechend reagieren muß. Da die Überprüfung der Gültigkeit das Parsen stark verlangsamen kann, ist sie nicht zwingend vorgeschrieben und kann optional hinzugezogen werden.

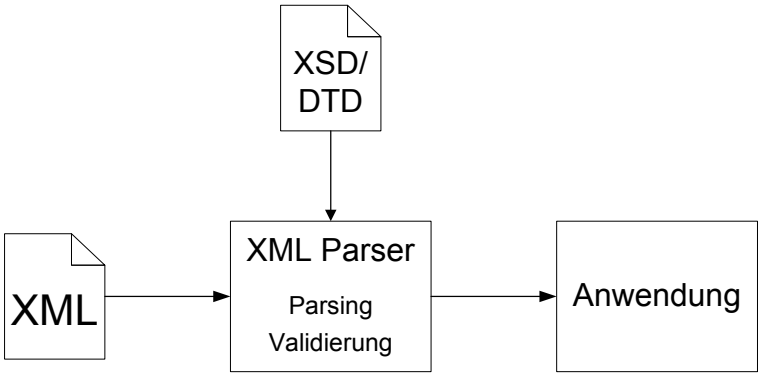

Abbildung 3.3: Parsen von XML-Dokumenten mit gleichzeitiger Validierung.

3.6 WSDL

Wenn Anwendungen und Dienste miteinander oder untereinander durch den Austausch von Daten kommunizieren wollen, müssen bestimmte Regeln eingehalten werden, damit die Kommunikation funktioniert und die Daten korrekt verarbeitet werden. Aus diesem Grund läuft die Kommunikation über vorher vereinbarte Softwareschnittstellen ab. Wollen also zwei Komponenten miteinander kommunizieren, müssen sie sich entsprechend den zugrundeliegenden Anforderungen auf eine gemeinsame Schnittstelle einigen. D. h., nimmt eine Anwendung einen Dienst in Anspruch, kann sie mit diesem nur kommunizieren, wenn sie dessen Schnittstelle kennt. Für Remoteprozeduraufrufe ist es beispielsweise notwendig zu wissen, welche Funktionen zur Verfügung stehen, und welche Ein- und Ausgabeparameter jede Funktion hat. Für die Definition von Schnittstellen existieren sogenannte *Schnittstellendefinitionssprachen (Interface Definition Languages (IDL))*. Diese Sprachen sind oftmals unabhängig von Programmiersprachen und Betriebssystemen. Dies hat einen einfachen Grund: Kommunizierende Komponenten arbeiten nicht zwangsläufig in einer homogenen Umgebung, so ist es mit diesen Schnittstellensprachen trotz Heterogenität möglich, einmalig eine gemeinsame Schnittstelle zu definieren. Die Middleware CORBA verwendet zum Beispiel auch eine gesonderte Schnittstellendefinitionssprache, die auf allen unterstützten Betriebssystemen gleich ist. Genauso müssen Webservices auch zuvor vereinbarte Schnittstellen aufweisen, damit diese von Applikationen aufgerufen werden können. Webservices verwenden dazu eine einheitliche Sprache, die *Web Service Description Language* ([BL05], [CMRW05], [CHL+05]). In einigen Punkten unterscheidet sich WSDL von den anderen Sprachen und ist zudem etwas umfangreicher. Während diese Sprachen das Ziel einer abstrakten Definition der Schnittstellen beabsichtigen und die konkrete Umsetzung unter einer Adresse der Anwendung überlassen, geht WSDL über die Definition hinaus. WSDL beschreibt neben der Schnittstellendefinition auch die Umsetzung unter einer Adresse.

WSDL liegt aktuell in der Version 2.0 als Entwurf vor, wurde von der W3C aber noch nicht abschließend standardisiert. Die Ausarbeitung ist aber schon so weit fortgeschritten, daß sich voraussichtlich keine großen Änderungen mehr ergeben werden und WSDL jetzt schon produktiv eingesetzt wird. Die Beschreibung eines Webservices mittels WSDL erfolgt in einem XML-Dokument und in zwei Phasen, einer abstrakten und einer konkreten. Im abstrakten Teil werden die Nachrichten definiert, über welche später die Dienste über Namen mit Eingabe- und Ausgabeparametern angesprochen werden. Der konkrete Teil der Beschreibung enthält Informationen über die Protokollbindung und Adresse, unter welcher der Dienst erreichbar ist. Der Austausch von Nachrichten zwischen einer Anwendung und einem Dienst bezeichnet WSDL als *Operation*. Eine Operation ist mit dem Aufruf eines Webservices vergleichbar. Mehrere Operationen werden zu einer Schnittstelle zusammengefaßt, die dann durch eine oder mehrere Bindungen mit einem Protokoll verknüpft wird [EF03].

3.6.1 Struktur der WSDL-Beschreibungen

WSDL-Beschreibungen liegen in XML-Dokumenten in einer vorgegebenen Struktur vor, die wiederum durch ein XML-Schema festgelegt ist. Der grundsätzliche Aufbau von WSDL-Dokumenten sieht folgendermaßen aus:

```
<description>
  <documentation />?
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</description>
```

Das »?« hinter den Attributen bedeutet, daß diese entweder keinmal aber höchtens einmal vorkommen dürfen. Das »*« gibt hingegen an, daß die Attribute entweder keinmal oder aber beliebig oft auftauchen dürfen.

Das Element `description` ist das Wurzelelement jeder Beschreibung und muß nebenbei noch einige Pflichtattribute enthalten. Zum einen muß das WSDL-Dokument selbst einem Namensraum zugeordnet sein, dies kann entweder über einen voreingestellten oder einen speziellen Namensraum geschehen. Zum anderen muß noch ein spezieller Namensraum für den hier definierten Webservice angegeben sein. Das Attribut `targetNamespace` muß ebenfalls enthalten sein, es dient der Identifizierung des hier definierten Webservices und hat als Wertzuweisung denselben URI wie der spezielle Namensraum des Webservices. Weitere Namensraumdeklarationen, wie beispielsweise für die Protokollbindung, sind hier auch enthalten. Ein Beispieldokument könnte dementsprechend so aussehen:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
   xmlns="http://www.w3.org/2005/08/wsdl"
   targetNamespace="http://netman.xt/2004/wsdl/ws"
   xmlns:tns="http://netman.xt/2004/wsdl/ws"
   xmlns:nmns="http://netman.xt/2004/schemas/ws"
   xmlns:wsoap="http://www.w3.org/2005/08/wsdl/soap"
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  ...
</description>
```

3.6.2 Modularisierung

WSDL bietet die Möglichkeit, Dokumente zu modularisieren. Ist eine Beschreibung etwa sehr umfangreich oder wird von mehreren Autoren entwickelt, kann es vorteilhaft sein, diese in mehrere Dokumente aufzuteilen. Ein weiteres Argument für die Modularisierung ist die Wiederverwendbarkeit einzelner Komponenten einer WSDL-Beschreibung. Für die Modularisierung bietet WSDL die XML-Elemente `import` und `include` an.

Jedes WSDL-Dokument ist über das Attribut `targetNamespace` im Wurzelelement `description` mit einem Namensraum verknüpft. Referenzieren Komponenten dieses WSDL-Dokuments auf Komponenten, die nicht in dessen Namensraum liegen, so müssen diese anderen Namensräume mit dem Element `import` eingebunden werden. Das optionale Attribut `location` kann zusätzlich das WSDL-Dokument des anderen Namensraums angeben. Beispiel:

```
<import namespace="http://netman.xt/nws/wsdl"
   location="netmanagement.wsdl"/>
```

Das Element `include` erlaubt es dagegen, Komponenten einer Webservicebeschreibung, die zu demselben Namensraum gehören, auf mehrere unabhängige Dokumente aufzuteilen. Die aufgeteilten Komponenten werden in das Dokument eingefügt, das mit dem Element `include` referenziert. Beispiel:

```
<include location="netmanagement.wsdl"/>
```

3.6.3 Spezifikation von Datentypen

Über das Element `types` unterstützt WSDL die Spezifikation von Datentypen. Datentypen können entweder direkt angegeben oder importiert werden. Für die Spezifikation sind eine Vielzahl von Schemasprachen erlaubt, allerdings definiert die Spezifikation 2.0 von WSDL nur die Benutzung der XML-Schemasprache.

3.6.4 Schnittstellendefinition

Das Element `interface` definiert, welche Operationen in einer Schnittstelle zusammengefaßt sind und welche Nachrichten die Operationen austauschen. Der folgende Ausschnitt zeigt ein Beispiel über die Definition einer Schnittstelle:

```
<interface name="netmanIntf">
  <operation name="op1" pattern="http://www.w3.org/2005/
     08/wsdl/in-out">
    <input element="tns:op1Request"/>
    <output element="tns:op1Response"/>
  </operation>
</interface>
```

3.6.5 Beschreibung von Bindungen

Das Element `binding` legt die Protokollbindung der Operationen einer Schnittstelle fest. Für eine Schnittstelle können dabei beliebig viele Bindungen definiert werden. Das folgende Beispiel zeigt das Wurzelelement einer Bindung der oben definierten Schnittstelle `netman`:

```
<binding name="netmanBinding" interface="tns:netmanIntf"
   type="http://www.w3.org/2005/08/wsdl/soap">
  ...
</binding>
```

3.6.6 Beschreibung von Diensten

Damit ein Webservice auch erreichbar ist, muß jeder Schnittstelle auch ein Endpunkt zugewiesen werden. Über den Endpunkt ist die Implementierung des Webservices für Dienstnutzer zugreifbar. Die Beschreibung erfolgt über das Element `service`. Dem folgenden Beispiel nach können Dienstnutzer den Webservice über den URI *http://netman.xt/configure* erreichen.

```
<service name="netmanConfig" interface="tns:netmanIntf">
  <endpoint name="netmanEndpoint"
     binding="tns:netmanBinding"
     address="http://netman.xt/configure"/>
  ...
</service>
```

Die WSDL-Beschreibungssprache ist wesentlich umfangreicher als hier dargestellt. Dieser Abschnitt soll nur einen kleinen Einblick in die Definition von Webservices bieten und die Funktionalität dieser Beschreibungssprache aufzeigen. Detaillierte Informationen über WSDL sind den Spezifikationen über WSDL 2.0 zu entnehmen.

3.7 SOAP

SOAP (vormals: Simple Object Access Protocol) ist ein Protokoll, welches ursprünglich für Remoteprozeduraufrufe konzipiert wurde. In der aktuellen Spezifikation 1.2 ([Mit03], [GHM+03a], [GHM+03b]) wurde das Protokoll so erweitert, daß es nicht nur Remoteprozeduraufrufe durchführt, sondern generell den Datenaustausch unter entfernten Systemen. Deswegen ist SOAP ab der Version 1.2 ein eigenständiger Name und keine Abkürzung mehr. SOAP stützt sich für die Kommunikation auf bereits standardisierte Protokolle und setzt für die Datenübertragung XML ein. Die Strukturierung der Daten in XML erfolgt nach einer festen Vorlage, hierzu definiert SOAP Namensräume und gibt Bezeichner für Elemente vor. Als Transportprotokoll verwendet SOAP üblicherweise das im Internet gebräuchlichste Protokoll *HTTP (Hypertext Transfer Protocol)*. Allerdings ist SOAP seit der Version 1.2 nicht mehr zwingend auf HTTP angewiesen und kann auch auf anderen Transportprotokollen wie *FTP (File Transfer Protocol)* oder *SMTP (Simple Mail Transfer Protocol)* aufsetzen. Die Nutzung dieser Transportprotokolle bringt Vorteile mit sich, allerdings gibt es dabei auch einige Dinge zu beachten. Beispielsweise werden die Standardprotokolle oftmals nicht durch Firewalls blockiert. Dazu muß man sich die Frage stellen, ob es sinnvoll oder gewollt ist, eine Firewall durch den Einsatz dieser Protokolle zu umgehen. Ein weiterer Punkt wäre, den Zugriff von Diensten desselben Trägerprotokolls (z. B. Webserver und

SOAP über HTTP) unterschiedlich zu regeln. Eine gute Übersicht für diese und weitere Punkte bezüglich HTTP bietet [Moo02]. Ein weiterer Vorteil ist, daß der Datenaustausch über SOAP, aufgrund der weit verbreiteten Transportprotokolle und Verwendung von XML, nahezu betriebssystem- und architekturunabhängig ist. SOAP kann so ohne Bedenken in heterogenen Umgebungen eingesetzt werden ([SS01], [HL04]).

3.7.1 Kommunikationsformen

SOAP unterstützt bei der Nachrichtenübertragung verschiedene Kommunikationsformen. Die Abbildung 3.4 stellt die Kommunikation in einem Frage-Antwort-Ablauf dar, d. h., schickt der Sender eine Anfrage an einen Empfänger, dann schickt dieser daraufhin eine Antwort an den Sender zurück. Diese Kommunikationsweise findet bei den bereits erwähnten Remoteprozeduraufrufen Anwendung. Die Art der Kommunikation wird auch als *synchrone Kommunikation* bezeichnet, da auf eine Anfrage immer sofort eine Antwort erfolgt. Die Abbildung 3.5 zeigt dagegen eine *asynchrone Kommunikation*. Diese Art der Kommunikation wird eingesetzt, wenn ein Sender ohne vorherige Anfrage Nachrichten verschicken will. Bei der asynchronen Kommunikation können die Nachrichten auch gleichzeitig an mehrere Empfänger versendet werden, vergleichbar mit einer *Punkt-zu-Mehrpunktverbindung (Multicast)* (s. Abbildung 3.6) [HL04].

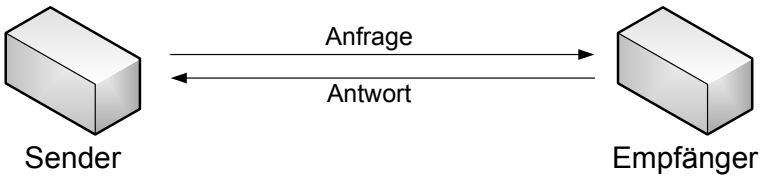

Abbildung 3.4: Synchrone Kommunikation.

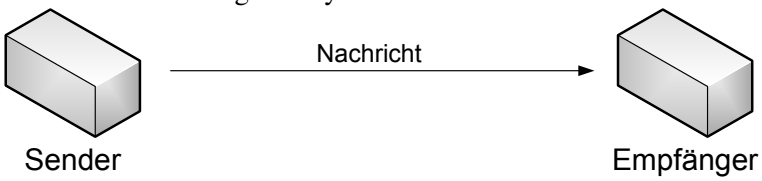

Abbildung 3.5: Asynchrone Kommunikation.

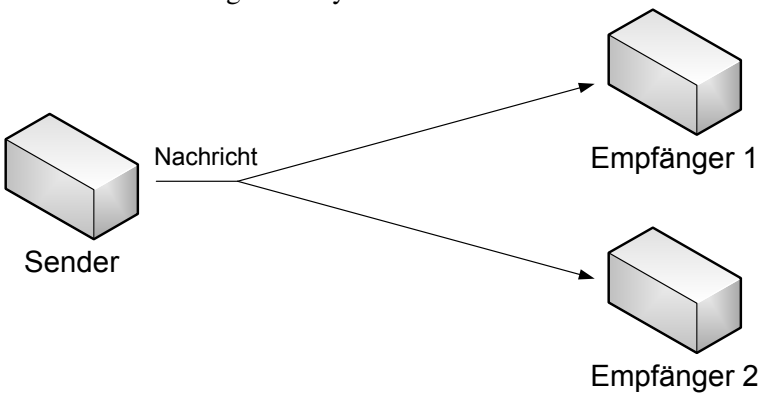

Abbildung 3.6: Asynchrone Punkt-zu-Mehrpunkt-Kommunikation.

3.7.2 Nachrichten

SOAP-Nachrichten werden als Nutzdaten des Transportprotokolls in einer Art Umschlag, dem *SOAP-Envelope* (s. Abbildung 3.7) verschickt. Die Daten in diesem Umschlag bestehen aus zwei Teilen, dem *Kopf (Header)* und dem *Rumpf (Body)*. Der Kopf ist in einem SOAP-Umschlag optional, muß aber im Fall des Vorhandenseins unbedingt vor dem Rumpf auftauchen. Der Kopf kann Metainformationen für den Transport der Nachrichten oder den Methodenaufruf enthalten. Diese können beispielsweise Informationen über die Zugehörigkeit der Nachricht zu einer Transaktion, Routinginformationen o. ä. sein. Der Rumpf muß in dem Umschlag immer vorhanden sein und beinhaltet die eigentlich zu übertragene XML-Nachricht [EF03].

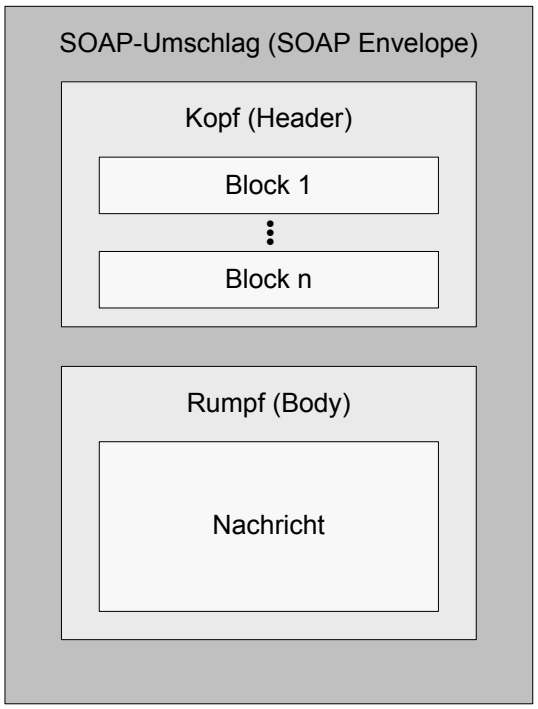

Abbildung 3.7: Aufbau einer SOAP-Nachricht [EF03].

3.7.2.1 Umschlag (Envelope)

Das Umschlagelement trägt als Bezeichner immer den Namen `Envelope` und ist immer mit einem Namensraum verknüpft. Der Namensraum für den Umschlag ist in der Spezifikation fest vorgegeben und unterscheidet sich nur durch die verwendete Version der Spezifikation. SOAP 1.1 definiert seinen Namensraum durch den URI `http://schemas.xmlsoap.org/soap/envelope`, während die Version 1.2 den URI `http://www.w3.org/2003/05/soap-envelope` verwendet. Anhand der verschiedenen URI lassen sich die beiden Versionen 1.1 und 1.2 in Nachrichten voneinander unterscheiden. Wichtig ist, daß das Umschlagelement und die direkt nachfolgenden Kindelemente (Kopf und Rumpf) mit dem o. g. Namensraum verknüpft sind.

Die inneren Elemente des Kopfes und Rumpfes, welche Anwendungsdaten betreffen, sollten zwecks der Eindeutigkeit mit einem eigenen Namensraum versehen werden. Eine einfache Darstellung eines Umschlages kann in XML-Notation folgendermaßen aussehen:

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/
   05/soap-envelope">
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

Weiterhin kann ein Umschlagelement zusätzliche Namensraumdeklarationen und Attribute enthalten. Attribute müssen allerdings durch einen Namensraum qualifiziert sein.

Neben den o. g. Informationen kann der SOAP-Umschlag auch noch ein weiteres Attribut `encodingStyle` enthalten. Der Versender kann mit Hilfe dieses Attributs festlegen, wie die übertragenen Daten einer Nachricht zu verarbeiten sind. Dies ist enorm wichtig, da es ansonsten passieren kann, daß der Empfänger die Daten falsch verarbeitet und dann, je nach Situation, ebenfalls ein falsches Ergebnis zurückliefert. Ein einfacher Fall eines solchen Problems könnte so aussehen:

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/
   05/soap-envelope">
  <env:Body>
    <ns:setSerial xmlns:ns="http://comp.xt/schema">
      <ns:number>05245236</ns:number>
    </ns:setSerial>
  </env:Body>
</env:Envelope>
```

In diesem Beispiel möchte eine Anwendung die entfernte Prozedur `setSerial` mit einem zusätzlichen Parameter `number` aufrufen. Die Senderseite geht zum Beispiel davon aus, daß es sich bei dem Parameter um einen Datentyp *Integer* handelt. Da der Datentyp des Parameters in der SOAP-Nachricht aber nicht enthalten ist, kann der Empfänger den Datentyp mißinterpretieren. Deutet der Empfänger den Parameter nicht wie vorgegeben als Integer sondern als *String*, wird die Prozedur nicht korrekt ausgeführt. Mit dem Attribut `encodingStyle` können die Datentypen als Attribute in den Elementen angegeben werden, so daß diese vom Datentyp nicht mehr falsch interpretiert werden können. Ab SOAP-Version 1.2 ist das Attribut `encodingStyle` optional, da sich XML-Schemata mittlerweile so weit entwickelt haben, daß die Datentypen einzelner Elemente auch dort definiert werden können ([HL04], [GHM+03a]).

3.7.2.2 Kopf (Header)

Das Kopfelement innerhalb des Umschlages hat immer den Bezeichner `Header` und liegt in demselben Namensraum wie das Umschlagelement. D. h., je nach SOAP-Version gibt es auch hier zwei verschiedene URI für den Namensraum. Die genauen Bezeichnungen der URI sind im vorherigen Abschnitt 3.7.2.1 zu finden. Ansonsten kann der Kopf noch mehrere namensraumqualifizierte Attribute und Kindelemente (s. Abbildung 3.7: Block *1* bis *n*) enthalten.

Ein Blockelement muß einen Bezeichner haben und durch einen Namensraum qualifiziert sein. Dieser Namensraum ist von der Spezifikation nicht vorgegeben und kann beliebig gewählt werden. Außerdem kann der Block weitere für SOAP spezifische Attribute (z. B. `encodingStyle`) und Daten, die durch weitere Kindelemente strukturiert sein können, enthalten [GHM+03a].

3.7.2.3 Rumpf (Body)

Der Rumpf ist analog zum Kopf aufgebaut. Der Bezeichner des Elements lautet `Body` und liegt in demselben Namensraum wie das Umschlagelement. Weiter kann dieses Element noch durch einen Namensraum qualifizierte Attribute und Kindelemente enthalten.

Die direkt im Rumpf folgenden Kindelemente können wieder (auch für SOAP spezifische) Attribute enthalten und sollten mit einem Namensraum verknüpft sein. Ferner können Daten enthalten sein, die gegebenenfalls durch weitere Elemente strukturiert sind [GHM+03a].

Diese Informationen geben nur einen kleinen Überblick über den Aufbau von SOAP-Nachrichten. Weiterführende Informationen und Details hierzu sind den Spezifikationen und Zusatzliteratur zu entnehmen.

3.8 UDDI

Wie in Kapitel 3.1 bereits erwähnt, finden Webservices oftmals in Unternehmen Anwendung. Unternehmen können Daten für Geschäftsprozesse außerhalb oder innerhalb des Unternehmens über Webservices verfügbar machen. Die Frage ist aber, wie solche angebotenen Webservices aufgefunden werden können. Am einfachsten ist es, Informationen über mehrere Webservices an einer zentralen Stelle zu verwalten. Genau das erledigt UDDI [CHvRR04].

3.8.1 Verzeichnisdienst

UDDI spezifiziert einen Verzeichnisdienst, der speziell dafür entwickelt wurde, Dienstanbieter und Dienstnutzer von Webservices zusammenzubringen. Dienstanbieter tragen Informationen über angebotene Webservices in UDDI ein. Dienstnutzer rufen die

Informationen ab und kommunizieren dann mit dem Webservice (s. Abbildung 3.8). Genau genommen bestimmt UDDI nicht nur einen Verzeichnisdienst, sondern jeder kann einen Verzeichnisdienst auf Basis von UDDI aufsetzen. Die Verzeichnisdienste werden auch als *Registries* bezeichnet und unterscheiden sich in zwei Arten, den öffentlichen und privaten. Öffentliche Verzeichnisdienste beinhalten Webservices für Dienstnutzer außerhalb eines Unternehmens. Jeder kann in öffentlichen Verzeichnisdiensten nach Webservices suchen. Die privaten Verzeichnisdienste werden meistens in Unternehmen eingesetzt, da nicht jeder darauf zugreifen darf. Ein Unternehmen könnte einen privaten Verzeichnisdienst etwa im Intranet einsetzen und den Zugriff nur innerhalb des Unternehmens gestatten.

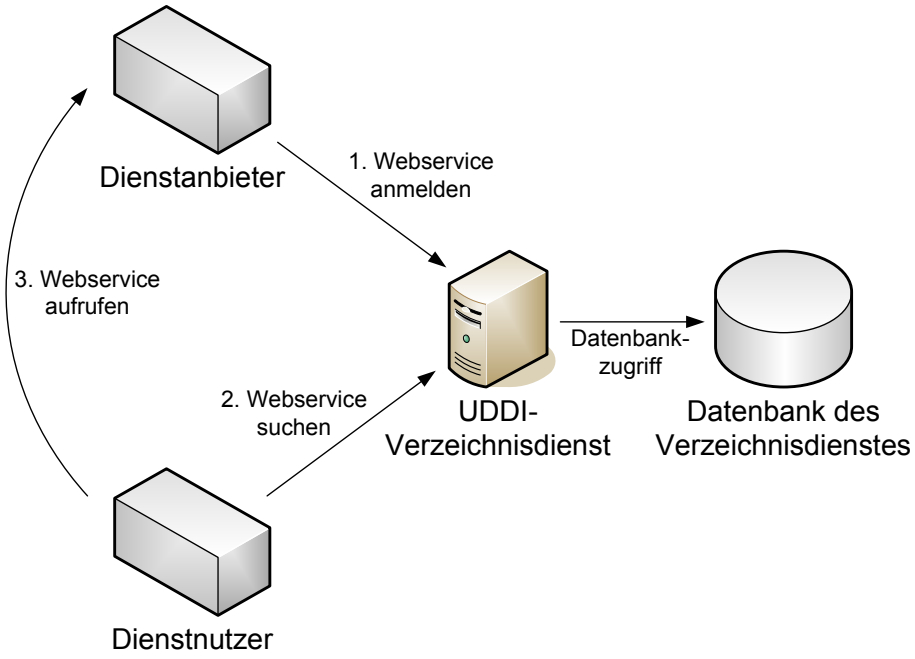

Abbildung 3.8: Kommunikation mit einem UDDI-Verzeichnisdienst.

3.8.2 Verzeichnisdienste im Verbund

Mehrere UDDI-Verzeichnisdienste können zu einem Verbund, den *UDDI Business Registries*, zusammengeschlossen werden. Alle beteiligten Verzeichnisdienste synchronisieren sich untereinander so, daß jeder den ganzen Datenbestand hat. Findet an einer Stelle ein Neueintrag oder eine Aktualisierung eines Eintrags statt, dann wird die Änderung auch an alle anderen, an dem Verbund beteiligten, Verzeichnisdienste weitergeleitet. Der Vorteil liegt in einer effizienten Lastverteilung, da Datenzugriffe auf die Teilnehmer verteilt sind.

3.8.3 Typen von Einträgen

Je nach Anwendungszweck gibt es verschiedene Arten von Verzeichnissen bzw. verschiedene Arten von darin enthaltenen Einträgen. Auch für UDDI gibt es verschiedene Typen von Verzeichniseinträgen. Ein guter Vergleich besteht dazu in Telefonbüchern, die es auch in mehreren Ausführungen (gewöhnliches Telefonbuch, Gelbe Seiten) gibt. UDDI-Einträge sind in drei Kategorien einteilbar:

- Die *White Pages* sind mit einem gewöhnlichen Telefonbuch vergleichbar und enthalten nur grundlegende Informationen über Firmen (z. B. Adresse, Telefonnummer, Beschreibung usw.), die sich bei einem UDDI-Verzeichnisdienst registriert haben.
- Die *Yellow Pages* entsprechen den Gelben Seiten und sind in Branchen gegliedert. In dieser Kategorie kann der Dienstnutzer nach einer genauen Art eines Dienstes suchen und bekommt die zutreffenden Dienstanbieter zurückgeliefert.
- Die *Green Pages* sind die wichtigste Kategorie für Webservices, sie enthält Servicebeschreibungen, Informationen über die Bindung, Verweise auf WSDL-Definitionen und weitere Details für den Zugriff auf Webservices.

3.8.4 Zugriffsverfahren

UDDI stellt im allgemeinen zwei verschiedene Verfahren für den Zugriff zur Verfügung. Bei dem einen Verfahren handelt es sich um einen Webservice. Dieser ist für den maschinellen Zugriff durch Anwendungen da, die nach Webservices suchen oder Einträge in dem Verzeichnisdienst verwalten möchten. Das andere Verfahren erlaubt den Zugriff über einen Internetbrowser und ist für den Anwender gedacht. Der Anwender kann die wichtigsten Funktionen (Suche, Verwaltung von Einträgen usw.) über den Browser steuern [EF03].

3.8.5 Verwendung von UDDI in dieser Arbeit

Die Verwendung eines UDDI-Verzeichnisdienstes ist in dieser Arbeit nicht unbedingt notwendig, da die Webserviceschnittstellen von dem Manager und Agenten im weiteren Verlauf dieser Arbeit genau spezifiziert werden. Da ein Verzeichnisdienst aber auch zum Auffinden von Webservices genutzt wird, könnte ein Manager darüber die aktiven Agenten auffinden.

4 Entwurf eines agentenbasierten Monitoringsystems

Dieses Kapitel beschäftigt sich mit dem Entwurf eines Systems für das agentenbasierte Monitoring (Überwachung) von Netzressourcen. Zunächst erläutert dieses Kapitel grundlegende Anforderungen und stellt anschließend mehrere Ansätze für die Realisierung dieser Aufgabe mit Hilfe von Webservices vor. Eine darauffolgende Bewertung der Ansätze untereinander entscheidet, welcher die besten Voraussetzungen für die Umsetzung beinhaltet. Anschließend folgt eine Erläuterung möglicher Benachrichtigungsmechanismen und Kommunikationsprotokolle. Der letzte Abschnitt des Kapitels stellt in einem Vergleich die Monitoringmöglichkeiten mittels SNMP und Webservices gegenüber.

4.1 Einführung

Das agentenbasierte Monitoring von Netzressourcen auf Basis von Webservices verfolgt wie u. a. auch SNMP das Ziel, Ressourcen innerhalb eines Netzwerkes zu überwachen. Der hier vorgestellte Entwurf unterteilt das System, wie auch SNMP, in die beiden Komponenten *Manager* und *Agenten*. Allerdings läuft das in dieser Arbeit vorgestellte agentenbasierte Monitoring auf einer anderen Art und Weise ab, als dies bei SNMP der Fall ist. Während Manager das Monitoring bei SNMP größtenteils selbst durchführen und Agenten die Monitoringinformationen den Managern in der Regel nur auf Anfrage zur Verfügung stellen, geht dieses System den umgekehrten Weg, der auch bei SNMP-Traps bzw. -Notifications zum Einsatz kommt. Manager können Agenten beauftragen, Netzwerkressourcen zu überwachen. Der Agent stellt hierfür unterschiedliche Monitoringfunktionen zur Verfügung, die er mit einem Monitoringauftrag ausführt. Agenten sammeln Informationen und überwachen Ressourcen selbständig. Die daraus resultierenden Benachrichtigungen an Manager führen Agenten eigenständig unter Berücksichtigung zuvor festgelegter Regeln aus. Manager können die Benachrichtigungen auswerten und falls notwendig, je nach Nachricht, weitere Maßnahmen ergreifen. Dieses System versucht demzufolge, den normalerweise in Managern eingegliederten aktiven Monitoringprozeß auf die Agenten auszulagern. Die Manager sollen aber trotz der Auslagerung weiterhin die Steuerung des Monitorings übernehmen.

Die Agenten sind im webservicebasierten Konzept in gleicher Weise wie bei SNMP mit zu überwachenden Netzwerkressourcen verbunden. Sie können wahlweise direkt in Geräten oder Diensten integriert oder auch außerhalb vorhanden sein und als Proxy-Agenten fungieren.

4.2 Monitoringablauf

Bevor ein Agent Benachrichtigungen an einen Manager schicken kann, muß er zunächst einmal wissen, welchen Manager er mit welchen Informationen versorgen soll. Zudem muß er auch noch wissen, unter welchen Umständen er die Informationen zustellen soll. Die Manager müssen daher selbst bestimmen, welche Mitteilungen ihnen die Agenten zur Verfügung stellen sollen, da es den Agenten nicht gestattet ist, willkürlich irgendwelche Informationen zu verschicken. Dies hat den Grund, daß das Netzwerk nur so gering wie möglich belastet werden soll und Agenten keine Meldungen verschicken, die von Managern nicht von Interesse sind. Wünscht ein Manager, von einem Agenten Monitoringmeldungen zugestellt zu bekommen, so startet er auf dem betreffenden Agenten zunächst einen Konfigurationsaufruf. In dem Konfigurationsaufruf teilt der Manager einem Agenten in Monitoringaufträgen mit, welche Monitoringmaßnahmen er im Interesse des Managers durchführen soll. Über diese Monitoringaufträge verlangt der Manager Mitteilungen seitens des Agenten. Hat ein Agent für einen angemeldeten Monitoringauftrag die erforderlichen Daten gesammelt, oder tritt eine Situation ein, die gerade überwacht wird, dann informiert der Agent den Manager umgehend in Form einer Benachrichtigung. Den generellen Ablauf zwischen Anmeldung und Benachrichtigung zeigt die Abbildung 4.1.

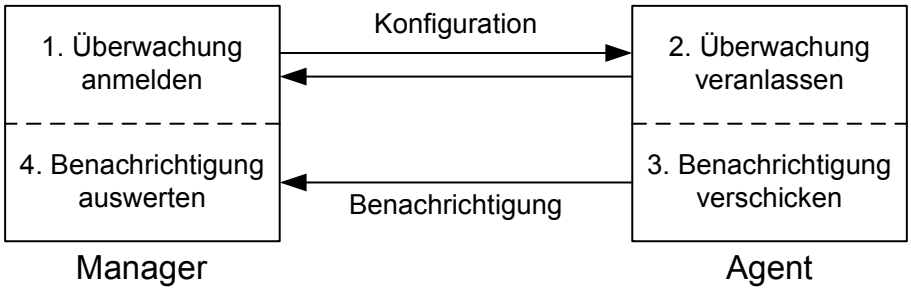

Abbildung 4.1: Ablauf des Monitorings in vier Schritten.

4.3 Monitoringfunktionen

Für das Monitoring von Ressourcen stellen Agenten Funktionen bereit, die unterschiedliche Ressourcenparameter überwachen und von Managern in Anspruch genommen werden können. Diese Funktionen werden als *Monitoringfunktionen* bezeichnet. Die Monitoringfunktionen in Agenten lassen sich grob in zwei Gruppen einordnen. Die eine Gruppe beinhaltet Funktionen, mit denen die Agenten allgemein Informationen über Ressourcen sammeln. Die zweite Gruppe enthält Funktionen, mit denen die Agenten Ressourcen auf bestimmte Ereignisse hin überwachen können, die implizit oder explizit mit Ereignisbedingungen verknüpft sind.

Funktionen der ersten Gruppe haben überwiegend einen statistischen Charakter. Sie können zum Beispiel Daten einer oder mehrerer miteinander verknüpfter Datenquellen über einen festgelegten Zeitraum protokollieren. Es ist auch denkbar, daß Funktionen

in dieser Gruppe Berechnungen über Daten erstellen und Managern nur das Ergebnis übermitteln. Die in der zweiten Gruppe enthaltenen Funktionen stellen auf dem Agenten Vergleichsoperationen mit den zuvor festgelegten Ereignisbedingungen an.

4.3.1 Ereignis

Ein Ereignis bezeichnet das Eintreten eines bestimmten Zustandes auf einem Agenten, welcher für einen Manager von Interesse sein kann. Die Ereignisse können dabei unterschiedlicher Natur sein, zum Beispiel Fehler, Statuswechsel, Überschreitung bzw. Unterschreitung von Grenzwerten, Konfigurationsänderungen, externe Eingaben, Statistikwerte u. a.

4.3.2 Nachrichtenklassifizierung

Benachrichtigungen lassen sich entsprechend ihrer Bedeutung in Klassen unterteilen. Dies hat den Vorteil, daß ein Manager den eingehenden Nachrichten schon anhand der Klasse eine Priorität zuweisen kann. In einem Netzwerk ist es sehr wichtig, daß Nachrichten, die einen Fehler auf einer Netzwerkressource signalisieren, gegenüber anderen Nachrichten bevorzugt abgearbeitet werden. In Anlehnung an die Definition der Ereignisklassen für Ereignisnachrichten im NETCONF-Protokoll [CC05] könnten Nachrichten den Klassen *Fehler*, *Information*, *Zustandswechsel*, *Konfiguration* und *periodische Benachrichtigung* angehören. Andere oder zusätzliche Klassen sind denkbar, sofern sie eine Benachrichtigung treffend beschreiben. Jedoch ist es vorteilhaft, die Anzahl unterschiedlicher Klassen möglichst gering zu halten, da Manager alle Klassenbezeichner kennen müssen, damit sie diese vor der weiteren Verarbeitung der Nachrichten auswerten können. Genauso sollten die wenigen Klassen so beschrieben sein, daß sie ein möglichst großes Spektrum an Nachrichten abdecken.

4.3.3 Parameter

Je nach Einsatzzweck oder Nachrichtenklasse (s. Abschnitt 4.3.2) benötigen Monitoringfunktionen eventuell zusätzliche Parameter. Diese legen entweder Optionen für die Ausführung fest oder definieren Bedingungen, unter welchen Umständen Agenten Ereignisse auslösen und Manager benachrichtigen müssen. Für jede Monitoringfunktion muß zunächst genau festgelegt werden, welche Datenwerte einer Ressource sie überwachen soll. Primär legt dies die Monitoringfunktion selbst fest. Aber möglicherweise existieren für einige Monitoringfunktionen mehrere entsprechende Quellen mit Datenwerten, da eine Ressource unter Umständen über mehrere Instanzen einer Teilressource verfügen kann. Hier muß dann ein Parameter die Instanz der Datenquelle kennzeichnen, die überwacht werden soll. Eine Beispielanwendung wäre, die Prozessortemperatur in einem Mehrprozessorsystem zu überwachen. Dort muß festgelegt werden, welcher Prozessor überwacht werden soll. Genauso könnten aber auch andere oder noch weitere Parameter erforderlich sein. Bei Statistikfunktionen könnten neben

Nachrichtenklasse	Monitoringfunktion
Fehler	Benachrichtigung bei physikalischer Unterbrechung der Netzwerkverbindung an Schnittstelle *x*
Statuswechsel	Benachrichtigung bei Aktivierung der Netzwerkschnittstelle *y*
Fehler	Benachrichtigung bei Überschreitung der Prozessortemperatur über 60 °C
Information	Benachrichtigung bei Unterschreitung des freien Speicherplatzes unter 100 MB

Tabelle 4.1: Beispiele für Ereignisüberwachungen.

Nachrichtenklasse	Monitoringfunktion
Information	Protokollierung der Netzwerkauslastung über einen Zeitraum von zwei Stunden im Minutenintervall
Information	Protokollierung von Übertragungsfehlern in Verbindung mit der Netzwerkauslastung über einen Zeitraum von fünf Tagen im Stundenintervall

Tabelle 4.2: Beispiele für statistische Monitoringfunktionen.

den o. g. wichtigen Parametern für die zu überwachenden Ressourcen zusätzliche Parameter das Aufzeichnungsintervall und die Anzahl der aufzuzeichnenden Datensätze festlegen. Bei Ereignissen könnten dies neben den essentiellen Parametern die Ereignisbedingungen sein. Die Anzahl der Parameter kann auch davon abhängen, in welche Benachrichtigungsklasse eine Funktion bzw. dessen Benachrichtigung fällt. Manche Fehlerüberwachungen kommen womöglich ganz ohne zusätzliche Parameter aus, falls ein Fehler so schwerwiegend ist, daß sein Auftreten an sich schon bedingt, gemeldet zu werden. Andere Monitoringfunktionen für die Fehlerüberwachung verlangen wiederum zusätzliche Bedingungen. Sollen Funktionen beispielsweise Zustandswechsel überwachen, so benötigen sie dagegen Parameter, die einen Zustandsübergang beschreiben. Vergleichbares gilt für andere Funktionen und Nachrichtenklassen.

4.3.4 Beispiele für definierte Monitoringfunktionen

Dieser Abschnitt zeigt ein paar Beispiele für Monitoringfunktionen und gibt einen kleinen Überblick über den Aufbau. Tabelle 4.1 zeigt ereignisbasierte Monitoringfunktionen auf einem Agenten. Neben diesen einfachen Ereignisüberwachungen können für das Monitoring von Netzressourcen auch beliebig komplexe Abläufe implementiert werden, über die sich Manager informieren lassen wollen. Tabelle 4.2 zeigt zwei Beispiele für komplexere Monitoringfunktionen.

4.4 Konfiguration

Für die Konfiguration der Monitoringfunktionen muß ein Agent einem Manager unabhängig von der genauen Implementierung drei grundlegende Funktionen bereitstellen. Ein Manager muß zum einen Monitoringaufträge registrieren und dessen Eigenschaften und Bedingungen für die Benachrichtigung festlegen können. Zum anderen muß er bereits abonnierte Monitoringaufträge auch wieder abbestellen können, falls seinerseits kein Interesse mehr an einer Benachrichtigung besteht. Ferner muß ein Agent als dritte Funktion die Umkonfiguration der Eigenschaften angemeldeter Monitoringaufträge erlauben. Auch wenn die dritte Funktion auf den ersten Blick nicht notwendig erscheint, verhindert sie, daß Benachrichtigungen ausbleiben, die in die Umkonfigurationsphase fallen. Ohne die dritte Funktion müßte eine Umkonfiguration über eine Abmeldung mit anschließender Neuanmeldung durchgeführt werden. Dies birgt das Risiko, daß Benachrichtigungen, die in den Zeitraum zwischen Ab- und Anmeldung fallen, verloren gehen.

4.4.1 Ansätze für einen Konfigurationsmechanismus

Die Steuerung der Monitoringfunktionen und Benachrichtigungen läßt sich in vielfältiger Weise umsetzen. Daher stellt dieser Abschnitt zunächst drei mögliche Konfigurationsansätze mit samt ihren Vor- und Nachteilen vor. Ein anschließender Vergleich der Ansätze untereinander bestimmt, welcher der drei Ansätze im nachfolgenden Kapitel genau spezifiziert und anschließend in Kapitel 6 implementiert wird.

4.4.1.1 Konfigurationsansatz 1

Der erste Konfigurationsansatz sieht vor, für alle Monitoringfunktionen (z. B. Prozessortemperatur oder Datendurchsatz auf einer Netzwerkschnittstelle) eine eigenständige Prozedur im Webservice zur Verfügung zu stellen. Der erste Parameter jeder Prozedur bestimmt die auszuführende Verwaltungsfunktion für die Konfiguration. Dabei handelt es sich um die drei möglichen Funktionen, welche die Anmeldung, Abmeldung und Umkonfiguration durchführen. Weitere Parameter adressieren, falls wesentlich, weitere Optionen der Monitoringfunktionen oder geben Bedingungen für die Ereignisauslösung, wie z. B. Grenz- oder Referenzwerte, an. Der Prozedurrückgabewert gibt darüber Aufschluß, ob die ausgewählte Funktion ordnungsgemäß ausgeführt wurde. Im Erfolgsfall einer Anmeldung bezeichnet er ein gültiges Handle zur eindeutigen Identifizierung der angemeldeten Überwachung. Soll beispielsweise ein Agent einen Manager benachrichtigen, sobald die Prozessortemperatur über 60 °C steigt, dann könnte der Prozeduraufruf für die Anmeldung folgendermaßen aussehen:

```
HANDLE cpu_temp("subscribe", "cpu0", 60);
```

Vorteile Jede Monitoringfunktion ist in einem eigenen Prozeduraufruf gekapselt. Für den Anwender ist das sehr übersichtlich.

Nachteile Jede Monitoringfunktion muß als eigene Prozedur implementiert werden. Zudem ist die Implementierung bei vielen Monitoringfunktionen sehr aufwendig. Des weiteren ist aus logischer Sicht die Reihenfolge von der Verwaltungsfunktion (Konfigurationsaufruf) und der Monitoringfunktion (eigentlich ein Parameter des Konfigurationsprozesses) vertauscht. Die eigentliche Funktion wird zum Prozedurparameter und der eigentliche Funktionsparameter bezeichnet die Prozedur. Die erforderliche Parameteranzahl innerhalb derselben Prozedur ist von der ausgewählten Monitoringfunktion abhängig.

4.4.1.2 Konfigurationsansatz 2

Der zweite Konfigurationsansatz basiert auf einer zentralen Tabelle und stellt zwei Verwaltungsfunktionen für das Auslesen und Schreiben der Tabelle auf einem Agenten bereit. Jede Monitoringfunktion entspricht einer Zeile in der Tabelle. In den Spalten befinden sich dagegen die notwendigen Parameter. Alle Monitoringfunktionen sind also mit allen möglichen Parametern (Optionen, Ereignisbedingungen usw.) verknüpft. Der Prozedurrückgabewert gibt an, ob die Tabelle korrekt ausgelesen oder geschrieben wurde. Die Funktionen zum Auslesen und Schreiben der Tabelle könnten so aussehen:

```
int getMonitoringTable(MONITORINGTABLE);

int setMonitoringTable(MONITORINGTABLE);
```

Vorteile Beschränkt sich das Monitoringsystem nur auf Ereignisüberwachungen mit nicht allzu vielen unterschiedlichen Bedingungen, dann hat die Tabelle gegenüber den anderen Konfigurationsansätzen den Vorteil, daß mehrere Ereignisse in einem Schritt über eine einzige Tabelle konfiguriert werden können.

Nachteile Ein gravierender Nachteil ist, daß es fast unmöglich ist, (komplexe) Monitoringfunktionen der ersten Gruppe (s. Abschnitt 4.3) über die Tabelle zu konfigurieren. Erschwerend kommt hinzu, daß die Parameter auf die einzelnen Funktionen zugeschnitten sind und nicht wie bei Ereignisüberwachungen mehrfach verwendet werden. Ansonsten hat die Tabelle noch einige weitere Nachteile. Bei vielen zur Verfügung stehenden Monitoringfunktionen kann die Tabelle schnell sehr umfangreich und unübersichtlich werden. Daneben muß auch für kleine Änderungen immer die gesamte Tabelle ausgelesen und geschrieben werden. Außerdem entstehen in der Tabelle viele ungültige Kombinationen zwischen Monitoringfunktionen (Zeilen) und Parametern (Spalten), die Tabelle müßte deswegen bei jeder Änderung einer aufwendigen Gültigkeitsprüfung unterzogen werden. Werden mehrere Parameter gleichzeitig geändert, so kann dies bei ungültigen Einträgen (Verknüpfungen zwischen Monitoringfunktionen und Parametern) problematisch sein, da die Konfiguration aller Änderungen in solch einem Fall nicht ordnungsgemäß abgeschlossen werden kann. Die Tabelle kann außerdem dynamisch ihre Größe ändern, falls Monitoringfunktionen zwischenzeitlich

hinzukommen oder wegfallen. Das ist beim Auslesen und Zurückschreiben zu berücksichtigen. Des weiteren lassen sich in der Tabelle keine unterschiedlichen Instanzen einer Monitoringfunktion verwalten.

4.4.1.3 Konfigurationsansatz 3

Der letzte hier vorgestellte Konfigurationsansatz ist ähnlich wie der erste aufgebaut. Dieser Ansatz stellt aber nicht je eine Prozedur für jede Monitoringfunktion bereit, sondern für jede Verwaltungsfunktion eine. Das sind demnach drei Prozeduren, jeweils eine für die Anmeldung, Abmeldung und Umkonfiguration. Der erste Parameter gibt hier die Monitoringfunktion an, die folgenden Parameter dessen erforderlichen Optionen oder Ereignisbedingungen. Die weiteren Parameter im ersten und dritten Ansatz unterscheiden sich daher nicht. Der Prozedurrückgabewert gibt an, ob die aufgerufene Prozedur erfolgreich ausgeführt wurde. Bei der Anmeldeprozedur bezeichnet der Rückgabewert im Erfolgsfall ein gültiges Handle. Eine Anmeldung für das bereits oben verwendete Ereignis, daß der Manager eine Benachrichtigung erhält, sofern die Prozessortemperatur über 60 °C steigt, könnte so aussehen:

```
HANDLE subscribe("cpu_temp", "cpu0", 60);
```

Vorteile Die gesamte Konfiguration wird über drei verschiedene Prozeduren im Webservice gesteuert. Je eine Verwaltungsfunktion wird auf eine Prozedur abgebildet. Die Konfiguration bleibt daher auch bei vielen Monitoringfunktionen und Instanzen übersichtlich. Im Vergleich zum ersten Ansatz ist aus logischer Sicht die Reihenfolge der Verwaltungsfunktionen und der Monitoringfunktionen nicht vertauscht.

Nachteile Die erforderliche Parameteranzahl innerhalb derselben Prozedur ist von der Monitoringfunktion abhängig.

4.4.2 Bewertung der Konfigurationsansätze

Eine einfache und übersichtliche Verwaltung der Monitoringfunktionen und -instanzen eines Agenten erfordert einen klar strukturierten Zugriff auf dessen Konfiguration. Der zweite Konfigurationsansatz bietet mit seiner zentralen Tabelle zwar eine Möglichkeit mehrere Monitoringfunktionen in einem Durchgang zu konfigurieren, allerdings stehen diesem Ansatz zu viele Nachteile gegenüber. Auch für kleine Änderungen die komplette Tabelle auszulesen, führt bei großen Tabellen zu einer erheblichen Belastung des Netzwerkes, die andere Konfigurationsansätze vermeiden können. Die beiden Konfigurationsansätze eins und drei haben einen sehr ähnlichen Aufbau, jedoch bietet der dritte Ansatz gegenüber dem ersten mit der Abbildung der drei Verwaltungsfunktionen auf je eine Webserviceprozedur eine komfortablere Konfigurationsmöglichkeit. Die Prozeduranzahl orientiert sich dort nicht an der Anzahl der Monitoringfunktionen, und die logische Reihenfolge von Verwaltungs- und Monitoringfunk-

tionen ist nicht vertauscht. Daher eignet sich der dritte Konfigurationsansatz am besten für die Verwaltung der Monitoringfunktionen und -instanzen auf einem Agenten.

4.5 Benachrichtigung

Damit ein Agent einem Manager Benachrichtigungen zustellen kann, muß dieser einen lokalen Endpunkt als asynchrone Callback-Schnittstelle besitzen. Diese nimmt eingehende Nachrichten von Agenten in Empfang und verarbeitet diese weiter. Je nach Inhalt der Nachrichten kann ein Manager, abhängig von seiner Implementierung, selbständig Aktionen ausführen. Die Aktionen könnten beispielsweise aufgetretene Fehler beheben, drohende Fehler vermeiden oder allgemein die Ursache eines Ereignisses ergründen. Ferner könnten Aktionen auch so aussehen, daß ein Manager Statistiken auf Datenträgern persistent macht, Warnmeldungen an Administratoren verschickt o. ä. Maßnahmen durchführt.

4.6 Kommunikation

Wie bereits im Kapitel 3.7 kurz erwähnt, verwendete SOAP ursprünglich nur HTTP als Übertragungsprotokoll. In neueren Versionen kann es aber auch andere Protokolle einsetzen. Da SOAP von sich aus unabhängig von Transportprotokollen ist, können Manager und Agenten für ihre gegenseitige Kommunikation prinzipiell jedes Transportprotokoll einsetzen, welches die Daten zuverlässig überträgt. Allerdings sind nicht alle diese Protokolle gleich leistungsfähig.

HTTP eignet sich hervorragend für eine synchrone Kommunikation zwischen Client (entspricht einem Manager) und Server (entspricht einem Agenten), da es bei seiner Entwicklung als Request-Reply-Protokoll konzipiert wurde. Eine asynchrone Nutzdatenübertragung vom Server zum Client ist dagegen nur mit Umwegen möglich. Ein HTTP-Server kann nämlich von sich aus keine Verbindung zu einem Client aufbauen. Für eine solche zusätzliche asynchrone Nutzdatenübertragung müßte der Client entweder eine Verbindung zu einem Server aufbauen, die solange blockiert, bis der Server eine Antwort liefert oder wie der Server auch einen HTTP-Server ausführen. Die asynchrone Datenübertragung mit Hilfe einer blockierenden Verbindung zu verwirklichen, ist keine empfehlenswerte Lösung. Für den asynchronen Nutzdatenverkehr eines Servers zu einem Client ist ein zusätzlicher HTTP-Server der ersten Variante vorzuziehen. Da Clients aber normalerweise keinen eigenen HTTP-Server integrieren, können andere Transportprotokolle eventuell einfachere Möglichkeiten bieten.

Eine Alternative für eine gemeinsame synchrone und asynchrone Datenübertragung ist eine Kommunikation wie bei Nachrichtenwarteschlangen oder SMTP-Servern. Bei den Nachrichtenwarteschlangen reihen Manager und Agenten ihre Nachrichten einfach in die Warteschlange des Kommunikationsziels ein. Manager und Agenten arbeiten die Nachrichten ihrer Warteschlange der Reihe nach ab und können so synchrone und asynchrone Nachrichten über einen gemeinsamen Datenkanal übertragen. Bei SMTP-Servern ist der Ablauf ähnlich, sie können Nachrichten empfangen aber auch

selbst verschicken. Deswegen bietet es sich ebenso an, daß Manager und Agenten ihre Kommunikation über SMTP-Server abwickeln [SS01].

4.7 Agentenbasiertes Monitoring über Webservices vs. SNMP

Eine interessante Frage ist, ob ein agentenbasiertes Monitoring über Webservices überhaupt irgendwelche Vorteile gegenüber SNMP hat. Mit SNMP ist es prinzipiell auch möglich, ein agentenbasiertes Monitoringsystem aufzubauen. In der Tabelle 2.2 sind die verschiedenen Typen von Traps aufgelistet, die bestimmte Ereignisse unter SNMP charakterisieren. Der letzte dort aufgelistete Typ *enterpriseSpecific* ermöglicht es Herstellern, auch eigene Ereignisse zu definieren. Dazu existiert in der Trap- bzw. Notification-PDU neben dem Typ ein weiteres Feld *specific-trap*, das nur für den Typ *enterpriseSpecific* gültig ist und bei allen anderen bisher definierten Typen den Wert *0* enthalten muß. Über dieses Feld lassen sich herstellerspezifische Ereignisse exakt identifizieren. Somit ist es möglich, zusätzliche Monitoringfunktionen, wie sie auch beim agentenbasierten Monitoring über Webservices Anwendung finden, zu definieren und über diese beiden Felder zu identifizieren. Mit den neu definierten Monitoringfunktionen kann man dann mit Einschränkungen ein agentenbasiertes Monitoringsystem aufbauen.

Ein agentenbasiertes Monitoring von Netzressourcen auf Basis von SNMP hat gegenüber Webservices trotzdem erhebliche Nachteile. Manche Monitoringfunktionen brauchen zusätzliche Parameter als Optionen oder Ereignisbedingungen (s. Abschnitt 4.3.3). Des weiteren müssen die Monitoringfunktionen für die Datenquellen einer Ressource auch noch bezüglich einer Aktivierung und Deaktivierung kontrollierbar sein. Die Festlegung der notwendigen Parameter und die Kontrolle des Monitoringablaufs kann nur aufwendig mittels einer Konfiguration über die MIB-Variablen erfolgen. Weiterhin kann eine Benachrichtigung nur mit einer Trap- bzw. Notification-PDU stattfinden. In diesem Bereich sind Webservices weitaus flexibler, Monitoringfunktionen können auf Agenten hierüber sehr einfach konfiguriert werden. Ein einziger Webserviceaufruf genügt, um die Funktionen inklusive ihrer Parameter zu kontrollieren. Benachrichtigungen an Manager sind gegenüber SNMP auch einfacher aufgebaut. Benachrichtigungen können neben der eigentlichen Information über die Auslösung eines Ereignisses, einzelnen Werten oder Wertereihen noch zusätzliche Informationen enthalten. Diese Zusatzinformationen können beispielsweise weitere Hinweise oder Auskünfte über ein Ereignis sein. Im Fehlerfall könnte es sich dabei etwa um (textuelle) Hinweise zur Fehlerbehebung handeln. SNMP-Traps bzw. -Notifications bieten diese Möglichkeit zwar auch, jedoch müssen die Daten dort in einzelne MIB-Variablen verpackt werden.

5 Spezifikation des agentenbasierten Monitoringsystems

Dieses Kapitel behandelt das Konzept des webservicebasierten Monitoringsystems von Netzwerkressourcen. Das Konzept wird in diesem Kapitel nur als Rahmenwerk vorgestellt. Dies hat den Vorteil, daß keine implementierungsspezifischen Vorgaben enthalten sind und das Konzept plattformunabhängig bleibt. Zunächst gibt der Aufbau einen Überblick über das Gesamtkonzept, beschreibt die einzelnen Module und zeigt die Kommunikationsbeziehungen zwischen ihnen auf. Die folgenden Abschnitte beschreiben danach detaillierter die einzelnen Module des Rahmenwerks inklusive der enthaltenen API-Funktionen.

5.1 Anforderungen

Die Spezifikation des Monitoringsystems muß gewisse Mindestanforderungen erfüllen, damit sie in der Praxis gute und vielfache Einsatzmöglichkeiten bietet und vom Anwender ohne Schwierigkeiten benutzt werden kann. Das System sollte möglichst so aufgebaut sein, daß der Anwender es sehr flexibel in den verschiedensten Plattformen und Betriebsumgebungen (Heterogenität) einsetzen kann. Dies ist hinsichtlich der Kommunikation durch Webservices gewährleistet, die sich sehr gut in heterogene Umgebungen einpassen. Die Portierbarkeit der Implementierung zwischen heterogenen Systemen ist dagegen größtenteils vom Betriebssystem und der Programmiersprache abhängig. Die Abhängigkeit läßt sich aber auch durch den Einsatz bestimmter Softwaretechnologien wie Java oder .NET umgehen. Des weiteren hebt sich eine Spezifikation dadurch hervor, daß sie eine zukünftige Erweiterbarkeit eines Systems ermöglicht. Dazu ist es vorteilhaft, das Monitoringsystem modular aufzubauen, was sich insbesondere für die vielen unterschiedlichen Monitoringkomponenten anbietet. Weiterhin sollte das Monitoringsystem gut skalieren, so daß viele mit einem Agenten verbundene Manager das Monitoring nicht beeinträchtigen und Benachrichtigungen immer rechtzeitig zugestellt werden. Des weiteren sollte das System bis zu einem vorgegebenen Grad fehlertolerant arbeiten, so daß zum Beispiel gestörte Kommunikationsbeziehungen zwischen einzelnen Managern und einem Agenten das Monitoring für andere Manager nicht blockieren.

5.2 Aufbau

Der Aufbau des Agenten läßt sich in drei Module aufteilen. Das *Konfigurationsmodul* bietet über eine Webserviceschnittstelle Verwaltungsfunktionen an, über die Manager Monitoringaufträge steuern können. Das *Monitoringmodul* nimmt die Aufgaben der Ressourcenüberwachung wahr und das *Benachrichtigungsmodul* verschickt im Auftrag des Monitoringmoduls Benachrichtigungen an Manager. Der Manager integriert für den Nachrichtenempfang, infolge registrierter Monitoringaufträge, eine Callback-Schnittstelle. Die Abbildung 5.1 zeigt schematisch den Aufbau des Rahmenwerks mit den drei Modulen und ihren Kommunikationsbeziehungen untereinander.

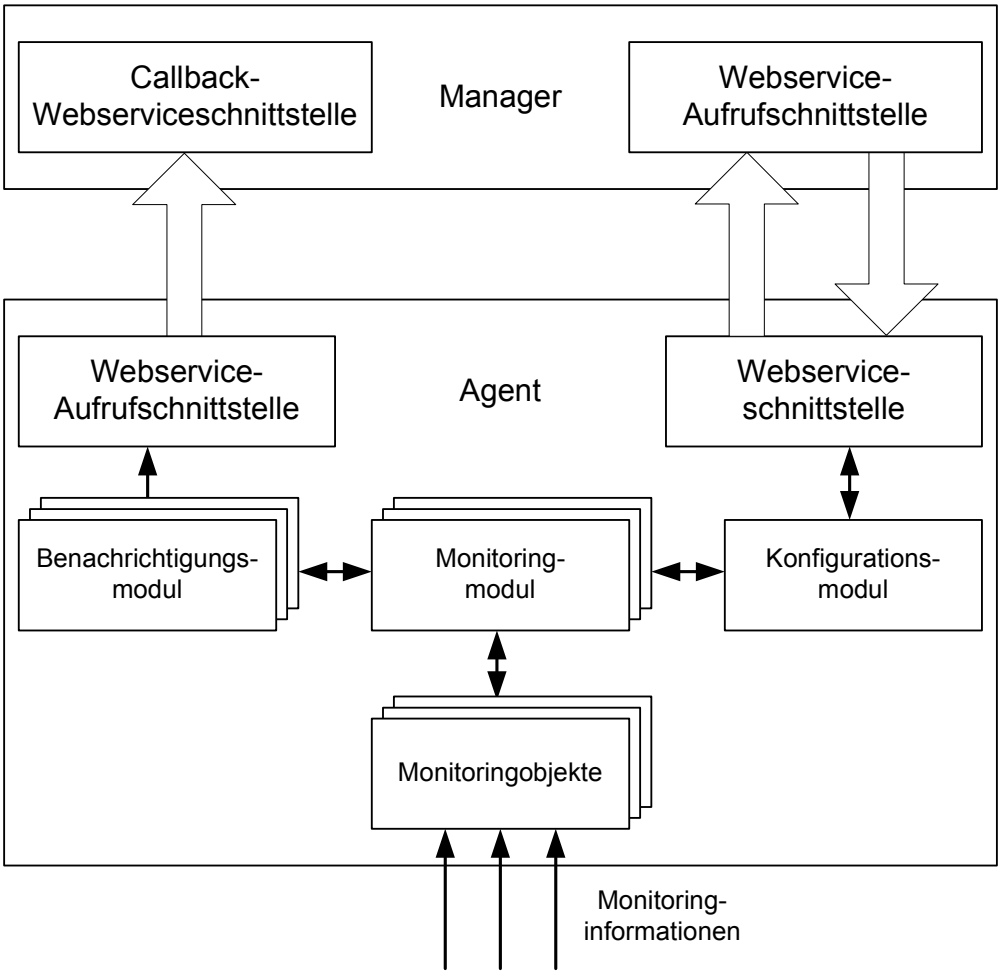

Abbildung 5.1: Rahmenwerk des agentenbasierten Monitoringsystems.

5.2.1 Konfigurationsmodul

Mehrere Manager können sich mit einem Agenten verbinden und Monitoringaufträge verwalten. Dabei arbeitet jeder Manager unabhängig von anderen Managern, die mit dem Agenten in Verbindung stehen. D. h., jeder Manager meldet nur für sich selbst

Monitoringaufträge an, ab oder konfiguriert diese um. Jeder Manager identifiziert sich daher gegenüber einem Agenten mit einer GUID (Global Unique Identifier). Es ist gestattet, daß mehrere Manager Aufträge für dieselbe Monitoringfunktion anmelden, jeder Monitoringauftrag wird deswegen als eigene Instanz ausgeführt. Es ist ebenfalls möglich, daß derselbe Manager mehrfach Aufträge für dieselbe Monitoringfunktion anmeldet.

5.2.1.1 Konfigurationsschnittstelle

Die extern zugänglichen Verwaltungsfunktionen des Konfigurationsmoduls erreicht ein Manager über die in Abbildung 5.1 dargestellte Webserviceschnittstelle des Agenten. Das Konfigurationsmodul macht hierfür einen Teil seiner Funktionalität als API zugänglich. Die Webserviceschnittstelle nimmt Webserviceaufrufe entgegen und ruft transparent die gewünschte Verwaltungsfunktion in dem API auf. Das API enthält folgende Funktionen:

`subscribe(guid, monitorfunction, parameterlist)` meldet einen Monitoringauftrag auf einem Agenten an. Der Paramter `monitorfunction` gibt den Namen der Monitoringfunktion an, `guid` entspricht der eindeutigen ID eines Managers. Der Parameter `parameterlist` gibt die erforderlichen Parameter für die Monitoringfunktion an. Bei erfolgreicher Anmeldung liefert die Funktion als Rückgabewert ein Handle zurück, welches jeden Monitoringauftrag eindeutig identifiziert.

`unsubscribe(guid, handle)` storniert zuvor registrierte Monitoringaufträge. Der Parameter `handle` gibt das Handle des zu stornierenden Monitoringauftrages an.

`reconfigure(guid, handle, parameterlist)` konfiguriert bereits angemeldete Monitoringaufträge um. Die Funktionsparameter sind die gleichen wie bei den beiden vorherigen Funktionen.

5.2.1.2 Hilfsfunktionen der Konfigurationsschnittstelle

Für die Vereinfachung der Konfiguration bietet das Konfigurationsmodul Managern noch mehrere Hilfsfunktionen an. Über diese können Manager unter anderem benötigte Parameter für die o. g. Konfigurationsfunktionen in Erfahrung bringen. Folgende Hilfsfunktionen sind in dem Konfigurations-API enthalten:

`getMonitorObjects()` liefert auf einem Agenten verfügbare Monitoringobjekte an den Aufrufer zurück.

`getParameterInformation(monitorfunction)` liefert Informationen über die erforderlichen Parameter für die Anmeldung oder Umkonfiguration eines Monitoringauftrages an den Aufrufer zurück. Der Parameter `monitorfunction` gibt die Monitoringfunktion an, für welche die Informationen abgefragt werden.

`getRegisteredMonitoringInstances(guid)` liefert eine Liste der aktuell aktiven Monitoringinstanzen eines Managers zurück. Der Parameter `guid` gibt die GUID des Managers an.

5.2.2 Monitoringmodul

Das Monitoringmodul nimmt Konfigurationsanforderungen für Monitoringfunktionen vom Konfigurationsmodul entgegen und verwaltet das Monitoring selbständig. Dazu implementiert das Modul Verwaltungsfunktionen, welche die vom Konfigurationsmodul weitergereichten Konfigurationsaufrufe (*subscribe*, *unsubscribe* und *reconfigure*) entgegennimmt. Weiterhin ist es erforderlich, daß das Monitoringmodul parallel zum Konfigurationsmodul läuft, damit sein zeitkritischer Monitoringmechanismus nicht durch Webserviceaufrufe unterbrochen wird. Zwecks Flexibilität und Erweiterbarkeit sind alle Monitoringfunktionen in eigenständigen Softwaremodulen zu implementieren, die über eine fest definierte Softwareschnittstelle aufgerufen werden und mit dem Monitoringmodul kommunizieren. Diese Softwaremodule werden als *Monitoringobjekte* bezeichnet. Das Monitoringmodul muß den Monitoringobjekten zudem ermöglichen, Benachrichtigungen über das Benachrichtigungsmodul zu verschicken.

5.2.2.1 Monitoringobjekte und -instanzen

Ein Monitoringauftrag generiert durch das Monitoringmodul eine Instanz eines Monitoringobjekts. Bei der Erstellung einer Instanz eines Monitoringobjekts hat dieses bei der Initialisierung alle übergebenen Parameter auf Gültigkeit zu überprüfen. Sind die Parameter nicht gültig, oder tritt ein anderer Fehler während der Initialisierung auf, dann muß die Initialisierung abgebrochen werden. Die Monitoringobjekte kommunizieren über eine fest definierte Schnittstelle mit dem Monitoringmodul. Dies ermöglicht später in der Implementierung eine Integration neuer Monitoringobjekte, ohne die vorhandene Agentensoftware zu ändern bzw. neu zu kompilieren. Die Monitoringschnittstelle ist in Abschnitt 5.2.2.3 dokumentiert.

5.2.2.2 Hierarchische Organisation der Monitoringobjekte

Da Agenten oftmals in vielen Bereichen zum Ressourcenmonitoring eingesetzt werden und häufig eine Vielzahl unterschiedlicher Monitoringobjekte implementieren, werden in naher Zukunft voraussichtlich sehr viele Monitoringobjekte für Monitoringanwendungen existieren. Aus diesem Grunde ist es vorteilhaft, Monitoringobjekte mit ähnlichem Ziel zusammenzufassen und in einer Hierarchie einzuordnen. Die Abbildung 5.2 zeigt in einem Beispiel den Aufbau einer Hierarchie mit mehreren Monitoringobjekten.

5.2.2.3 Monitoringschnittstelle

Die Monitoringschnittstelle definiert Funktionen, welche das Monitoringmodul für die Monitoringobjekte implementieren muß. Zugleich müssen die Monitoringobjekte Funktionen implementieren, die das Monitoringmodul aufruft. Zunächst die Funktionen, die das Monitoringmodul den Monitoringobjekten als API zur Verfügung stellt:

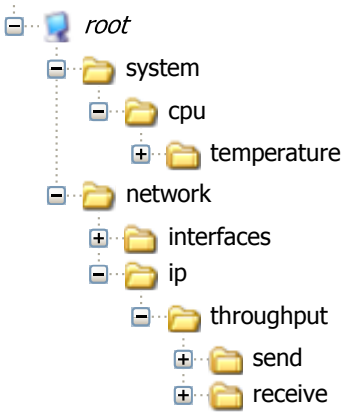

Abbildung 5.2: Hierarchische Organisation der Monitoringobjekte.

`notifyManager(monitorObjectInst, messageClass, data)` erlaubt es Monitoringinstanzen, Benachrichtigungen zu verschicken. Das Monitoringmodul leitet die Benachrichtigungen an das Benachrichtigungsmodul weiter, das diese an den richtigen Manager verschickt. Der Parameter `monitorObjectInst` gibt eine Referenz auf die eigene Monitoringinstanz an, `messageClass` die Nachrichtenklasse und `data` spezifische Daten der Monitoringinstanz.

`removeInstance(monitorObjectInst)` erlaubt Monitoringinstanzen, sich selbst aus der Liste der Monitoringinstanzen in einem Monitoringthread zu entfernen. Dies ist sinnvoll, wenn sich Monitoringinstanzen selbst beenden wollen. Der Parameter `monitorObjectInst` enthält eine Referenz auf die eigene Monitoringinstanz.

Nachfolgend das API, welches die Monitoringobjekte implementieren müssen:

`<Constructor>(parentMonitoringThread, parameterlist)` wird anläßlich der Instanziierung eines Monitoringobjekts aufgerufen und übernimmt die Initialisierung der Monitoringinstanz. Der Parameter `parentMonitoringThread` enthält die Referenz des aufrufenden Monitoringthreads und dient dazu, damit Monitoringinstanzen auf das API des Monitoringthreads zugreifen können. Die für die Initialisierung benötigten Parameter sind in `parameterlist` enthalten. Der Konstruktor hat diese Parameter auf Gültigkeit zu überprüfen. Er entscheidet, ob die Monitoringinstanz erstellt oder wegen ungültiger Parameter bzw. anderer Fehler nicht erstellt wird.

`getParameterInformation()` liefert Informationen über die Initialisierungsparameter, welche die Initialisierungsroutine eines Monitoringobjekts während der Instanziierung benötigt.

`monitor_run()` ist die Methode, die periodisch von dem Monitoringthread aufgerufen wird, bei dem die Monitoringinstanz registriert ist. Sie enthält Programmcode für die Ausführung der Monitoringaufgabe und Veranlassung von Benachrichtigungen.

`alterConfig(parameterlist)` ändert die Konfiguration einer Monitoringinstanz. Der Parameter `parameterlist` enthält vergleichbare Konfigurationsinformationen,

welche der Monitoringinstanz bei ihrer Instanziierung übergeben wurden. Diese Methode muß die neuen Parameter auf Gültigkeit überprüfen und darf bei ungültigen Parametern keine Umkonfiguration durchführen.

`cleanup()` wird bei der Abmeldung einer aktiven Monitoringinstanz durch einen Manager aufgerufen. Sie dient Aufräumarbeiten innerhalb der Instanz.

5.2.3 Benachrichtigungsmodul

Damit ein Agent einem Manager Benachrichtigungen zustellen kann, muß dieser zunächst den Endpunkt dessen Callback-Schnittstelle kennen. Während der ersten Anmeldung eines Monitoringauftrages ermittelt der Agent deshalb die Hostadresse des Managers, unter welcher die Callback-Schnittstelle erreichbar ist. Die Hostadresse wird zusammen mit der GUID gespeichert.

Verlangt eine Instanz eines Monitoringobjekts, eine Benachrichtigung an einen Manager zu verschicken, muß der Agent genau wissen, an welchen Manager die zu versendende Nachricht gerichtet ist. Manager dürfen nur Benachrichtigungen der Monitoringinstanzen erhalten, die sie selbst erzeugt haben. Aufgrund der gespeicherten GUID bei der Erstellung einer Monitoringinstanz weiß der Agent immer, an welchen Manager eine Benachrichtigung zu schicken ist. Aus Sicherheitsgründen verschickt der Agent Benachrichtigungen an Manager immer inklusive der Manager-GUID, da Nachrichten bei einem Wechsel der Hostadresse unter Umständen falsche Manager erreichen. Jeder Manager hat die GUID beim Empfang der Nachrichten auf Richtigkeit zu überprüfen. Stimmt die GUID der Nachricht nicht mit der eigenen überein, so muß der Manager die Nachricht ignorieren.

Jeder Manager stellt wie Agenten ebenfalls einen Webservice zur Verfügung. Dieser Webservice ist in Abbildung 5.1 als *Callback-Webserviceschnittstelle* gekennzeichnet. Agenten können Managern über diese Schnittstelle von Monitoringinstanzen erzeugte Benachrichtigungen zustellen. Hierfür bieten Manager einen Teil ihrer Funktionalität als API an. Die Webserviceschnittstelle nimmt Webserviceaufrufe entgegen und ruft transparent die Benachrichtigungsfunktion in dem API auf:

`pushNotification (guid, handle, messageClass, data)` nimmt Benachrichtigungen von Monitoringinstanzen auf Agenten entgegen. Der Parameter `guid` ist die GUID, die ein Manager bei der Anmeldung des mit der Benachrichtigung korrespondierenden Monitoringauftrages verwendet hat. Der zweite Parameter `handle` gibt das Handle der Monitoringinstanz an, die diese Nachricht verschickt hat. Der dritte Parameter `messageClass` gibt die Nachrichtenklasse an. Der letzte Parameter `data` enthält spezifische Daten der Monitoringinstanz.

6 Implementierung

Dieses Kapitel dokumentiert die Implementierung einer Software auf Basis der Spezifikation aus Kapitel 5. Die Implementierung teilt sich auf zwei Anwendungen auf, einer *Agentenanwendung* und einer *Manageranwendung*. Die Implementierung erfolgte in der Programmiersprache Java [Micb]. Der Vorteil liegt darin, daß die Anwendungen nahezu plattformunabhängig sind, da diese in einer JVM (Java Virtual Machine) ausgeführt werden, die für zahlreiche Hardware-Architekturen und Betriebssysteme vorhanden ist[1].

Damit Manager- und Agentenanwendungen untereinander über Webservices kommunizieren können, muß entweder die Programmiersprache eine entsprechende Unterstützung bereitstellen oder externe Komponenten integrieren können, welche die fehlende Unterstützung nachrüsten. Da die derzeitige Version 5.0 der Java 2 Standard Edition (J2SE) von sich aus keine Webserviceunterstützung mitbringt, habe ich mich für das externe Paket *Glue* [Web] der Firma Webmethods entschieden[2].

6.1 Webmethods Glue

Das Paket *Glue* der Firma Webmethods ist ein Paket, mit dem sich auf einfache Weise Anwendungen als verteilte Anwendungen mit Webservices, Java Server Pages (JSP) und Servlets erstellen lassen. Neben JSP und Servlets gewährt Glue dem Programmierer auch, eigenständige Webservice-Anwendungen zu erstellen, die keinen Webserver erfordern. Die Implementierung der Anwendung verwendet Glue für den Nachrichtenaustausch zwischen Manager- und Agentenanwendungen. Die Nachrichten werden mittels Remoteprozeduraufrufe übertragen.

6.1.1 Generelle Funktionsweise

Der Umfang von der Veröffentlichung bis zum Aufruf eines Webservices läßt sich in vier Schritte unterteilen. Zunächst wählt der Programmierer Methoden aus, die über

[1] Es sei nochmals ausdrücklich darauf hingewiesen, daß es nicht zwingend erforderlich ist, Manager und Agenten in derselben Programmiersprache für eine Plattform zu implementieren, da Webservices für den Gebrauch in heterogenen Systemen spezifiziert sind (s. Kapitel 3 ff.).

[2] Die Implementierung basiert auf der Version *Glue Standard Edition 5.0.2*. Sie bietet gegenüber den Paketen *Glue Professional Edition* und *Glue Enterprise Edition* nur einen eingeschränkten Funktionsumfang, der aber keinen großen Einfluß (Beschränkung auf ein einziges Trägerprotokoll) auf die in diesem Kapitel vorgestellte Implementierung hat.

den Webservice veröffentlicht werden sollen. Über die *Serverbindung* macht der Programmierer die Methoden unter Angabe eines Protokolls an einem Endpunkt verfügbar. Nachdem die Webserviceschnittstelle mit ihren Methoden bekannt ist, kann auf Seite des Clients die Bindung zu dem veröffentlichten Webservice generiert werden. Anschließend kann der Client alle Methoden, die in der *Clientbindung* angegeben sind, aufrufen.

Bevor der Server irgendwelche Methoden der Anwendung über einen Webservice veröffentlicht, ist es wichtig, eine genaue Entscheidung darüber zu treffen, welche Methoden als API veröffentlicht werden sollen. Es ist ratsam, möglichst nur die Methoden zu veröffentlichen, die für den Informationsaustausch zwischen Server und Client unbedingt notwendig sind. Mögliche vorhandene Methoden, die nur innerhalb des Servers gebraucht werden, könnten bei einer Veröffentlichung im Webservice und zwischenzeitlichem Aufruf den Programmablauf stören.

6.1.2 Webservice erstellen

Voraussetzung für die Veröffentlichung eines Webservices ist eine einheitliche Schnittstellenbeschreibung, welche die angebotenen Methoden, deren Aufrufparameter mit Datentypen sowie den Rückgabewert mit Datentypen enthält. Sofern Methodenaufrufe Fehlermeldungen zurückliefern können (in Java als Exceptions), so sind diese in der Schnittstellenbeschreibung auch definiert. Die Schnittstellenbeschreibung wird über die bereits in Kapitel 3.6 vorgestellte Sprache WSDL definiert. Der Programmierer muß die Schnittstellenbeschreibung in WSDL aber nicht aufwendig von Hand erledigen. Bei der Erstellung eines Webservices mit Glue reicht es, die Schnittstelle in Form eines Javainterfaces zu definieren, die WSDL-Beschreibung generiert Glue später selbständig.

Der erste Schritt für die Erstellung eines Webservices besteht darin, alle zu veröffentlichenden Methoden in einem Javainterface zu definieren. Der folgende Codeausschnitt zeigt einen Teil der Schnittstellenbeschreibung *IServiceConfig* aus der Agentenanwendung, in welcher die Methoden zur Konfiguration von Managementaufträgen definiert sind:

```
public interface IServiceConfig
{
  //web methods
  int subscribe(long guid, String monitorfunction,
     String[] parameterlist) throws ...;

  void unsubscribe(long guid, int handle) throws ...;

  void reconfigure(long guid, int handle,
     String[] parameterlist) throws ...;

  String[] getMonitorObjects();

  String[] getParameterInformation(String
     monitorfunction) throws ...;
```

```
    String[][] getRegisteredMonitoringInstances(long guid)
       throws ...;
    ...
}
```

Diese Schnittstelle muß dann in einer Javaklasse implementiert werden. In der Agentenanwendung ist dies die Klasse *ServiceConfig*:

```
public class ServiceConfig implements IServiceConfig
{
  ...
}
```

In der Manageranwendung ist der Webservice mit dem Javainterface *INotificationEndpoint* und dessen implementierende Klasse *NotificationEndpoint* in ähnlicher Weise ausgeführt. Er nimmt die Benachrichtigungen von Agenten entgegen.

6.1.3 Serverbindung

Nach der Definition und Implementierung des Webservices folgt die Veröffentlichung über ein Trägerprotokoll an einem lokalen Endpunkt. Da diese Version von Glue als Protokoll nur HTTP unterstützt, ist das Protokoll von vornherein festgelegt und bedarf in diesem Kapitel keiner weiteren Diskussion. In der Agentenanwendung geschieht die Bindung in der Klasse *Main* über die folgenden zwei Methodenaufrufe:

```
HTTP.startup("http://localhost:8080/");
Registry.publish("agent", new ServiceConfig());
```

`HTTP.startup` veranlaßt Glue, unter der angegebenen URL eingehende HTTP-Nachrichten entgegenzunehmen. Läuft unter dem in der URL angegebenen Endpunkt noch kein HTTP-Server, so startet Glue einen. Darauffolgend bindet die `publish`-Methode den Webservice an den Zugriffsnamen `agent`. Alle Methoden des Interfaces *IServiceConfig* sind von nun an über den Webservice zugreifbar. Gleichzeitig mit der Publikation des Webservices generiert Glue aus dem Javainterface die Schnittstellenbeschreibung in der WSDL-Sprache und veröffentlicht diese unter der URL des Webservices mit dem Zusatz `.wsdl`. Die vollständigen URLs des Webservices und WSDL-Dokuments lauten:

- Webservice: `http://localhost:8080/agent`
- WSDL-Dokument: `http://localhost:8080/agent.wsdl`

In der Manageranwendung erfolgt die Bindung für die Entgegennahme der Benachrichtigungen in der Klasse *WebserviceOperations* über die beiden Methoden

```
HTTP.startup("http://localhost:8081/");
Registry.publish("notifications",
   new NotificationEndpoint());
```

Der Webservice und das WSDL-Dokument sind demnach unter den nachfolgenden URLs erreichbar:

- Webservice: `http://localhost:8081/notifications`
- WSDL-Dokument: `http://localhost:8081/notifications.wsdl`

6.1.4 Clientbindung

Damit der Client auf den Webservice des Servers zugreifen kann, wird auf diesem zuerst ein Stub generiert. Der Stub dient dem Client als lokaler Zugriffspunkt und verbirgt die dahinterstehende Komplexität. Der Stub stellt die gesamte Funktionalität des entfernten Webservices so zur Verfügung, als wären dessen Methoden lokal vorhanden.

Genauso wie die WSDL-Dokumente muß der Stub nicht von Hand geschrieben werden, sondern kann mit dem Hilfsprogramm *wsdl2java* aus dem WSDL-Dokument des Webservice generiert werden. In der Manageranwendung ist der Stub in dem Paket *manager.stub* abgelegt. Der folgende Programmaufruf generiert den Stub für die Manageranwendung:

```
wsdl2java http://localhost:8080/agent.wsdl ↩
   -p manager.stub
```

Der Stub der Agentenanwendung liegt in dem Paket *agent.stub* und wird mit folgendem Programmaufruf erstellt:

```
wsdl2java http://localhost:8081/notifications.wsdl ↩
   -p agent.stub
```

Die Stubs enthalten das Javainterface des Webservices und eine Klasse, welche die `bind`-Methode für den Verbindungsaufbau kapselt, die Glue bereitstellt. Falls Methoden im Javainterface Exceptions generieren können, so enthält der Stub auch noch die erforderlichen Exceptiondefinitionen.

6.1.5 Methodenaufruf im Webservice

Der Client verbindet sich zuerst über die `bind`-Methode mit dem Webservice des Servers, diese liefert eine Proxyinstanz auf die Serverimplementierung zurück. Über die Instanz kann der Client alle veröffentlichten Methoden transparent aufrufen. In der Manageranwendung erfolgt die Bindung an die Agenten über den Methodenaufruf

```
Registry.bind("http://<Agenthost>/agent.wsdl");
```

Agenten verbinden sich über folgenden Methodenaufruf mit der Benachrichtigungsschnittstelle eines Managers:

```
Registry.bind("http://<Managerhost>/notifications.wsdl");
```

6.2 Agentenanwendung

Die Agentenanwendung enthält in dieser Implementierung nicht nur einen Agenten nach der im vorherigen Kapitel vorgestellten Spezifikation. Vielmehr simuliert die Anwendung eine komplette Ressource, die neben dem Agenten auch noch einen Datengenerator enthält, dessen Daten der Agent für das Monitoring heranzieht. Das Hauptaugenmerk liegt aber weiterhin auf der Implementierung des Agenten.

6.2.1 Implementierung der Monitoringfunktion

Entsprechend der Spezifikation implementiert jedes Monitoringobjekt seine Monitoringfunktionalität in der Methode `monitor_run()`. Diese Methode wird periodisch von einem Monitoringthread aufgerufen. Sollte dieser periodische Methodenaufruf für die Aufgabe eines Monitoringobjekts nicht geeignet sein, so kann ein Monitoringobjekt seine Aufgabe stattdessen auch in einem eigenen Thread implementieren. Hierfür muß das Monitoringobjekt in der Methode `cleanup()` aber zusätzlichen Programmcode integrieren, der den Thread, im Falle einer Abmeldung der Monitoringinstanz durch einen Manager, beendet.

6.2.2 Monitoringaufträge anmelden

Möchte ein Manager einen Monitoringauftrag anmelden, so ruft er die Remoteprozedur

```
subscribe(guid, monitorfunction, parameterlist);
```

auf. Hat der Manager zu dieser Zeit keinen weiteren Monitoringauftrag angemeldet, dann erstellt der Agent einen neuen Thread, der nur dem jeweiligen Manager zugeordnet ist. Dieser Thread (*MonitoringThread*) übernimmt das Monitoring aller angemeldeten Aufträge des Managers. Ferner erstellt der Monitoringthread einen weiteren Thread *NotificationThread*, dieser übernimmt die Benachrichtigung an den Manager. Laufen für den anmeldenden Manager bereits Monitoringaufträge, werden keine neuen Threads erstellt. Der neue Auftrag wird dann einfach nur an den entsprechenden Monitoringthread übergeben.

Die Klasse *ManagerAdmin* ordnet jedem Manager einen eigenen Monitoringthread zu, um die Skalierbarkeit bei vielen Monitoringaufträgen zu verbessern. Da jeder Monitoringthread für jeden Monitoringauftrag die Methode `monitor_run()` in einem festen Zeitintervall aufruft, ist die Skalierbarkeit geringer, falls alle diese Methodenaufrufe in einem gemeinsamen Thread ausgeführt werden. Ähnlich ist es, wenn jeder Monitoringauftrag in einem eigenen Thread ausgeführt wird. Dann sinkt die Skalierbarkeit ebenfalls, da der Verwaltungsaufwand für die Threads im Verhältnis zu dem Arbeitsaufwand jedes einzelnen Threads stark ansteigt. Die o. g. Vorgehensweise verhindert außerdem, daß Monitoringaufträge anderer Manager beeinträchtigt werden, falls in einer Monitoringinstanz eines Managers ein Fehler auftritt. Ebenso soll die Zuordnung

eines einzelnen Benachrichtigungsthreads an jeden Manager zusichern, daß Benachrichtigungen an andere Manager weiterhin möglich sind, falls der Remoteprozeduraufruf auf einem Manager aufgrund eines Fehlers blockiert.

Die Verwaltung der Monitoringthreads übernimmt die Klasse *ManagerAdmin*, sie entscheidet, ob neue Monitoringthreads gestartet werden dürfen oder nicht. Aus Leistungsgründen ist die Anzahl zulässiger Monitoringthreads beschränkt. Diese Implementierung erlaubt maximal zehn Monitoringthreads. Die Abhängigkeiten zwischen den Klassen *ManagerAdmin*, *MonitoringThread* und *NotificationThread* ist in Abbildung 6.1 dargestellt.

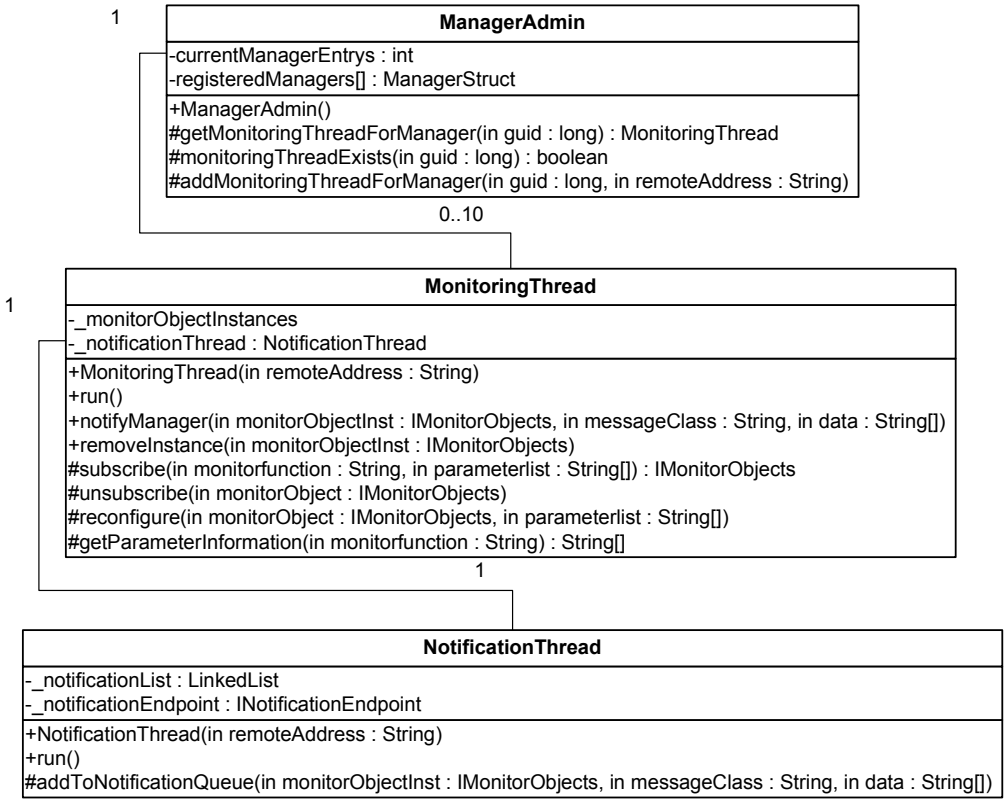

Abbildung 6.1: Klassendiagramm: Monitoring und Benachrichtigung.

6.2.3 Monitoringaufträge abmelden

Möchte ein Manager einen Monitoringauftrag abmelden, dann ruft er die Remoteprozedur

```
unsubscribe(guid, handle);
```

auf. Die Agentenanwendung sucht dann in der Klasse *ServiceConfig* in der globalen Liste aller registrierten Monitoringinstanzen die Instanz heraus, auf welche die GUID und das Handle zutreffen. Ist eine zutreffende Monitoringinstanz vorhanden, wird anschließend der Monitoringthread, der die Monitoringinstanz verwaltet, angewiesen,

die aktive Monitoringinstanz zu beenden. Danach löscht die Agentenanwendung den gefundenen Eintrag in der globalen Liste.

6.2.4 Monitoringaufträge umkonfigurieren

Die Umkonfiguration läuft ähnlich wie die Löschung der Monitoringaufträge aus dem vorherigen Abschnitt ab. Dazu ruft ein Manager die Remoteprozedur

```
reconfigure(guid, handle, parameterlist);
```

auf. Die Agentenanwendung durchsucht die globale Liste aller registrierten Monitoringinstanzen nach der Instanz, auf die GUID und Handle zutreffen. Ist die Instanz gefunden, weist sie den Monitoringthread, der die Monitoringinstanz verwaltet, an, eine Umkonfiguration derselben vorzunehmen. Der Monitoringthread ermittelt nun seinerseits in seiner eigenen Liste die gesuchte Monitoringinstanz und ruft dessen Methode

```
alterConfig(parameterlist);
```

auf. Die Monitoringinstanz überprüft zuerst, ob die übergebenen Parameter gültig sind. Sind alle Parameter gültig, dann konfiguriert sie ihre Einstellungen um.

6.2.5 Hierarchieaufbau der Monitoringobjekte

Die Monitoringobjekte in der Agentenanwendung sind in eigenen Javapaketen unterhalb der Ebene *agent.monitorObjects* implementiert. Innerhalb dieser Ebene kann die Ebenentiefe beliebig weitergeführt werden. Das ist in der Beispielimplementierung des Monitoringobjekts *Temperature* mit dem Paketnamen *agent.monitorObjects.system.cpu* gut nachvollziehbar. Java speichert die Pakete entsprechend ihrer Paketnamen in eine äquivalente Verzeichnishierarchie im Dateisystem oder in einem Java-Archiv (JAR)[3].

Aus der Verzeichnishierarchie erstellt die Implementierung die im Abschnitt 5.2.2.2 behandelte hierarchische Organisation der Monitoringobjekte. Hierzu durchsucht die Agentenanwendung rekursiv das zugehörige Verzeichnis zum Paket *agent.monitorObjects* nach Javaklassen mit der Dateiendung *.class*. Allen Monitoringobjekten wird so ein eindeutiger vollqualifizierter Name, bestehend aus dem partiellen Verzeichnispfad und seinem Dateinamen (ohne Dateierweiterung *.class*), zugeordnet. Das o. g. Monitoringobjekt *Temperature* bekommt so den Namen *system.cpu.Temperature*. Über die vollqualifizierten Namen werden die Monitoringobjekte adressiert.

[3] Eine JAR-Datei ist eine ZIP-Datei (komprimierbares Dateiarchiv) mit der Bedingung, daß diese als erste Datei eine Datei mit dem Namen *manifest* enthalten muß. Diese Datei enthält Attribute und digitale Signaturen für die weiteren Javadateien, die in dem Archiv enthalten sind [Fla99].

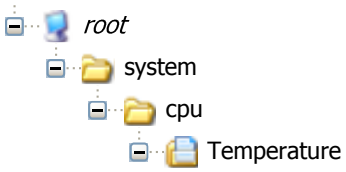

Abbildung 6.2: Monitoringobjekt *Temperature* im Javapaket *agent.monitorObjects.system.cpu*.

6.2.6 Datengenerator

Die Agentenanwendung enthält neben der Monitoringkomponente noch einen Datengenerator, da die Prototypimplementierung kein Bestandteil einer Netzwerkressource ist. Im Produktiveinsatz entfällt der Datengenerator, da die Netzwerkressourcen ihre Daten in diesem Fall selbst liefern.

Der Datengenerator ist in der Klasse *DataGenerator* implementiert und läuft als eigenständiger Thread. Der Thread generiert in einer Endlosschleife diverse Testdaten, welche die Monitoringobjekte im Prototyp abfragen. Über statische Methoden können die Monitoringobjekte die Testdaten abrufen.

6.3 Manageranwendung

Die Manageranwendung implementiert eine grafische Benutzeroberfläche. Der Anwender kann sich darüber mit einem beliebigen Agenten verbinden und Monitoringaufträge an den Agenten delegieren. Bereits aktive Aufträge kann der Anwender zwischenzeitlich auch umkonfigurieren oder auch wieder abmelden. Weiterhin kann sich der Manager auch mit anderen Agenten verbinden und dort weitere Monitoringaufträge steuern. Eventuell auftretende Benachrichtigungen von Agenten gibt die Benutzeroberfläche in einem Textfenster aus. Die Zustellung der Benachrichtigungen ist unabhängig davon, mit welchem Agenten der Manager gerade eine Konfigurationsverbindung unterhält. Die Abbildung 6.3 zeigt die Bedienoberfläche der Manageranwendung.

6.3.1 Monitoringaufträge anmelden

Möchte ein Anwender einen Monitoringauftrag anmelden (s. Abschnitt 6.2.2), dann muß der Manager den Agenten zunächst fragen, welche Monitoringobjekte er unterstützt. Über den Remoteprozeduraufruf

```
getMonitorObjects();
```

liefert der Agent ein Stringarray mit den vollqualifizierten Namen seiner Monitoringobjekte zurück. Der Manager bietet diese in einer Auswahlbox an. Hat der Anwender das gewünschte Monitoringobjekt ausgewählt, dann ruft der Manager über die Remoteprozedur

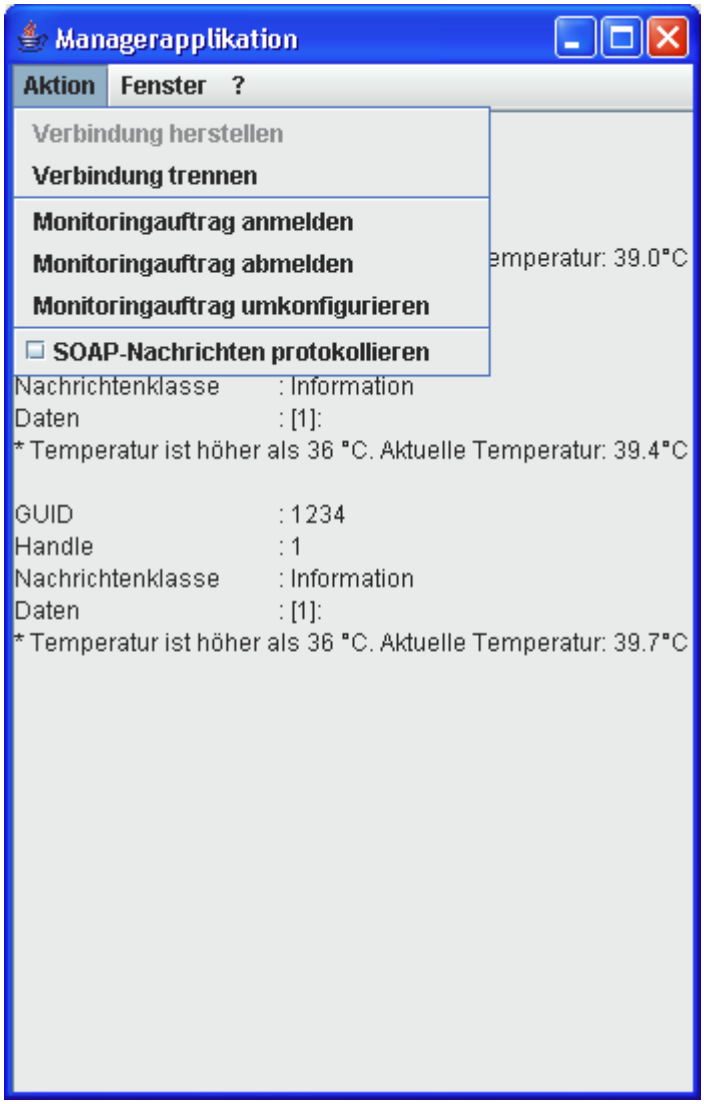

Abbildung 6.3: Bedienoberfläche der Manageranwendung.

```
getParameterInformation(monitorfunction);
```

Informationen über dessen Parameter ab. Nachdem der Anwender die Parameter des Monitoringobjekts festgelegt hat, ruft der Manager die Anmeldeprozedur im Agenten auf.

6.3.2 Monitoringaufträge abmelden oder umkonfigurieren

Die Abmelde- und Umkonfigurationsprozedur gleichen sich im ersten Schritt, hierzu fragt der Manager über die Remoteprozedur

```
getRegisteredMonitoringInstances(guid);
```

zunächst ab, welche Monitoringinstanzen unter seiner GUID beim Agenten registriert sind. Der Agent liefert ein Stringarray mit den vollqualifizierten Namen der registrierten Monitoringinstanzen inklusive ihrer Handles zurück. Der Anwender kann dann den richtigen Auftrag aus einer Auswahlbox auswählen. Soll eine Monitoringinstanz abgemeldet werden, ruft der Manager die Abmeldeprozedur im Agenten auf (s. Abschnitt 6.2.3). Möchte der Anwender eine Monitoringinstanz statt dessen nur umkonfigurieren, dann ruft der Manager die Remoteprozedur

```
getParameterInformation(monitorfunction);
```

auf, um Informationen über die Parameter des zugehörigen Monitoringobjekts zu bekommen. Nach Angabe der Parameter durch den Anwender ruft der Manager die Umkonfigurationsprozedur im Agenten auf (s. Abschnitt 6.2.4).

7 Test der Implementierung

Bevor diese Arbeit einer Evaluierung unterzogen werden kann, muß das entwickelte System mit samt der Prototypimplementierung der Monitoringanwendung auf Tauglichkeit überprüft werden. In einem ersten Schritt wurde hierfür der Programmablauf in der Agenten- und Manageranwendung mit einem Debugger untersucht. Obendrein wurde im darauffolgenden Schritt das Framework *JUnit* [GB] für weitere Tests eingesetzt. Im letzten Testabschnitt wurden Agenten- und Manageranwendung nochmals unter normalen Betriebsbedingungen getestet.

7.1 JUnit

JUnit ist ein Framework zum Testen von in Java geschriebenen Programmen. Neben der eigentlichen Softwareentwicklung programmiert der Entwickler mit JUnit Tests für einzelne Programmkonstrukte. Die einzelnen Tests werden als Testfälle bezeichnet. Die Testfälle sind so aufgebaut, das sie Methoden der Anwendung aufrufen und überprüfen, ob diese erwartete Rückgabewerte zurückliefern. Die Tests sollten neben der Überprüfung auf gültige Rückgabewerte möglichst auch Testfälle beinhalten, die durch bewußt falsch gewählte Eingabeparameter Fehler provozieren. In diesen Fällen überprüfen die Tests, ob als Rückgabewert der entsprechende Fehler zurückgeliefert wird. Die Ausführung eines JUnit-Tests ruft alle oder vereinzelt programmierte Testfälle nacheinander auf. Stimmt in einem Testfall der zurückgelieferte Rückgabewert eines Methodenaufrufs nicht mit dem erwarteten überein, protokolliert JUnit einen Fehler für diesen Testfall. Der JUnit-Test wird so oft durchgeführt, bis alle Testfälle fehlerfrei ablaufen.

Die Tests in der Implementierung verfahren auf die o. g. Weise. Sie überprüfen, ob die in den Tests aufgerufenen Methoden korrekte Rückgabewerte liefern. Bei absichtlich fehlerhaften Eingabewerten überprüfen die Tests entsprechend, ob die Methoden einen korrekten Fehlerwert in Form einer Exception zurückliefern. Mit JUnit wurden Tests für die Konfigurations- und Hilfsmethoden im Agentenwebservice erstellt und ausgeführt.

7.2 Benutzertest

Dieser Test überprüft die ordnungsgemäße Funktion der Agenten- und Manageranwendung in der Praxis durch einen Benutzer. Für diesen Test wurden mit beiden Anwendungen drei Testszenarien mit unterschiedlichen Kommunikationsbeziehungen

durchgespielt. Dabei soll gezeigt werden, daß sich das System bei einer Eins-zu-Eins-Verbindung sowie bei Mehrfachverbindungen auf Seite des Agenten als auch auf Seite des Managers den Anforderungen entsprechend verhält. Die drei Testszenarien decken folgende Kommunikationsbeziehungen ab:

- 1 Manager $\Leftrightarrow$ 1 Agent
- 2 Manager $\Leftrightarrow$ 1 Agent
- 1 Manager $\Leftrightarrow$ 2 Agenten

Der Ablauf der drei Testszenarien wird in den folgenden Abschnitten noch einmal kurz erläutert:

7.2.1 Testszenario 1

In diesem Testszenario kommunizieren je ein Manager und Agent miteinander. Dieser Test überprüft über die grafische Oberfläche der Manageranwendung die generelle Funktionalität der beiden Anwendungen. Dieser Test umfaßt den Aufruf aller drei Konfigurationsfunktionen auf unterschiedliche Monitoringobjekte mit diversen Parameterkonfigurationen. Zusätzlich überprüft der Test noch, ob die Benachrichtigungen zugestellt werden.

7.2.2 Testszenario 2

In diesem Testszenario steuern zwei Manager gleichzeitig Monitoringaufträge auf einem Agenten. Dieser Test soll prüfen, ob die Agentenanwendung mehrere Manager gleichzeitig bedienen kann. Dieser Test überprüft insbesondere, ob jeder Manager nur auf seine eigenen Daten zugreifen kann. Für Manager dürfen auf der Agentenanwendung nur ihnen selbst zugeordnete Daten sichtbar und konfigurierbar sein. Wäre dies nicht der Fall, ermöglicht dies einem Manager Monitoringaufträge eines anderen Managers zu verändern.

7.2.3 Testszenario 3

Dieser Benutzertest überprüft, ob die Manageranwendung während ihrer Laufzeit ohne Störungen mit mehreren Agenten kommunizieren kann. In diesem Test kommuniziert ein Manager mit zwei Agenten. Der Manager konfiguriert zuerst nacheinander mehrere Monitoringaufträge auf beiden Agenten. Danach wird überprüft, ob der Manager von beiden Agenten Benachrichtigungen erhält.

7.3 Schlußbemerkung

Die hier dargelegten Tests haben die Implementierung auf drei Ebenen überprüft. Die implementierungsbegleitende Überprüfung mit dem Debugger hat den inhaltlichen Ablauf innerhalb der Methoden überprüft. Auf der nächsthöheren Ebene hat das Hilfswerkzeug *JUnit* den Zusammenhang von Eingabe und Ausgabe einiger Methoden überprüft. Auf der höchsten Ebene hat der Benutzer dann in mehreren Anwendungsbeispielen das Zusammenspiel zwischen Manager- und Agentenanwendung getestet. Die Tests auf allen drei Ebenen wurden erfolgreich abgeschlossen.

8 Bewertung

Mit dem Entwurf eines agentenbasierten Monitoringsystems, der daraus entwickelten Spezifikation und anschließender Implementierung einer Beispielanwendung wurde untersucht, inwiefern sich Webservices für ein agentenbasiertes Monitoring eignen.

Die fertig entwickelte Spezifikation und die darauf aufbauende Implementierung eines Prototypen haben zwei wesentliche Dinge gezeigt. Zum einen ist ein agentenbasiertes Monitoring einem managerbasierten Monitoring in einigen Bereichen überlegen. Und zum anderen eignen sich Webservices hervorragend für ein agentenbasiertes Monitoring. Der ausführliche Test der Implementierung untermauert diese Ergebnisse.

Der Entwurf in Kapitel 4 legt bereits dar, daß ein auf Remoteprozeduren basierender Datenaustausch, wie ihn auch Webservices bereitstellen, eine gute Kommunikationsbasis bietet. Mit den weiteren Funktionen der Schnittstellendefinition über WSDL, dessen Veröffentlichung und der Auffindung von Webservices über den Verzeichnisdienst UDDI bieten Webservices ein enormes Potential für ein agentenbasiertes Monitoring. Da Webservices mittlerweile auf den unterschiedlichsten Plattformen verfügbar und hinreichend gut standardisiert sind, ermöglichen sie heterogenen Systemen eine problemlose Kommunikation. Dies spricht ebenfalls dafür, sie als Basis für den Datenaustausch zu verwenden.

Der agentenbasierte Ansatz hat gezeigt, daß dieser dem gegenwärtig verwendeten Framework SNMP im Monitoringbereich in mehrfacher Hinsicht überlegen ist. Der agentenbasierte Ansatz zeichnet sich durch eine einfache Konfiguration der Monitoringaufträge aus und erlaubt eine wesentlich ausführlichere Benachrichtigung als Traps oder Notifications in SNMP. Die ausführlicheren Benachrichtigungsmöglichkeiten können die unterschiedlichsten Daten transportieren. Daher eignet sich dieses System neben der Überwachung einzelner Laufzeitparameter einer Ressource auch für komplexere Monitoringaufgaben, die unter Umständen auch umfangreiche Datenmengen für Manager erzeugen.

In der Implementierung wurde bewußt Wert auf eine Modularität gelegt. Sie gewährt, nebem dem Aspekt der Codewartbarkeit, eine einfache und problemlose Erweiterung um weitere Monitoringmodule. Die fest definierte Schnittstelle zwischen dem Monitoringmodul und den Monitoringobjekten garantiert die Kommunikation der beiden Softwarekomponenten.

Die Tests der Implementierung haben auf unterschiedlichen Ebenen gezeigt, daß die Spezifikation des agentenbasierten Monitoringsystems sowie die Anforderungen der Aufgabenstellung von der Implementierung erfüllt wurden.

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

In den letzten Jahren läßt sich in der Gesellschaft des Netzwerkmanagements ein verstärkter Trend beobachten, moderne Technologien wie Webservices und XML im Bereich des Netzwerkmanagements einzusetzen. Das in der Entwicklung befindliche NETCONF-Protokoll ist ein Beispiel dafür. In dieser Diplomarbeit wurde daher untersucht, ob und wie ein agentenbasiertes Netzwerkmonitoring mit Hilfe von Webservices verwirklicht werden kann.

Der erste Teil dieser Arbeit zeigt eine Entwicklung des Netzwerkmanagements bis heute und beschreibt den weit verbreiteten Standard SNMP. Er bestimmt die minimalen Anforderungen an das neu zu entwickelnde System des agentenbasierten Monitorings, das Webservices als Grundlage hat.

Nach einer detaillierten Beschreibung des Themas *verteilte Systeme* und *Webservices* mit den Standards XML, SOAP, UDDI und WSDL wurde ein agentenbasiertes Monitoringsystem entworfen. Es beschreibt in groben Zügen Ablauf, Konfiguration und Benachrichtigung. Für die Konfiguration wurden drei verschiedene Konfigurationsansätze miteinander verglichen und ausgewertet, welcher am besten geeignet ist.

Die nachfolgend entwickelte Spezifikation setzt auf dem Entwurf auf. Sie definiert den modularen Aufbau des Gesamtkonzepts, die Kommunikationsbeziehungen der Module und die erforderlichen Schnittstellen für Konfiguration, Benachrichtigung und Monitoringobjekte.

Mit der Spezifikation als Vorlage wurde darauf ein Prototyp implementiert, der zeigen soll, daß die theoretischen Überlegungen in die Praxis umgesetzt werden können und auch funktionieren. Abschließend wurden Entwicklung und Implementierung einer Bewertung unterzogen.

9.2 Ausblick

Die in dieser Diplomarbeit entwickelte Spezifikation und die anschließende Implementierung in einer Testanwendung zeigen, daß das agentenbasierte Monitoringsystem funktionsfähig ist. In der Spezifikation und der Implementierung gibt es noch mehrere Möglichkeiten für zukünftige Erweiterungen oder Änderungen. Einige davon sind in den folgenden Abschnitten ansatzweise beschrieben:

9.2.1 Agenten über UDDI auffinden

Agenten könnten sich nach ihrem Start in den *Green Pages* eines UDDI-Verzeichnisdienstes eintragen. So brauchen Manager nur den Verzeichnisdienst befragen, um vorhandene Agenten aufzufinden. Über den Verzeichnisdienst können Manager die Adresse des Konfigurationsendpunktes eines Agenten beziehen und sich daraufhin mit diesem verbinden. Statt die Adresse jedes Agenten kennen zu müssen, braucht ein Manager nur noch wissen, unter welcher Adresse der Verzeichnisdienst erreichbar ist.

9.2.2 Metadaten über Monitoringobjektparameter

Jedes Monitoringobjekt erwartet bei der Initialisierung einer neuen Instanz oder während ihrer Umkonfiguration relevante Parameter in dem Stringarray `parameterlist`. Die Methode `getParameterInformation(monitorfunction)` liefert für Monitoringobjekte Informationen über die Anzahl der Parameter in der Parameterliste und eine textuelle Beschreibung über diese zurück. Zukünftig wäre es sinnvoll, diese Informationen durch präzisere zu ersetzen, die zudem ohne großen Aufwand maschinell verarbeitet werden können. Bezüglich Webservices bietet es sich an, diese Informationen strukturiert in einem XML-Dokument zusammenzufassen. Dieses Dokument könnte folgende Daten beinhalten:

- Parameteranzahl
- Datentyp jedes Parameters
- Wertebereiche für Parameter
- vordefinierte gültige Werte für Parameter
- maximale Längenangaben für Strings
- empfohlene Standardwerte für Parameter
- Kurzbeschreibung des Monitoringobjekts
- ...

9.2.3 Erweiterung der Monitoringschnittstelle

Die Monitoringschnittstelle könnte um weitere Funktionen erweitert werden, die eine umfangreichere Steuerung zwischen dem Monitoringmodul und den -objekten ermöglichen. Beispielsweise könnte für Benachrichtigungen in das Monitoringmodul eine Flußkontrolle integriert werden, um mögliche Benachrichtigungsstürme von Monitoringobjekten zu vermeiden.

9.2.4 Übertragungsprotokolle für Webservices

Der SOAP-Nachrichtenaustausch zwischen Managern und Agenten findet zur Zeit über das Protokoll HTTP statt. Weitere Transportprotokolle könnten daraufhin untersucht werden, ob sie sich gut für den Austausch von SOAP-Nachrichten in dieser Anwendung eignen.

A Bedienung der Manageranwendung

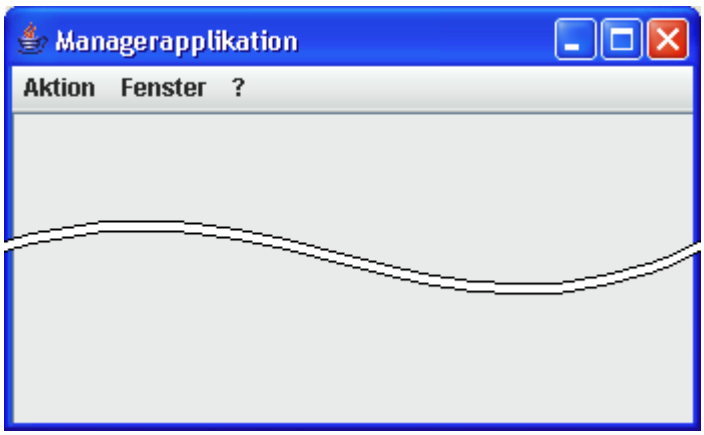

Abbildung A.1: Bedienoberfläche der Manageranwendung.

A.1 Starten der Manager- und Agentenanwendung

Die Manageranwendung wird mit der Befehlszeile

```
java -jar manager.jar
```

gestartet[1]. Vergleichbares gilt für die Agentenanwendung mit

```
java -jar agent.jar
```

A.2 Verbindung zu einem Agenten herstellen

1. Wählen Sie im Menü *Aktion* den Menüpunkt *Verbindung herstellen* aus.
2. Im darauffolgenden Dialogfenster geben Sie die Endpunktadresse des Agenten ein, mit dem sich die Manageranwendung verbinden soll.

[1] Vor dem Start der Anwendungen müssen sich die externen Webservicekomponenten von *Glue* in demselben Verzeichnis wie die Anwendungen befinden. Konkret betrifft dies die Webservicebibliothek *glue-all.jar* und die zugehörige Lizenz *webMethods-license.xml*.

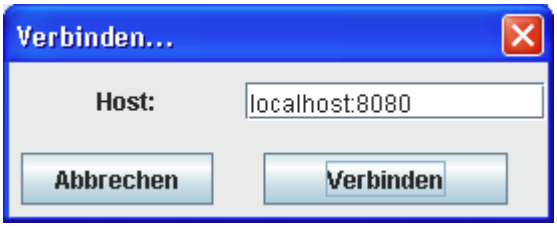

Abbildung A.2: Verbindungsaufbau.

3. Bestätigen Sie die Eingabe über die Schaltfläche *Verbinden*. (Die Schaltfläche *Abbrechen* bricht den Verbindungsaufbau ab.)

A.3 Verbindung von einem Agenten trennen

1. Wählen Sie im Menü *Aktion* den Menüpunkt *Verbindung trennen* aus. (Hinweis: Die Verbindungstrennung von einem Agenten beeinträchtigt nicht den Empfang von Benachrichtigungen, sondern betrifft allein die Konfigurationsverbindung.)

A.4 Monitoringauftrag anmelden

1. Wählen Sie im Menü *Aktion* den Menüpunkt *Monitoringauftrag anmelden* aus.
2. Im darauffolgenden Dialogfenster wählen Sie das gewünschte Monitoringobjekt aus.

Abbildung A.3: Monitoringauftrag anmelden.

3. Bestätigen Sie die Auswahl mit der Schaltfläche *Weiter* >. (Die Schaltfläche *Abbrechen* bricht den Anmeldevorgang ab.)
4. Über das darauffolgende Dialogfenster können Sie die erforderlichen Parameter des neuen Monitoringauftrages einstellen. Das linke Teilfenster enthält eine Beschreibung des Monitoringobjekts und eine Anleitung für dessen Parameter. Das rechte Teilfenster enthält die Eingabefelder für die Parameter.

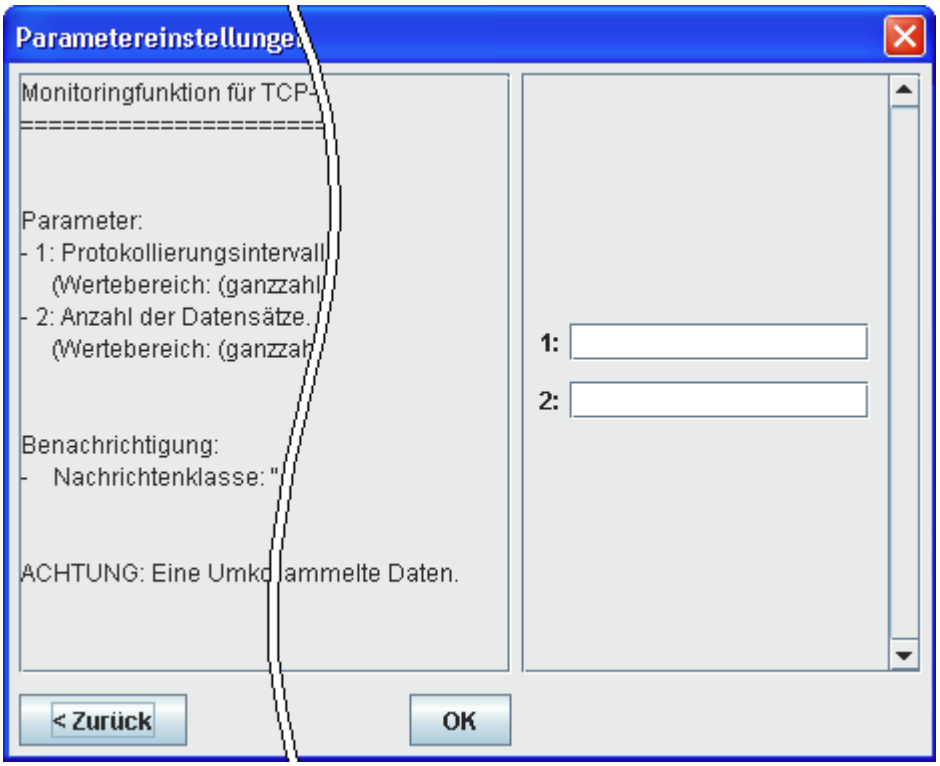

Abbildung A.4: Monitoringparameter einstellen.

5. Bestätigen Sie die Parametereingabe mit der Schaltfläche *OK*. (Die Schaltfläche *< Zurück* führt zum vorherigen Dialogfeld.)

A.5 Monitoringauftrag abmelden

1. Wählen Sie im Menü *Aktion* den Menüpunkt *Monitoringauftrag abmelden* aus.
2. Im darauffolgenden Dialogfenster wählen Sie den abzumeldenden Monitoring-auftrag aus.

Abbildung A.5: Monitoringauftrag abmelden.

3. Bestätigen Sie die Auswahl mit der Schaltfläche *OK*. (Die Schaltfläche *Abbrechen* bricht den Abmeldevorgang ab.)

A.6 Monitoringauftrag umkonfigurieren

1. Wählen Sie im Menü *Aktion* den Menüpunkt *Monitoringauftrag umkonfigurieren* aus.
2. Im darauffolgenden Dialogfenster wählen Sie den umzukonfigurierenden Monitoringauftrag aus.

Abbildung A.6: Monitoringauftrag umkonfigurieren.

3. Bestätigen Sie die Auswahl mit der Schaltfläche *Weiter >*. (Die Schaltfläche *Abbrechen* bricht den Umkonfigurationsvorgang ab.)
4. Über das darauffolgende Dialogfenster können Sie die neuen Parameter des aktiven Monitoringauftrages einstellen. Das linke Teilfenster enthält eine Beschreibung des Monitoringauftrages und eine Anleitung für dessen Parameter. Das rechte Teilfenster enthält die Eingabefelder für die Parameter.

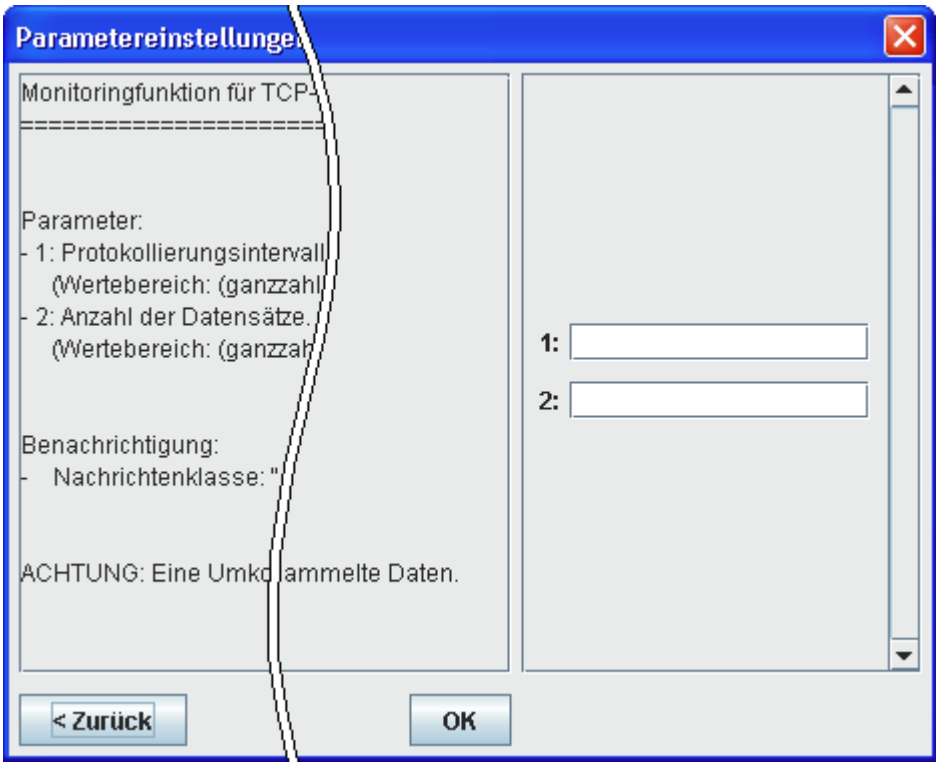

Abbildung A.7: Monitoringparameter einstellen.

5. Bestätigen Sie die Parametereingabe mit der Schaltfläche *OK*. (Die Schaltfläche *< Zurück* führt zum vorherigen Dialogfeld.)

A.7 SOAP-Nachrichten protokollieren

Für Debuggingzwecke ermöglicht die Manageranwendung, SOAP-Nachrichten, die zwischen Manageranwendung und Agenten ausgetauscht werden, auf der Konsole zu protokollieren.

1. Aktivieren Sie im Menü *Aktion* den Menüpunkt *SOAP-Nachrichten protokollieren*.

B Webserviceschnittstelle des Agenten

Die Webserviceschnittstelle auf Seite des Agenten ist die Kommunikationsschnittstelle zwischen Manager und dem Agenten. Sie definiert Methoden für die Generierung und Konfiguration von Monitoringinstanzen durch einen Manager.

Konfigurationsmethoden

```
int public subscribe(long guid, String monitorfunction,
    String[] parameterlist)
    throws IServiceConfig.InvalidGuidException,
    IServiceConfig.NoFreeEventHandlesAvailableException,
    IServiceConfig.MaxRegisteredManagersReachedException,
    IServiceConfig.InvalidParameterListException,
    IServiceConfig.UnknownMonitoringfunctionException,
    IServiceConfig.ErrorInMonitoringfunctionException,
    IServiceConfig.CallbackInterfaceBindErrorException,
    IServiceConfig.UnknownErrorException
```

Meldet einen neuen Monitoringauftrag beim Agenten an. Der Agent erstellt hierfür eine neue Instanz des Monitoringobjekts.

Parameter:

`guid`	-	Globale eindeutige ID des Managers.
`monitorfunction`	-	Vollqualifizierter Name eines Monitoringobjekts.
`parameterlist`	-	Liste mit benötigten Parametern des Monitoringobjekts.

Rückgabewert:

Ein Handle zur eindeutigen Identifikation der Monitoringinstanz.

```
void unsubscribe(long guid, int handle)
    throws IServiceConfig.InvalidGuidException,
    IServiceConfig.InvalidEventHandleException
```

Meldet eine aktive Monitoringinstanz beim Agenten ab.

Parameter:

`guid`	-	Globale eindeutige ID des Managers.
`handle`	-	Handle einer Monitoringinstanz.

```
void reconfigure(long guid, int handle, String[]
    parameterlist)
    throws IServiceConfig.InvalidGuidException,
    IServiceConfig.InvalidEventHandleException,
    IServiceConfig.InvalidParameterListException
```

Konfiguriert eine aktive Monitoringinstanz im Agenten um.

Parameter:

`guid`	-	Globale eindeutige ID des Managers.
`handle`	-	Handle einer Monitoringinstanz.
`parameterlist`	-	Liste mit benötigten Parametern des Monitoringobjekts.

Hilfsmethoden

```
String[] getMonitorObjects()
```

Liefert eine Liste aller auf dem Agenten verfügbaren Monitoringobjekte zurück.

Rückgabewert:

Array mit vollqualifizierten Namen der Monitoringobjekte.

```
String[] getParameterInformation(String monitorfunction)
    throws IServiceConfig.InvalidParameterListException
```

Liefert Informationen über die benötigten Parameter eines Monitoringobjekts zurück, die im Parameter `parameterlist` der Anmelde- und Umkonfigurationsmethode übergeben werden.

Parameter:

`monitorfunction` - Vollqualifizierter Name eines Monitoringobjekts.

Rückgabewert:

Array mit zwei Elementen ([0] – Anzahl der Elemente im Parameter `parameterlist` der Anmelde- und Umkonfigurationsmethode; [1] – Textuelle Beschreibung des Monitoringobjekts und der `parameterlist`-Elemente).

```
String[][] getRegisteredMonitoringInstances(long guid)
    throws IServiceConfig.InvalidGuidException
```

Ermittelt alle aktiven Monitoringinstanzen, die im Auftrag eines Managers auf dem Agenten laufen.

Parameter:

`guid` - Globale eindeutige ID des Managers.

Rückgabewert:

Zweidimensionales Array mit Handles der Monitoringobjektinstanzen und vollqualifizierten Namen der zugehörigen Monitoringobjekte ([n][0] – Handle; [n][1] – vollqualifizierter Name).

Exceptions

```
public InvalidGuidException()
```

Signalisiert eine ungültige Manager-GUID.

```
public InvalidEventHandleException()
```

Signalisiert ein ungültiges Handle für eine Monitoringinstanz.

```
public NoFreeEventHandlesAvailableException()
```

Signalisiert, daß keine freien Handles für die Anmeldung weiterer Monitoringaufträge mehr vorhanden sind.

`public` **`MaxRegisteredManagersReachedException`**`()`

Signalisiert, daß Manager, für die kein laufender Monitoringthread auf dem Agenten existiert, keine Monitoringaufträge anmelden können. Dies resultiert daraus, daß die maximale Anzahl erlaubter Monitoringthreads bereits erreicht ist.

`public` **`InvalidParameterListException`**`()`

Signalisiert, daß die Daten in dem Parameter `parameterlist` fehlerhaft sind oder die Anzahl der Elemente falsch ist.

`public` **`UnknownMonitoringfunctionException`**`()`

Signalisiert, daß ein Manager im Methodenaufruf eine dem Agenten unbekannte Monitoringfunktion angegeben hat.

`public` **`ErrorInMonitoringfunctionException`**`()`

Signalisiert, daß der Agent beim Zugriff auf ein Monitoringobjekt einen Fehler entdeckt hat und dieses deswegen nicht instanziieren kann. Möglicherweise ist das angesprochene Monitoringobjekt fehlerhaft implementiert.

`public` **`CallbackInterfaceBindErrorException`**`()`

Signalisiert, daß der Agent keine Verbindung zur Callback-Schnittstelle des Webservices des Managers aufbauen konnte, über welche der Manager Benachrichtigungen empfängt.

`public` **`UnknownErrorException`**`()`

Signalisiert, daß ein unbekannter Fehler aufgetreten ist.

C Webserviceschnittstelle des Managers

Die Webserviceschnittstelle des Managers dient der Benachrichtigung desselben durch Agenten. Sie definiert eine Methode, über die ein Agent einem Manager Benachrichtigungen zustellen kann.

Benachrichtigungsmethoden

```
void pushNotification(long guid, int handle, String
    messageClass, String[] data)
```

Nimmt Benachrichtigungen von Agenten entgegen.

Parameter:

`guid`	-	Globale eindeutige ID des Managers.
`handle`	-	Handle einer Monitoringinstanz.
`messageClass`	-	Name einer Nachrichtenklasse.
`data`	-	Stringarray mit benachrichtigungsspezifischen Daten.

D Monitoringschnittstelle

Diese Schnittstelle legt fest, wie die einzelnen Monitoringobjekte mit dem Monitoringmodul der Agentenanwendung kommunizieren. Sie definiert, welche Methoden das Monitoringmodul für die Monitoringobjekte implementieren muß und welche Methoden die Monitoringobjekte implementieren müssen. Der Abschnitt *Methoden – Monitoringmodul* enthält die Methoden, die das Monitoringmodul den Monitoringobjekten zur Verfügung stellt. Der zweite Abschnitt *Methoden – Monitoringobjekt* enthält die Methoden, die das Monitoringmodul aufruft und jedes Monitoringobjekt implementieren muß.

Methoden – Monitoringmodul

```
public void notifyManager(IMonitorObjects
    monitorObjectInst, String messageClass, String[] data)
```

Nimmt Benachrichtigungen von Monitoringinstanzen entgegen und leitet diese an das Benachrichtigungsmodul weiter.

Parameter:

`monitorObjectInst`	-	Referenz auf die aufrufende Monitoringinstanz.
`messageClass`	-	Name einer Nachrichtenklasse.
`data`	-	Stringarray mit benachrichtigungsspezifischen Daten.

```
public void removeInstance(IMonitorObjects
    monitorObjectInst)
```

Methode, über die sich aktive Monitoringinstanzen selbständig abmelden können. Diese Methode müssen Monitoringinstanzen aufrufen, bevor sie sich selbst beenden wollen.

Parameter:

`monitorObjectInst`	-	Referenz auf die aufrufende Monitoringinstanz.

Methoden – Monitoringobjekt

```
public <Constructor>(MonitoringThread
    parentMonitoringThread, String[] parameterlist)
    throws userExceptions.InvalidParameterListException
```

Der Konstruktor eines Monitoringobjekts übernimmt die Initialisierung einer neuen Instanz eines Monitoringobjekts. Der Konstruktor muß die übergebenen Daten für die Initialisierung auf Gültigkeit überprüfen und im Fehlerfall die Initialisierung mit einer Exception abbrechen.

Parameter:

`parentMonitoringThread`	-	Referenz auf den aufrufenden Monitoringthread.
`parameterlist`	-	Liste mit benötigten Parametern für die Initialisierung des Monitoringobjekts.

```
public static String[] getParameterInformation()
```

Liefert Informationen über Parameter eines Monitoringobjekts zurück, die für die Anmeldung und Umkonfiguration benötigt werden.

Rückgabewert:

Array mit zwei Elementen ([0] – Anzahl der Elemente im Parameter `parameterlist` des Konstruktors; [1] – Beschreibung des Monitoringobjekts und der `parameterlist`-Elemente).

```
public void monitor_run()
```

Diese Methode führt das Monitoring aus und veranlaßt Benachrichtigungen an Manager. Sie wird periodisch von dem Monitoringthread aufgerufen, der die Instanz des Monitoringobjekts erstellt hat.

```
public void alterConfig(String[] parameterlist)
    throws userExceptions.InvalidParameterListException
```

Ändert die Konfiguration einer aktiven Monitoringinstanz.

Parameter:

`parameterlist` - Liste mit benötigten Parametern der Monitoringinstanz.

`public void` **`cleanup`**`()`

Diese Methode führt Aufräumarbeiten vor der Abmeldung der Monitoringinstanz aus. Sie wird vor einer Abmeldung von dem Monitoringthread aufgerufen, der die Instanz erstellt hat.

Exceptions

`public` **`InvalidParameterListException`**`()`

Signalisiert, daß die Daten in dem Parameter `parameterlist` fehlerhaft sind oder die Anzahl der Elemente falsch ist.

Literaturverzeichnis

[ABFG04] D. AUSTIN, A. BARBIR, C. FERRIS und S. GARG: *Web Services Architecture Requirements*. W3C Note, W. W. Grainger Inc., Nortel Networks Inc., IBM, Intel Corporation, Februar 2004. http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211.

[BHL99] T. BRAY, D. HOLLANDER und A. LAYMAN: *Namespaces in XML*. W3C Recommendation, Textuality, Hewlett-Packard, Microsoft, Januar 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114.

[BL05] D. BOOTH und C. K. LIU: *Web Services Description Language (WSDL) Version 2.0 Part 0: Prime*. W3C Working Draft, W3C Fellow / Hewlett-Packard, SAP Labs, August 2005. http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803.

[BLFM98] T. BERNERS-LEE, R. FIELDING und L. MASINTER: *Uniform Resource Identifiers (URI): Generic Syntax*. RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[BM04] P. V. BIRON und A. MALHOTRA: *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation, Kaiser Permanente, Microsoft, Oktober 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028.

[BPSM+04] T. BRAY, J. PAOLI, C. M. SPERBERG-MCQUEEN, E. MALER und F. YERGEAU: *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C Recommendation, Textuality, Netscape, Microsoft, W3C, Sun Microsystems Inc., Februar 2004. http://www.w3.org/TR/2004/REC-xml-20040204.

[BW02] U. BLUMENTHAL und B. WIJNEN: *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*. RFC 3414, Lucent Technologies, Dezember 2002.

[CC05] S. CHISHOLM und K. CURRAN: *Netconf Event Messages*. Internet-Draft, Nortel, Juli 2005. http://www.ietf.org/internet-drafts/draft-chisholm-netconf-event-00.txt.

[CDK02] G. COULOURIS, J. DOLLIMORE und T. KINDBERG: *Verteilte Systeme – Konzepte und Design*. Addison-Wesley, 3. Auflage, 2002. ISBN 3-8273-7022-1.

[CFSD90] J. CASE, M. FEDOR, M. SCHOFFSTALL und J. DAVIN: *Simple Network Management Protocol (SNMP)*. RFC 1157, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, Mai 1990.

[CHL+05] R. CHINNICI, H. HAAS, A. LEWIS, J.-J. MOREAU, D. ORCHARD und S. WEERAWARANA: *Web Services Description Language (WSDL) Version 2.0 Part 0: Prime*. W3C Working Draft, Sun Microsystems, W3C, TIBCO, Canon, BEA Systems, August 2005. http://www.w3.org/TR/2005/WD-wsdl20-adjuncts-20050803.

[CHvRR04] L. CLEMENT, A. HATELY, C. VON RIEGEN und T. ROGERS: *UDDI*. Committee Draft, Systinet, IBM, SAP AG, Computer Associates, Oktober 2004. http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf.

[CMRW96a] J. CASE, K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *Introduction to Community-based SNMPv2*. RFC 1901, SNMP Research Inc., Cisco Systems Inc., Dover Beach Consulting Inc., International Network Services, Januar 1996.

[CMRW96b] J. CASE, K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)*. RFC 1907, SNMP Research Inc., Cisco Systems Inc., Dover Beach Consulting Inc., International Network Services, Januar 1996.

[CMRW96c] J. CASE, K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)*. RFC 1905, SNMP Research Inc., Cisco Systems Inc., Dover Beach Consulting Inc., International Network Services, Januar 1996.

[CMRW96d] J. CASE, K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*. RFC 1902, SNMP Research Inc., Cisco Systems Inc., Dover Beach Consulting Inc., International Network Services, Januar 1996.

[CMRW05] R. CHINNICI, J.-J. MOREAU, A. RYMAN und S. WEERAWARANA: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Working Draft, Sun Microsystems, Canon, IBM, August 2005. http://www.w3.org/TR/2005/WD-wsdl20-20050803.

[DCFS87] J. DAVIN, J. CASE, M. FEDOR und M. SCHOFFSTALL: *Simple Gateway Monitoring Protocol*. RFC 1028, Proteon Inc., University of Tennessee at Knoxville, Cornell University, Rensselaer Polytechnic Institute, November 1987.

[EF03] A. EBERHART und S. FISCHER: *Web Services – Grundlagen und praktische Umsetzung mit J2EE und .NET*. Carl Hanser Verlag, 2003. ISBN 3-446-22530-7.

[Enn05] R. ENNS: *NETCONF Configuration Protocol*. Internet-Draft, Juniper Networks, Juni 2005. http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-07.txt.

[Fla99] D. FLANAGAN: *Java in a Nutshell*. O'Reilly & Associates Inc., 3. Auflage, November 1999. ISBN 1-56592-487-8.

[FW04] D. C. FALLSIDE und P. WALMSLEY: *XML Schema Part 0: Primer Second Edition*. W3C Recommendation, IBM, Oktober 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028.

[GB] E. GAMMA und K. BECK: *JUnit*. http://www.junit.org.

[GHM+03a] M. GUDGIN, M. HADLEY, N. MENDELSOHN, J.-J. MOREAU und H. F. NIELSEN: *SOAP Version 1.2 Part 1: Messaging Framework*. W3C Recommendation, Microsoft, Sun Microsystems, IBM, Canon, Juni 2003. http://www.w3.org/TR/2003/REC-soap12-part1-20030624.

[GHM+03b] M. GUDGIN, M. HADLEY, N. MENDELSOHN, J.-J. MOREAU und H. F. NIELSEN: *SOAP Version 1.2 Part 2: Adjuncts*. W3C Recommendation, Microsoft, Sun Microsystems, IBM, Canon, Juni 2003. http://www.w3.org/TR/2003/REC-soap12-part2-20030624.

[God05] T. GODDARD: *Using the Network Configuration Protocol (NETCONF) Over the Simple Object Access Protocol (SOAP)*. Internet-Draft, ICEsoft Technologies Inc., April 2005. http://www.ietf.org/internet-drafts/draft-ietf-netconf-soap-05.txt.

[HL04] T. HAUSER und U. M. LÖWER: *Web Services – Die Standards*. Galileo Computing, 2004. ISBN 3-89842-393-X.

[ISO89] ISO: *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management Framework*. International Standard ISO/IEC 7498-4, ISO, November 1989.

[ISO02] ISO: *Document Schema Definition Languages (DSDL) – Part 2: Regular-grammar-based validation – RELAX NG*. Draft International Standard ISO/IEC FDIS 19757-2, ISO, Dezember 2002.

[JS93] R. JANSSEN und W. SCHOTT: *SNMP – Konzepte, Verfahren, Plattformen*. DATACOM-Verlag, 1993. ISBN 3-89238-077-5.

[Koz05] C. M. KOZIEROK: *The TCP/IP Guide – TCP/IP Internet Standard Management Framework and SNMP Versions (SNMPv1, SNMPv2 Variants, SNMPv3)*, 2005. http://www.tcpipguide.com/free/t_TCPIPInternetStandardManagementFrameworkandSNMPVer.htm.

[LC05] E. LEAR und K. CROZIER: *Using the NETCONF Protocol over Blocks Extensible Exchange Protocol (BEEP)*. Internet-Draft, Cisco Systems, März 2005. http://www.ietf.org/internet-drafts/draft-ietf-netconf-beep-05.txt.

[McC96a] K. MCCLOGHRIE: *SNMPv2 Management Information Base for the Internet Protocol using SMIv2*. RFC 2011, Cisco Systems, November 1996.

[McC96b] K. MCCLOGHRIE: *SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2*. RFC 2012, Cisco Systems, November 1996.

[McC96c] K. MCCLOGHRIE: *SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2*. RFC 2013, Cisco Systems, November 1996.

[Mica] MICROSOFT: *DCOM (Distributed Component Object Model)*. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/dcom.asp.

[Micb] SUN MICROSYSTEMS: *Java 2 Standard Edition (J2SE)*. http://java.sun.com/j2se/index.jsp.

[Mit03] N. MITRA: *SOAP Version 1.2 Part 0: Primer*. W3C Recommendation, Ericsson, Juni 2003. http://www.w3.org/TR/2003/REC-soap12-part0-20030624.

[MK00] K. MCCLOGHRIE und F. KASTENHOLZ: *The Interfaces Group MIB*. RFC 2863, Cisco Systems, Argon Networks, Juni 2000.

[Moo02] K. MOORE: *On the use of HTTP as a Substrate*. RFC 3205, University of Tennessee, Februar 2002.

[MPS99a] K. MCCLOGHRIE, D. PERKINS und J. SCHÖNWÄLDER: *Structure of Management Information Version 2 (SMIv2)*. RFC 2578, Cisco Systems, SNMPinfo, TU Braunschweig, April 1999.

[MPS99b] K. MCCLOGHRIE, D. PERKINS und J. SCHÖNWÄLDER: *Textual Conventions for SMIv2*. RFC 2579, Cisco Systems, SNMPinfo, TU Braunschweig, April 1999.

[MR90] K. MCCLOGHRIE und M. ROSE: *Management Information Base for Network Management of TCP/IP-based internets*. RFC 1156, Hughes LAN Systems, Performance Systems International, Mai 1990.

[MR91] K. MCCLOGHRIE und M. ROSE: *Management Information Base for Network Management of TCP/IP-based internets (MIB-II)*. RFC 1213, Hughes LAN Systems Inc., Performance Systems International, März 1991.

[Oec] R. OECHSLE: *SNMP-Manager*. FH Trier - FB Design & Informatik. http://www.ainformatik.fh-trier.de/~oechsle/snmp.

[OMG] OMG: *CORBA (Common Object Request Broker Architecture)*. http://www.omg.org/technology/corba/corba3releaseinfo.htm.

[Pos02] *Datenkommunikation – VI. Netzwerkmanagement*. Hochschule für Technik, Zürich, 2002. http://pubwww.hsz-t.ch/~tpospise/Datenkomm/materialien/Datenkommunikation_6_Netzwerk_Management_V1.2.pdf.

[Pre02] R. PRESUHN: *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)*. RFC 3418, BMC Software Inc., Dezember 2002.

[PT87] C. PARTRIDGE und G. TREWITT: *High-level Entity Management System (HEMS)*. RFC 1021, BBN/NNSC, Stanford, Oktober 1987.

[RM90] M. ROSE und K. MCCLOGHRIE: *Structure and Identification of Management Information for TCP/IP-based Internets*. RFC 1155, Performance Systems International, Hughes LAN Systems, Mai 1990.

[Ros91] M. ROSE: *Convention for defining traps for use with the SNMP*. RFC 1215, Performance Systems International, März 1991.

[Sch99] J. SCHÖNWÄLDER: *Vorlesung Netzwerkmanagement*. Institut für Betriebssysteme und Rechnerverbund, TU-Braunschweig, 1999. http://www.ibr.cs.tu-bs.de/lehre/ws9900/nm.

[Sch02] J. SCHÖNWÄLDER: *Simple Network Management Protocol Over Transmission Control Protocol Transport Mapping*. RFC 3430, TU Braunschweig, Dezember 2002.

[Sch05] J. SCHÖNWÄLDER: *Characterization of SNMP MIB Modules*. In: *Proc. 9th IFIP/IEEE International Symposium on Integrated Network Management*, Seiten 615–628, Mai 2005.

[SS01] K. SCRIBNER und M. C. STIVER: *SOAP developer's guide – Spezifikation, XML, BizTalk-Server*. Markt+Technik Verlag, 2001. ISBN 3-8272-5944-4.

[Sta96] W. STALLINGS: *SNMP, SNMPv2 and RMON – Practical Network Management*. Addison-Wesley, 2. Auflage, 1996. ISBN 0-201-63479-1.

[TBMM04] H. S. THOMPSON, D. BEECH, M. MALONEY und N. MENDELSOHN: *XML Schema Part 1: Structures Second Edition*. W3C Recommendation, University of Edinburgh, Oracle Corporation, Lotus Development Corporation, Oktober 2004. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028.

[WB89] U. WARRIER und L. BESAW: *Common Management Information Services and Protocol over TCP/IP (CMOT)*. RFC 1095, Unisys Corporation, Hewlett-Packard, April 1989.

[Web] WEBMETHODS: *Glue Standard Edition*. http://www.webmethods.com/meta/default/folder/0000006047.

[WG05] M. WASSERMAN und T. GODDARD: *Using the NETCONF Configuration Protocol over Secure Shell (SSH)*. Internet-Draft, ThingMagic, ICEsoft Technologies Inc., April 2005. http://www.ietf.org/internet-drafts/draft-ietf-netconf-ssh-04.txt.

[WPM02] B. WIJNEN, R. PRESUHN und K. MCCLOGHRIE: *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*. RFC 3415, Lucent Technologies, BMC Software Inc., Cisco Systems Inc., Dezember 2002.