Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund

Diplomarbeit

Web Services im Internet Management

von
cand. inform. Oliver Bohle

Aufgabenstellung und Betreuung:
Prof. Dr. S. Fischer und Dipl.-Inform. F. Strauss

Braunschweig, den 10. September 2003

Erklärung

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

Braunschweig, den 10. September 2003

Kurzfassung

Webservices stellen eine Technologie dar, plattform- und systemunabhängig Daten auszutauschen. Sie finden dadurch auch eine immer stärkere Beachtung in kommerziellen Systemen, obwohl sich die gesamte Architektur noch in der Standardisierungsphase befindet. Aus diesem Grund wird in dieser Arbeit betrachtet, in welcher Art und Weise Webservices dazu geeignet sind, Netzwerkmanagementaufgaben mit dieser Technologie zu modellieren und welche Unterschiede es zum zur Zeit vorherrschenden Managementansatz in TCP/IP-basierten Netzen, SNMP, gibt.

Dazu werden zuerst die Grundlagen zu einem systematischen Vergleich der beiden Technologien beschrieben. Nach der Beschreibung der beiden Basistechnologien erfolgt die Erarbeitung einer konkreten Managementlösung anhand eines ausgewählten Szenarios in beiden Technologien, was bei den Webservices zu der Entwicklung eines Prototypen führt. Abschließend wird ein Vergleich der beiden Netzwerkmanagementlösungen vorgenommen.

Abstract

Web Services represent a technology to communicate in a system- and platformindependent manner. For this reason, they find a growing interest by commercial systems, even though their architecture is still being standardized. Therefore this thesis will show in which way Web Services are suitable for the realisation of network management tasks and which differences exist realated to the predominating network management framework in TCP/IP-based networks SNMP.

In the beginning the fundamentals of both technologies are described by a systematic comparison. This description of the base technologies is followed by the development of a concrete solution for a specific management problem on the basis of a chosen szenario in both technologies. This leads in the case of Web Services to the development of a prototype. Finally the specifics of both approaches are shown in a comparison.

[Hier wird später die Aufgabenstellung eingefügt.]

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

Das “Simple Network Management Protocol” (SNMP) stellt zumindest im Bereich IP-basierter Netze heute den Standard des Netzwerkmanagements dar. Es hat sich aber in der letzten Zeit mit der Entwicklung von XML und immer heterogener zusammengesetzten Netzwerken ein neuer Kommunikationsansatz entwickelt, nämlich Webservices. Dieser Ansatz stellt eine system- und plattformunabhängige Kommunikationstechnologie auf XML-Basis dar.

In dieser Arbeit soll systematisch untersucht werden, welche Vor- und Nachteile das herkömmliche SNMP-Netzwerkmanagementkonzept gegenüber einer Modellierung des Netzwerkmanagements mit der neuen Technologie Webservices hat.

Um einen Vergleich der beiden Ansätze vornehmen zu können, wird im Kapitel 2 eine Systematik eingeführt, anhand derer die jeweiligen Konzepte in Kapitel 4 und 6 beschrieben und im Anschluss in Kapitel 7 gegeneinander bewertet werden sollen. In den Kapiteln 3 und 5 werden die jeweiligen Ansätze grundlegend erläutert.

Für die Bewertung und Entwicklung eines Webservice-Prototypen wird zudem in Kapitel 2 ein Managementszenario beschrieben, welches im Fall von SNMP auf einer exemplarischen Basis gelöst wird, im Fall von Webservices führt die Lösung des Problems zu der Entwicklung eines Prototypen.

Alle erstellten Programme sowie frei erhältlichen Dokumente, die zur Entwicklung der Arbeit verwendet worden sind, finden sich auf der beiliegenden CD-ROM.

2. Grundlagen des Netzwerkmanagements

Mit dem Übergang des Internets von einem Forschungsnetz in ein kommerziell genutztes Kommunikationsnetz stieg die Anzahl der Hersteller und der angebotenen Netzwerkgeräte an. Da man nur über die Abgrenzung gegenüber seinen Konkurrenten einen dauerhaften Markterfolg erreichen kann, wurde der Aufbau des Kommunikationsnetzes immer heterogener. Mit einer immer heterogener werdenden Netzwerklandschaft stieg auch die Zahl der unterschiedlichen Programme, mit denen diese Ressourcen administriert werden konnten. Dabei ist zu beachten, dass die meisten Administrationswerkzeuge proprietäre Lösungen der Hersteller der Ressource waren und somit schon bei kleineren Netzwerken eine beachtliche Anzahl an Programmen dazu dienten, die Ressourcen zu überwachen.

Diesem isolierten Ansatz, in dem ein isoliertes Managementwerkzeug für jedes einzelne Managementproblem geschaffen wird, folgt der koordinierte Einsatz dieser Werkzeuge (s. Abb 2.1, S. 4). In diesem Ansatz werden die in ihrer Funktionserbringung nach wie vor isoliert arbeitenden Werkzeuge bezüglich der zu erbringenden Managementaufgabe im Einsatz koordiniert. Das heißt, dass sich die Werkzeuge in verschiedenster Art und Weise ergänzen können und zum Beispiel mit Hilfe von Skripten zusammengefasst werden.

Einen weiteren Schritt des Managements bedeutet der integrierte Ansatz. Bei der Integration von verschiedenen Managementanwendungen liefern die zu managenden Komponenten in einem heterogenen Umfeld Informationen, die in einer herstellerunabhängigen Weise interpretierbar sind. Dies beinhaltet in der Regel wohldefinierte Schnittstellen und Protokolle. Das Bewältigen der Komponentenvielfalt zählt zu den vordringlichsten Zielen des integrierten Managements.

Für die Entwicklung eines integrierten Managementansatzes sind herstellerübergreifende Modelle notwendig, um in heterogenen Umgebungen erfolgreich arbeiten zu können [1].

2.1. Modelle zur Beschreibung eines integrierten Netzwerkmanagementansatzes

Die wichtigsten Modelle zur Beschreibung eines integrierten Managementansatzes sind das Informationsmodell, das Organisationsmodell, das Kommunikationsmodell und das Funktionsmodell (vgl. [1] S.100).

Das **Informationsmodell** spezifiziert einen Beschreibungsrahmen für Managementobjekte. Aus Managementsicht ist es nicht nötig und durch die heterogenen Ressourcen auch in der Vielzahl dieser unmöglich, die internen Abläufe von Ressourcen zu kennen. Für das Management von Ressourcen reicht es aus, auf diese über ein Modell ihrer Hardware- und Softwareressourcen

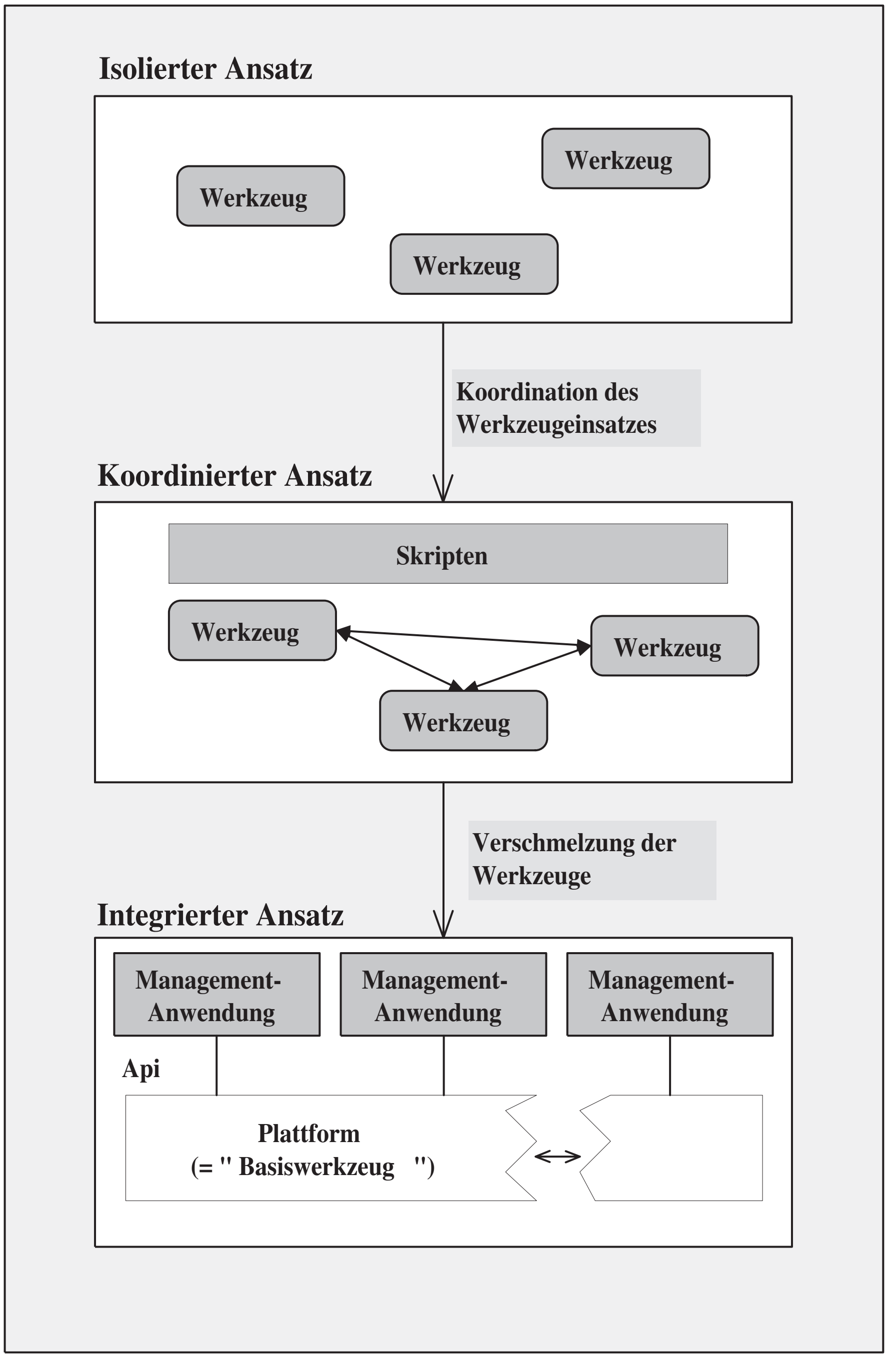

Abbildung 2.1.: Ansätze zur Realisierung von Managementlösungen ([1], S. 98)

auf einzelne Parameter zugreifen zu können. Diese Parameter dienen zum Beispiel zur Konfiguration oder zum Tuning oder bestehen aus Funktionen zum Starten, Stoppen oder Sichern. Managementobjekte stellen somit die Eigenschaften der Ressourcen dar, auf denen das Management operiert. Das Informationsmodell muss gewährleisten, dass eine eindeutige Identifizierung eines Managementobjektes möglich ist. Weiterhin wird im Informationsmodell beschrieben, wie das Objekt zusammengesetzt ist, wie es sich verhält, wie es manipuliert werden kann, welche Beziehungen zu anderen Managementobjekten bestehen und wie es über das Managementmodell angesprochen werden kann. Somit werden vom Informationsmodell sowohl der Modellierungsansatz (z.B. Entity Relationship, Datentypenansatz (IAB), Objektorientierung), als auch die Syntax für eine eindeutige Beschreibung von Managementinformationen festgelegt.

Das **Organisationsmodell** einer Managementarchitektur legt die Akteure, ihre Rollen und die Grundprinzipien ihrer Kooperation fest ([1], S. 105). Bei den Kooperationsmodellen der Komponenten gibt es zwei unterschiedliche Verfahren: zum einen das in der OSI eingesetzte Agent-Manager-Modell, zum anderen den Peer-To-Peer Ansatz. Im ersten Modell, auch asynchrone Kooperationsform genannt, ist eine strikte Trennung zwischen den Managementsystemen, die das Management des Netzes betreiben, und den ausführenden Einheiten, den Agenten, gegeben. Ein Agent ist dabei aber nicht an einen Manager gebunden, sondern kann die verschiedenen Manager des Managementsystems bedienen. Bei dem zweiten, symmetrischen Kooperationsmodell wird ein so genannter Peer-To-Peer Ansatz verfolgt([1], S. 105). Hier sind alle Systeme prinzipiell gleichberechtigt, sie können sowohl selber Managementfunktionen auf dem Netz ausführen, als auch Managementobjekte bereitstellen. Es findet also eine flexible, wechselseitige Auftragsbeziehung und ein Informationsaustausch in beiden Richtungen statt. Bei alle dem ist aber zu beachten, dass das Organisationsmodell einer Managementarchitektur die Aufbau- und Ablaufsituation eines Netzwerkes nicht präjudizieren darf.

Das **Kommunikationsmodell** einer Managementarchitektur legt die Konzepte zum Austausch der Managementinformationen zwischen den Akteuren fest. Somit sind folgende Aspekte im Kommunikationsmodell zu beachten: Festlegung der kommunizierenden Partner, Festlegung des Kommunikationsmechanismus für Controlling, Monitoring, Ereignismeldungen, Definition der Syntax und Semantik der Protokoll-Datenstrukturen, Einbettung der Managementprotokolle in die Protokollhierarchie der zugrunde liegenden Kommunikationsarchitektur.

Das **Funktionsmodell** einer Managementarchitektur unterteilt den Gesamtaufgabenkomplex des Managements in verschiedene Funktions-Teilbereiche. Eine solche Unterteilung ist zum Beispiel die Management-Definition der ISO für die OSI. Diese definiert folgende fünf Hauptaufgaben des Managements.

"OSI Management Functional Areas"([3] S.151, [10] S.3f):

Configurations Management: Management der Namen und Konfigurationen für alle Netzwerkelemente. Die Bezeichnung Netzwerkelemente bezieht sich in diesem Fall auf Hardware und Software zugleich. Zudem ist die Aufgabe des Konfigurationsmanagements, das Beobachten (Monitoring) der Zustände der Netzwerkelemente und ihrer Komponenten.

Performance Management: bestimmt die effektive Nutzung des Netzwerks, die Effektivität der bereitgestellten Dienste und die Nutzung der angebotenen Dienste. Dieser Bereich umfasst das Beobachten der Datenströme in einem Netzwerk, um in diesem Bereich Überlastungen oder Überkapazitäten zu erkennen.

Fault Management: Erkennung, Isolierung und Behebung von Netzwerkproblemen. Um die zuverlässige Operation eines Netzwerkes zu gewährleisten, muss ein Netzwerkmanager sowohl das Netzwerk als Ganzes, als auch die einzelnen wichtigen Komponenten überwachen. Falls ein Fehler auftritt, sollte er schnell eingrenzbar sein, um geeignete Maßnahmen zu ergreifen, so dass der Fehler einen so kleinen Einfluss wie möglich auf das laufende System hat.

Security Management: Zugangskontrolle und Sicherung der Daten auf dem Netzwerk vor Verfälschung und Abhören. Zudem kann die Verwaltung von Schlüsseln und den dazugehörenden Instanzen von kryptographischen Verfahren zum Sicherheitsmanagement gezählt werden.

Accounting: umfasst die Messung der Benutzung einer Ressource durch bestimmte Gruppen, Nutzer und die dazugehörige Berechnung der angefallenen Kosten.

Diese Funktionsbereiche gliedern sich wiederum in lokale[1] und globale[2] Managementfunktionen, je nachdem, ob die Funktion auf ein einzelnes System oder das Kommunikationsnetz gerichtet ist.

Um den Vergleich der Ansätze des Netzwerkmanagements durch SNMP und Webservices zu systematisieren, kann man die verschiedenen Modelle, mit denen die Ansätze beschrieben werden, zur Strukturierung des Vergleiches verwenden.

2.2. Managementszenario: Lokales Netzwerk

Da das Netzwerkmangement ein sehr weit gefasster Bereich ist, wird im Folgenden ein Managementszenario beschrieben, anhand dessen ausgesuchte Managementaufgaben herausgehoben werden. Diese Aufgaben dienen dann als Ausgangspunkt zur Entwicklung von Netzwerkmanagementlösungen mittels SNMP und Webservices.

Ein im Management von Netzwerken häufig vorkommendes Szenario besteht aus einem lokalen Netzwerk mit Anschluss an das Internet (vgl. Abbildung 2.2). In diesem Netzwerk sind zum einen die Arbeitsplatz-PCs enthalten, zum anderen Server, die sowohl Dienste extern als auch intern im lokalen Netzwerk bereitstellen. Ein Teil der anfallenden Managementaufgaben tritt im Bereich des Routings und der Dienstbereitstellung auf. Hier fallen während des Normalbetriebs vor allem Monitoringaufgaben an, um Fehler und Performanceprobleme frühzeitig zu erkennen. Zudem sollte dieser Teil des Netzwerkes durch Managementwerkzeuge gut zu warten (Controlling) sein, damit nötige Konfigurationsänderungen rasch und konsistent durchgeführt werden können. Typische Dienste eines solchen Netzwerkes sind in der Abbildung 2.2 mit den Datei-, Mail- und Webserverdiensten dargestellt. Diese speziellen Dienstserver sind durch eine externe und eine interne Firewall gegen Angriffe geschützt.

[1] z.B. Generieren u. Parametrisieren von Netzsoftware, Anlegen, Pflegen Netzrelevanter Datenbestände (Netzverzeichnisse, Routingtabellen), Einschalten, Bedienen der Netzhardware, ...

[2] z.B. Funktionstests der Verbindungen im Netz, statische Verfassung des Datenverkehrs, ...

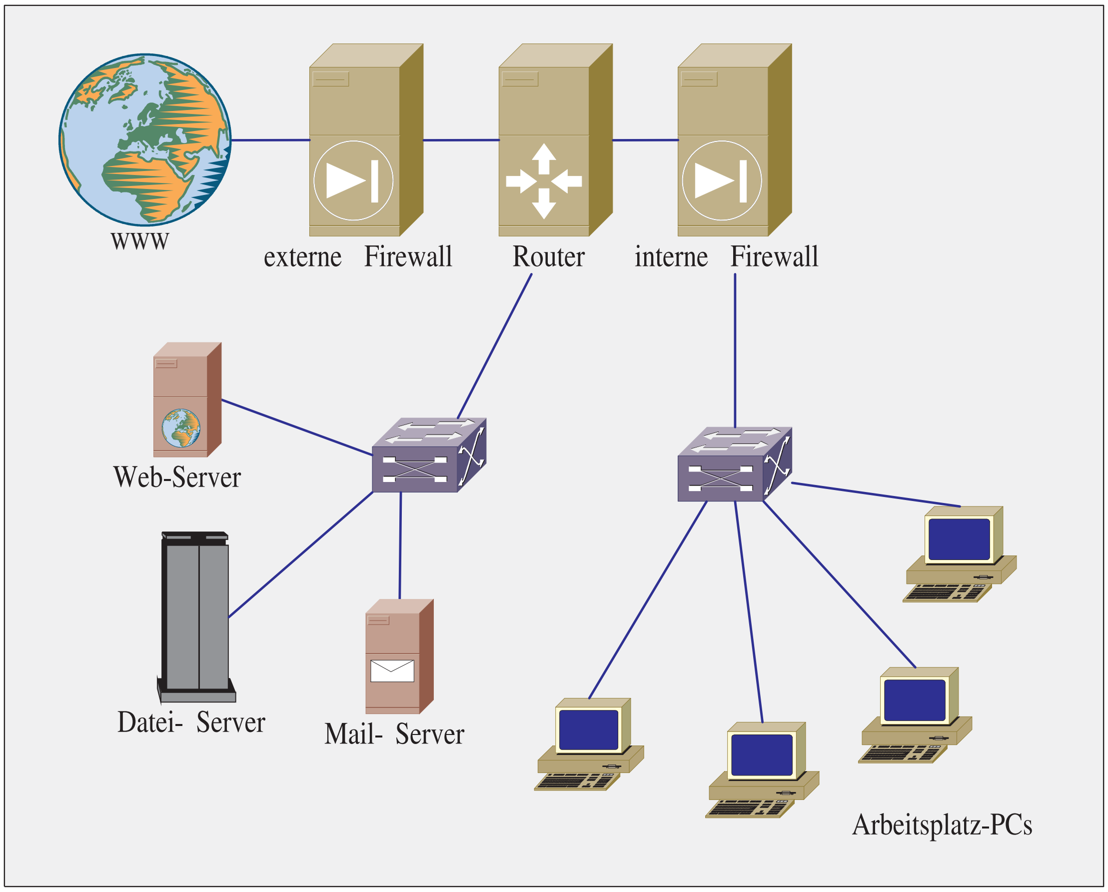

Abbildung 2.2.: Beispiel einer Konfiguration eines Lokalen Netzwerks

Für das Management eines solchen Netzes ergibt sich folgendes Anforderungsprofil:

- Hohe Verfügbarkeit während der Hauptarbeitszeit
- Bei Einsatz eines extern verfügbaren Internetangebotes: hohe Verfügbarkeit des Webservers mit entsprechender Bandbreite und Dienstgüte.
- Tägliche Sicherung der unternehmensrelevanten Daten.

Aus diesem Anforderungprofil ergeben sich die Managementaufgaben:

- Überwachung und Optimierung der Performance von Netzwerk und Servern
  - Überwachung der Auslastungssituationen der Server,
  - Überwachung der Auslastungssituation des Netzwerkes,
- Controlling der Server,

  - Konfigurationsmanagement, Umsetzung der Optimierungsergebnisse,

- Sicherung der Datenbestände der Server.

2.2.1. Ausschnitt: Schnittstellen

In den Bereich des Schnittstellenmanagements fällt vor allem die Auslastungsüberwachung, Kontrolle und Konfiguration der Schnittstellen. Die benötigten Parameter und Aufgaben sind:

Performance Monitoring

- Gesendete/Empfangene Pakete pro Schnittstelle,
- Anzahl der fehlerhaften Pakete pro Schnittstelle.

Status

- Adressbindungen (Hardware-Adressen, protokollspezifische Adressen, Netzwerkmasken) pro Schnittstelle,
- Status der Schnittstellen (aktiv, inaktiv, ...).

Events, Actions

- Statuswechsel einer Schnittstelle,
- Überschreitung der prozentualen Fehlerhäufigkeit einer Schnittstelle in einem definierten Zeitintervall.

Configuration

- Hinzufügen, Entfernen, Modifizieren von Schnittstellen.

Die Aufgaben des lokalen Netzwerkes sollen als Grundlage zur Entwicklung des Webservice-Prototypen dienen. Um das Hauptaugenmerk auf die Umsetzung einer Managementaufgabe mittels Webservices zu richten, wird der Funktionsumfang des Managementwerkzeuges eingeschränkt. Dazu wird die Betrachtung im Folgenden auf Aufgaben, die die Schnittstellen eines Systems betreffen, beschränkt.

3. Vorhandene Möglichkeiten des Netzwerkmanagements

Im Folgenden soll näher auf die vorhandenen Möglichkeiten zum Managen von Rechnernetzen eingegangen werden. Nach einem allgemeinen Einstieg wird das SNMP-Rahmenwerk als Grundlage für spätere Vergleiche genauer betrachtet.

3.1. Verschiedene Werkzeuge

Mit der zunehmenden Größe von Netzwerken ist es notwendig geworden diese mit effektiven Hilfsmitteln zu überwachen (Monitoring) und zu administrieren. Dies führte in der Entwicklung dieser Hilfsmittel zuerst zu Werkzeugen, die einzelne Dienste in den unteren drei Schichten des Internet-Protokollstacks mit Hilfe des ICMP-Protokolls überwachen. Diese Art der Überwachung von Netzwerkressourcen nennt man auch “aktives Management”.

Eine große Anzahl von Managementproblemen kann aber auch durch so genanntes “passives Management” erkannt werden. Der passive Ansatz des Managements beruht darauf, Pakete, die lokal empfangen werden, zu analysieren und anhand der Paketdaten Rückschlüsse auf den Zustand des Netzes zu treffen.

3.1.1. ICMP

ICMP ist ein Steuerprotokoll für Routinginformationen, das über verschiedene Ereignisse informiert. Haupteinsatzgebiet ist das Mitteilen von unerwarteten Problemen beim Transport von Datenpaketen im Internet. Kann ein Router sein Ziel zum Beispiel nicht finden, so signalisiert er dieses mit der ICMP-`"DESTINATION UNREACHABLE"`-Nachricht an den Versender des Paketes.

3.1.2. Ping, Traceroute

Ping und Traceroute sind Werkzeuge, die ICMP benutzen, um Informationen über das Routing herauszufinden.

Ping sendet ein ICMP `echo`-Request Paket zu einer IP-Adresse und wartet die Antwort ab. Falls ein `echo`-Reply zurückkommt, hat man zum einen die Information, dass der Zielrechner erreichbar ist, zum anderen die Information, wie lange der Transport der Daten zwischen den einzelnen Punkten dauert. Man hat aber keinerlei Hinweise darüber, wo ein Problem vorliegen könnte, falls der Request nicht beantwortet wird. Zudem kann man keinerlei Rückschlüsse auf den Weg des Paketes ziehen.

Um dieses zu ermöglichen, wird das Programm Traceroute verwendet. Es macht sich die Eigenschaften des ICMP-Protokolls zunutze. So werden UDP-Pakete mit aufsteigender TTL gesendet. Jeder Server, der den TTL in einem Paket auf Null setzt, sendet diesem eine ICMP-Fehlermeldung. Durch dieses Vorgehen kann jetzt der Pfad des Datenpakets nachvollzogen werden.

Mit Ping konnte also nur festgestellt werden, ob der Zielrechner erreichbar ist, mit Traceroute kann man eventuell im Fehlerfall feststellen, an welcher Stelle der Fehler auftritt [6, 11].

3.2. SNMP

Anfang der 90er Jahre begann die Entwicklung eines standardisierten integrierten Managementansatzes. In diesem Prozess, in den sich dann auch das IAB einschaltete, gab es drei konkurrierende Ansätze. HEMS wurde als Forschungsprojekt begonnen und entwickelte sich nicht über diesen Status hinaus. Die beiden diskutierten und weiterverfolgten Ansätze waren SGMP und CMOT[1], ein Ansatz, der auf dem OSI-Protokollstack aufbaut. Im Entwicklungsprozess wurde dem CMOT-Ansatz der Durchbruch auf längere Zeit zugetraut. Das Problem lag aber in der Umsetzung des OSI-Protokollstacks auf den Internetstack. SGMP hatte den Vorteil, ein Protokoll zu sein, welches schon erfolgreich implementiert und benutzt wurde. Um diese beiden Entwicklungsprozesse zusammenzuführen, wurde eine Arbeitsgruppe gebildet. Parallel dazu entstand aus SGMP das Management Protokoll SNMP. Im April 1989 wurde SNMP dann “de-facto” Management Standard für TCP/IP-basierte Netzwerke [6].

Im Mai 1990 wurde in den RFCs 1157 (aktuell RFC 1213) und 1155 [12, 8, 13][2] der Grundstein zu einer systematischen Überwachung und Verwaltung eines Rechnernetzes durch SNMP gelegt. In diesen Dokumenten wird das Informations- und Strukturmodell eines TCP/IP-basierten Netzwerkmanagementansatzes beschrieben. Wie schon in der Einleitung dargestellt, gab es zu Anfang der Entwicklung eines Netzwerkmanagementstandards durchaus divergierende Meinungen darüber, in welcher Form und mit welcher Architektur ein Managementansatz entstehen sollte. Die ersten Entwürfe waren noch davon geprägt, dass das Informationsmodell mit SMI/MIB sowohl kompatibel zu SNMP als kurzfristige Lösung als auch zu einem OSI-Managementansatz als langfristige Lösung sein sollte. Es hat sich aber gezeigt, dass die Unterschiede in den Architekturen der beiden Standards größer waren als erwartet, und so hat man mit dieser Erkenntnis die MIB-II entwickelt, um auf die entstandenen operationalen Bedürfnisse des Netzwerkmanagements einzugehen.Davon abgesehen sollte eine Kompatibilität zu den beiden Standards zugunsten der Unterstützung des SNMP-Rahmenwerks gewährleistet werden.

Durch die Erfahrung mit diesen Grundlagen wurde dann in den RFCs 1441 bis 1452 die erweiterte Version SNMPv2 definiert, in der neben zusätzlichen Protokollfunktionen vor allem Sicherheitsmechanismen eingeführt werden sollten. Diese erwiesen sich aber im Betrieb als zu kompliziert und es wurde eine reduzierte Version unter der Bezeichnung SNMPv2c (classic) entwickelt. Da aber SNMPv1 und SNMPv2 inkompatibel zueinander waren, hat sich SNMPv2 nie wirklich durchgesetzt. Um das Problem der Sicherheit des Netzwerkmanagementprotokolls zu lösen, ist dann mit den RFCs 2271 bis 2275 SNMPv3 vorgeschlagen worden. SNMPv3 stellt

[1]CMIP over TCP, CIMP-OSI Common Management Information Protocol
[2]RFC 1157 SNMPv1, RFC 1155 MIB-I, RFC 1213 MIB-II

zwischen der Transportschicht und der jeweilige SNMP-Implementierung eine neue Schicht mit verschiedenen Sicherheitsmechanismen zur Verfügung (vgl. Kapitel 3.2.5, RFCs 3410-3419).

Im Folgenden wird die SNMP-Architektur systematisch nach den Vorgaben aus Kapitel 2.1 beschrieben.

3.2.1. Das Organisationsmodell

Die Struktur des SNMP-Netzwerkmanagements beruht auf einer asynchronen Kooperationsform der Teilnehmer, dem Client-Server-Prinzip. Dies bedeutet, dass auf der einen Seite die Agenten die Informationen bereitstellen, auf der anderen Seite die Manager die Managementinformationen abfragen, manipulieren und weiterverwenden (vgl. Abb. 3.1).

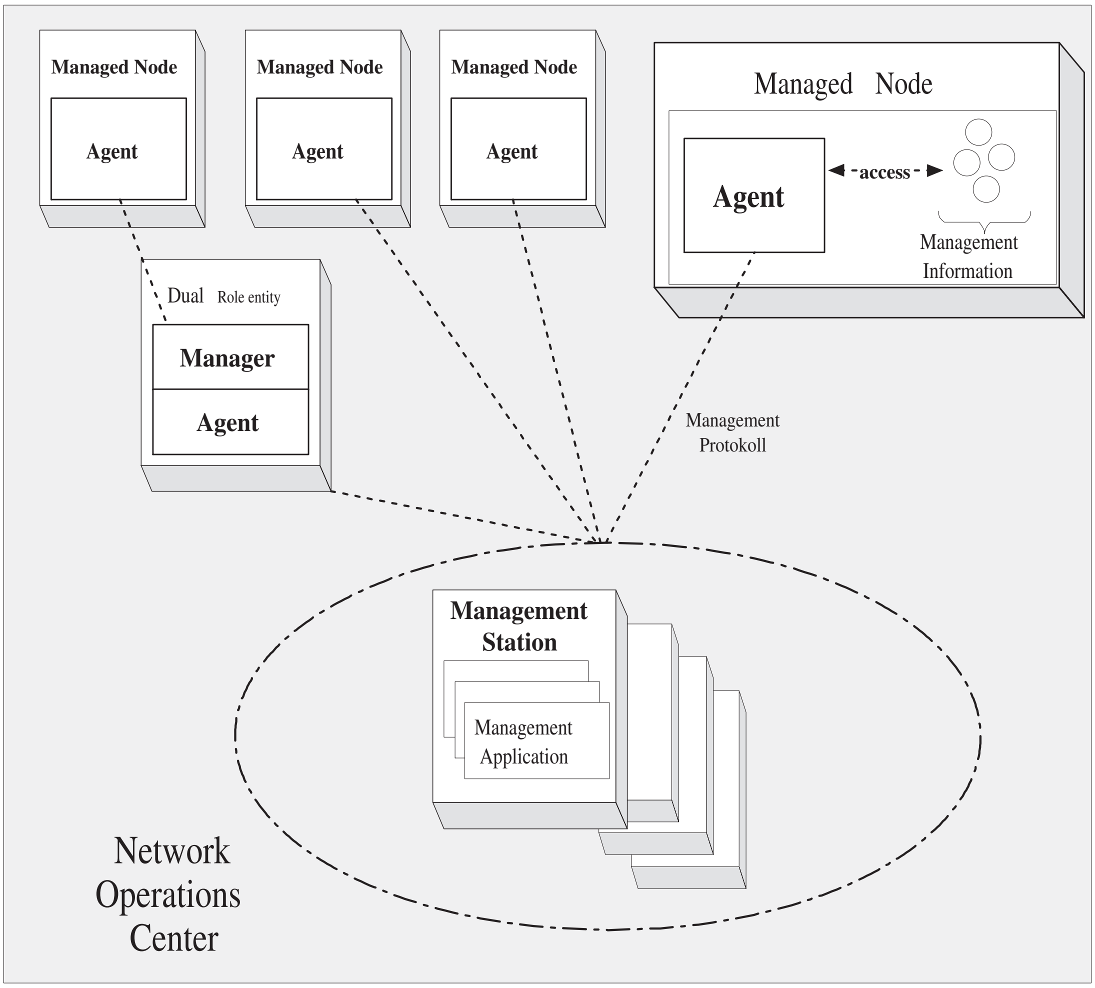

Abbildung 3.1.: Das SNMP-Modell

Das Organisationsmodell besteht dabei im Wesentlichen aus vier Komponenten:

Verwaltete Knoten: so genannte “Managed Nodes”, sind Komponenten, die von sämtlichen im Netzwerk vorhandenen managebaren Systemen (Hosts, Gateways, Bridges, Hubs, Multiplexer, Switches, ...) gebildet werden. Eines der fundamentalen Axiome bei der Entwicklung des SNMP-Modells im Bezug auf Managed Nodes war: *“The impact of adding network management to managed nodes must be minimal, reflecting a lowest common denominator.”* ([6], S.71), demnach soll der Knoten mit so wenig Managementaufgaben wie möglich belastet werden. Diese Aufgaben übernimmt ein Prozess, der “SNMP-Agent”.

Management Station: führt die Verwaltung der Knoten durch. Durch das bei den Managed Nodes angesprochene fundamentale Axiom werden hier die Managementfunktionen implementiert. Diese Komponente führt die Hauptrechenlast der Managementanwendung durch. Mehrere dieser Managementstationen, die für einen bestimmten Bereich zuständig sind, können organisatorisch zu einem “Network Operations Center” zusammengefasst werden.

Managementinformationen: sind die Informationen, die zum Management eines “Verwalteten Knotens” nötig sind. Diese Informationen sind in der “Management Information Base” (MIB) zusammengefasst und strukturiert. Über den Agenten können diese Informationen manipuliert und ausgelesen werden. Eine Manipulation sollte dann eine Verhaltensänderung der Ressource zur Folge haben.

Managementprotokoll: ist das Applikationsprotokoll für die Managementinformationen. Beim SNMP-Modell wird SNMP als Protokoll verwendet.

Jeder “verwaltete Knoten” beschreibt seinen Zustand selbst, indem er Managementinformationen in Form von Variablen, auch Objekte genannt, vorhält und durch einen Agenten manipulierbar macht. Diese Objekte können dann von der Management Station über das SNMP-Protokoll erfragt oder gesetzt werden. Mit diesen Werten kann die Management Station dann Aussagen über den Zustand des Netzwerkes und seiner Ressourcen/Geräte treffen und regelnd eingreifen. Damit das Management auch bei heterogenen Netzwerkarchitekturen erfolgen kann, sind mit der Bindung an das SNMP-Modell durch die “Structure of Management Information” (SMI) (vgl. Kap. 3.2.2) auch die Bereitstellung und Beschreibung von Managementinformationen in einer hersteller- und plattformunabhängigen Weise vorgegeben. Sämtliche mit den Objekten realisierten Managementinformationen, auch die proprietären der verschiedenen Hersteller, sind in so genannten “Management Information Bases” (MIBs) organisiert.

Es ist keine strikte Trennung zwischen Agenten und Managern vorgeschrieben, so kann auf einem Gerät beides existieren. Ein Beispiel für ein solches Gerät ist ein SNMP-Proxy, der die Informationen eine Teilnetzes gebündelt und vorausgewertet an die Rechner des Netzwerk-Operations-Zentrums weiterleitet.

3.2.2. Das Informationsmodell

Um das Informationsmodell von SNMP zu beschreiben, wird zuerst auf die “Structure of Managementinformation” (SMI) eingegangen, die das syntaktische Rahmenwerk der Informationsmodellierung darstellt. Danach wird näher auf die Strukturierung der Managementinformationen in der “Management Information Base” eingegangen.

3.2.2.1. SMI

Die “Structure of Management Information” ist die formale Sprache zur Definition von Datenmodellen, also konkreten Ausprägungen des Informationsmodells von SNMP. In dieser Sprache verfasste Dokumente werden als MIB Module bezeichnet und beschreiben die Form, Bedeutung und Zusammenhänge von “managed objects”. Die SMI liegt derzeit in den Versionen SMIv1 [13] und SMIv2 [15] vor. Die SMIv2 ist eine Erweiterung der SMIv1, wobei die Grundkonzepte aber erhalten geblieben sind. Im Folgenden soll die SMIv2 betrachtet werden.

Wie schon angesprochen, basiert das Internet-Management auf einem Client-Server-Modell, in dem die Clients die Management-Stationen darstellen und die Server die Management-Agenten. Die Management-Station stellt die Schnittstelle zum menschlichen Operateur dar. Der Agent stellt Mechanismen für den Zugriff des Managers auf die in den Agenten (vgl. 3.2.2.2) organisierten Managementinformationen über das Kommunikationsprotokoll SNMP zur Verfügung.

Die Objekte (Variablen), die ein Agent implementiert und die von den Managementstationen gelesen und geschrieben werden, bilden den Kern des SNMP-Informationsmodells. Diese Managementinformationen werden als eine Zusammenstellung von gemanagten Objekten in einem virtuellen Speicher aufgefasst. Dieser virtuelle Speicher wird als “Management Information Base”, MIB, bezeichnet. In diesen MIBs sind zusammenhängende Objekte wiederum als Module zusammengefasst.

Um diese Objekte plattform- und herstellerunabhängig zu beschreiben, wird ein abgewandelter Ausschnitt von ASN.1 verwendet.
Ein SMI Modul hat folgende syntaktische Anordung:

```
<<module>> DEFINITIONS ::= BEGIN

<<linkage>>

<<identity>>

<<declarations>>

END
```

Listing 3.1: SMI Modulsyntax [6], S.81

`<<module>>` ist der Name des Moduls, der sowohl informeller Natur als auch möglicherweise zur eindeutigen Bestimmung verwendet werden kann. Im `<<linkage>>`-Teil werden die Beziehungen zu anderen Modulen definiert. In diesem Teil wird bestimmt, welche Definitionsteile importiert werden. Der in der SMIv2 eingeführte `<<identity>>`-Abschnitt dient zur Beschreibung von Modulerstellern, Versionsverlauf, Kontaktinformationen und zur Beschreibung des MIB-Moduls. Dieser Abschnitt muss laut SMIv2 exakt einmal in jedem Modul existieren. Der `<<declarations>>`-Teil enthält die Definitionen des Moduls.

```
   Printer-MIB DEFINITIONS ::= BEGIN
 2
   IMPORTS
 4     MODULE-IDENTITY, OBJECT-TYPE, mib-2, Counter32, Integer32,
           TimeTicks, NOTIFICATION-TYPE, OBJECT-IDENTITY FROM SNMPv2-SMI
 6     TEXTUAL-CONVENTION FROM SNMPv2-TC
       MODULE-COMPLIANCE, OBJECT-GROUP FROM SNMPv2-CONF
 8     hrDeviceIndex, hrStorageIndex FROM HOST-RESOURCES-MIB;

10 printmib MODULE-IDENTITY
       LAST-UPDATED "9411250000Z"
12     ORGANIZATION "IETF Printer MIB Working Group"
       CONTACT-INFO
14             "       Steven Waldbusser
            Postal: Carnegie Mellon University
16                    4910 Forbes Ave
                  Pittsburgh, PA, 15213
18
                 Tel: 412-268-6628
20               Fax: 412-268-4987
              E-mail: waldbusser@cmu.edu"
22     DESCRIPTION
               "The MIB module for management of printers."
24     ::= { mib-2 43 }
```

Listing 3.2: Ausschnitt der Printer-MIB[7]

Als Beispiel ist in Listing 3.2 der Kopfteil der `Printer-MIB` zu sehen, anhand derer man die verschiedenen Teile sehen kann. Die MIB beginnt in Zeile 1 mit der Moduldefinition. Darauf folgt der Linkage-Teil von Zeile 3 bis 8, in dem verschiedene Definitionen aus anderen Modulen importiert werden. In den Zeilen 10 bis 24 findet dann die Deklaration des Identity-Moduls statt.

Einen eingeschränkten, zum Teil neu definierten Teil von ASN.1 bildet nach RFC 2578 [15] und RFC 1155 [13] die Beschreibungssprache der SMI. Mit dieser Notation können folgende Informationen eines Objektes beschrieben werden:

- Name und Objektidentifikator des Knotens
- Syntax der in dem Knoten enthaltenen Managementinformation in Form eines der Spezifikation entsprechenden ASN.1-Datentyps
- Informelle Beschreibung der Semantik dieser Managementinformation
- Statusinformation, die festlegt, ob die Managementinformation obligatorisch (mandatory), optional oder ungültig (obsolete) ist, weil eine neue Festlegung der MIB inzwischen stattgefunden hat.

Dabei sind folgende Datentypen von ASN.1 in der SMI erlaubt:

- INTEGER: Ganze Zahl, unbegrenzt; die möglichen Werte können durch eine Aufzählung vorgegeben und mit symbolischen Namen benannt werden.
- OCTET STRING: Null oder mehr Oktette als Wert. Jedes Byte eines Oktetts kann einen Wert von 0 bis 255 annehmen.
- OBJECT IDENTIFIER: Referenz auf eine eindeutige Kennzeichnung (s.o. Objektidentifikatorbaum)
- SEQUENCE, SEQUENCE OF: Zusammengesetzter Datentyp aus anderen ASN.1 Datentypen, wie eine Struktur in anderen Sprachen.

Zusätzlich zu den erlaubten Datentypen aus ASN.1 definiert die SMIv1 (SMIv2) folgende Datentypen:

- NetworkAddress: Protokollfamilie
- IpAddress: 32bit Internet-Adresse.
- Time Ticks: Zeitraum in 1/100 Sekunden .
- Gauge32: Zählobjekt, Wert zwischen 0 und $2^{32}-1$, inkrementieren, dekrementieren, kein Umlauf.
- Counter32 (Counter64): Zählobjekt, Wert zwischen 0 und $2^{32} - 1$ ($2^{64} - 1$), nur inkrementieren, Umlauf.
- Opaque: Enthält den Wert eines beliebigen ASN.1-Datentyps.
- (Unsigned32): Eine ganze 32bit Zahl ohne Vorzeichen, Wert zwischen 0 und $2^{32} - 1$.

Neben diesen Datentypen haben sich im Laufe der Zeit Datentypennotationen gebildet, die einheitlich in den meisten Dokumenten verwendet werden. Diese Datentypen, wie zum Beispiel `DisplayString`, sind in RFC 2579 [16] zusammengefasst.

Einen wichtigen Teil im Informationsmodell und auch in der SMI stellt die eindeutige Identifizierung der Managementobjekte dar. Um ein Objekt herstellerübergreifend zu identifizieren, existiert in der OSI-Welt ein Objektidentifizierungsbaum (vgl. Abb. 3.2 S. 16), in dem die Objekte hierarchisch organisiert sind. Jede Kante in diesem Baum hat einen Namen und eine Nummer. Über eine dieser beiden Möglichkeiten kann man jeden Pfad in der Baumstruktur festlegen und so eine Referenz auf das gewünschte Objekt erzeugen. Die inneren Knoten dienen lediglich zur Strukturierung und enthalten keine darüber hinausgehenden Zusatzinformationen.

Möchte man zum Beispiel ein Objekt in der IP-Objektgruppe referenzieren, so kann man zum einen ein eine reine Zahlenliste, die dem Pfad entspricht, angeben { 1 3 6 1 2 1 4 } oder aber eine Liste der Kantennamen { iso identified-organization(3) dod(6) ...}. Mischformen sind auch zulässig. Was in der Referenz noch fehlt, ist die Referenz auf das Objekt in der IP-Objektgruppe selbst. Möchte man das zweite Objekt in der IP-Gruppe referenzieren, so ist die OID, wie man diese Referenzen nennt, { 1 3 6 1 2 1 4 2 }.

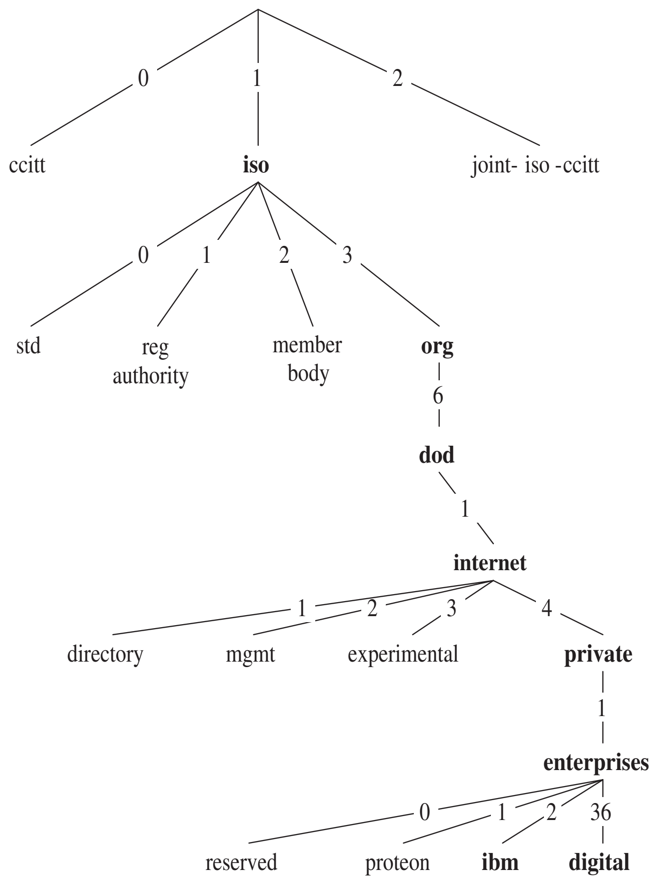

Abbildung 3.2.: Ausschnitt aus dem Objektidentifizierungsbaum von ASN.1

Um herstellerspezifische Gruppen und Objekte zu ermöglichen, ist unter dem "internet"-Knoten (OID { 1 3 6 1 }) in der "private"-Gruppe die Gruppe "enterprises" angelegt worden. In diesem Teilbaum können die Hersteller eine Knoten-ID beantragen und unterhalb dieser einen eigenen Teilbaum spezifizieren. In Abbildung 3.2 ist dies graphisch dargestellt.

Eine der entscheidenden Eigenschaften der OIDs ist ihre Gültigkeit. Sobald die OID für ein Objekt im Baum registriert worden ist, kann diese OID oder die Eigenschaften des mit ihr beschriebenen Objektes nicht mehr zurückgenommen oder für eine andere Aufgabe verwendet werden.

Um nun den Wert eines Objektes zu erfahren, muss die Objektinstanz auf der Ressource direkt abgefragt werden. Dazu benutzt man den oben beschriebenen Object Identifier und hängt die Instanzreferenznummer an diese OID an. Bei einfachen Objekten ist das die Null, da nur eine Instanz existiert. Im obigen Beispiel wäre die Referenz auf die Objektinstanz 1.3.6.1.2.1.4.2.0 für das Objekt 1.3.6.1.2.1.4.2 (IP-MIB::ipDefaultTTL). Diese Objektinstanznummern haben eine große Bedeutung bei Tabellen. Da Tabellen so modelliert werden, dass man `SEQUENCE`- und `SEQUENCE OF`-Konstrukte ineinander verschachtelt. Diese Modellierung bewirkt eine spaltenorientierte Referenzierung der Daten. Eine mögliche Indizierung zur eindeutigen Bestimmung der Objekte in den Spalten liefern so genannte Indizierungsspalten in den Tabellen. Sie müssen in der Tabellendefinition angegeben und in der angegebenen Reihenfolge indiziert werden.

Der Zeilenindex richtet sich dabei immer nach den Datentypen der Indizierungsspalten. So richtet sich bei der IP-Adressierungstabelle der Index nach den Einträge der `ipAdEntAddr`-Spalte. Die Indizierung der Spalte für die standard Loopback-Schnittstelle wäre zum Beispiel `127.0.0.1`. Damit ergibt sich für die Objektinstanz der Spalte `ipAdEntNetMask` der Loopback-Schnittstelle folgende OID:
“`mib-2.4.ipAddrTable.1.ipAdEntNetMask.`*`127.0.0.1`*”.

Als Beispiel für eine einfache Tabelle dient die Interfaces-Gruppe aus der MIB-II (siehe Listing 3.3).

```
ifTable OBJECT—TYPE
    SYNTAX SEQUENCE OF IfEntry
    ACCESS not—accessible
    STATUS mandatory
    DESCRIPTION
           "A list of interface entries. The number of
           entries is given by the value of ifNumber."
    ::= { interfaces 2 }



 ifEntry OBJECT—TYPE
    SYNTAX IfEntry
    ACCESS not—accessible
    STATUS mandatory
    DESCRIPTION
           "An interface entry containing objects at the
           subnetwork layer and below for a particular
           interface."
    INDEX  { ifIndex }
    ::= { ifTable 1 }



 IfEntry ::=
    SEQUENCE {
        ifIndex
            INTEGER,
        ifDescr
            DisplayString,
```

```
            ifType
32              INTEGER,
            ifMtu
34              INTEGER,

36          [...]

38          ifSpecific
                OBJECT IDENTIFIER
40      }

42


44  ifIndex OBJECT-TYPE
        SYNTAX INTEGER
46      ACCESS read-only
        STATUS mandatory
48      DESCRIPTION
                "A unique value for each interface. Its value
50              ranges between 1 and the value of ifNumber. The
                value for each interface must remain constant at
52              least from one re-initialization of the entity's
                network management system to the next re-
54              initialization."
        ::= { ifEntry 1 }
```

Listing 3.3: Definition der Schnittstellen-Tabelle der MIB-II [8]

Da die Organisation der Elemente spaltenweise erfolgt, sind die Werte auch spaltenorientiert auszulesen. Die Objektinstanz des Typs der dritten Schnittstelle ist somit: `{mib-2.2.ifTable.ifEntry.ifType.3}`. Dies erklärt sich damit, dass als Indizierungsspalte in dieser Tabelle die Spalte *ifIndex*, Listing Zeile 20, mit den Indizes 1, 2, 3, 4,... angegeben ist. Somit ergeben sich für die Objektinstanzen auch die Indizierungen 1, 2, 3,.. . Ein Zugriff in einer Tabelle ist somit folgendermaßen strukturiert: `TabellenName.Strukturname.SpaltenIndex.ZeilenIndex`. Der Teil `TabellenName.Strukturname` ergibt sich aus dem Zugriff auf die Tabelle, `ifTable`, Listing Zeile 1-8, und der zugehörigen Struktur, `ifEntry`, Listing Zeile 12-21, um auf die (Spalten-)Elemente der Tabelle zugreifen zu können. Eine Besonderheit beider Objekte ist der Status *not-accessible* (Listing Zeile 3 und 14), der darauf hinweist, dass auf beide Elemente nicht direkt zugegriffen werden kann.

Vorhandene Tabellen können mit Hilfe des `Augments` Konstrukts erweitert werden (vgl. [10] S. 347). Damit wird eine Flexibilisierung der starrqen Tabellenstruktur erreicht, um zum Beispiel geräteabhängige Eigenschaften zu modellieren.

3.2.2.2. MIBs

Die von SNMP überwachten Objekte werden in MIBs definiert. MIBs sind somit *"Spezifikationen, die Definitionen von Managementinformationen enthalten, anhand derer Netzwerksysteme entfernt überwacht, konfiguriert und kontrolliert werden können"* ([3], S.xi). Die standardisierten Objekte der MIB-Spezifikation werden in MIBs/MIB-Module eingeteilt. In diesen Standard-Gruppen sind die Objekte der gebräuchlichsten gemanagten Gegenstände eines Netzwerkes, wie zum Beispiel Bridges, Routers, LANs, spezifiziert.

Eine MIB-Spezifikation ist standardmäßig aus drei Teilen zusammengesetzt.

Wörtliche Beschreibung In diesem Teil der Spezifikation findet die umgangssprachliche Beschreibung der MIB statt. Dieser Teil ist wichtig für die Entwicklung von Agenten und Managern, um eine eindeutige Interpretation der in der MIB definierten Objekte zu gewährleisten.

Modulbeschreibungen In diesem Teil findet die SMI Modulbeschreibung statt.

Referenz auf andere Dokumente Der letzte Teil enthält Referenzen auf andere Dokumente.

Nach diesem Schema ist auch die MIB-II beschrieben. Die Definition der MIB-II in RFC 1213 [8] umfasst die Datentypen der Objekte in den Objektgruppen. Wenn ein Gerät Objekte einer Objektgruppe bereitstellen will, so müssen alle in RFC 1213 definierten Objekte einer Gruppe zum Abruf vorgehalten werden. Teilimplementierungen sind nicht vorgesehen. Eine Möglichkeit mit der Implementierung des Agenten von der beschriebenen MIB abzuweichen, bilden die in der SNMPv2 als "Information Modules" eingeführten `AGENT-CAPABILITIES` und `MODULE-COMPLIANCE` Makros (vgl [10] S. 405). Mit diesen Makros können von den MIBs abweichende Implementierungen beschrieben und dokumentiert werden.

Die Definitionen von RFC 1213 befinden sich inzwischen in mehreren aktualisierten Modulen in neueren RFCs[3]. Daher ist es eigentlich nicht mehr sinnvoll von der MIB-II zu sprechen, aber es hat sich nicht nur in diesem Bereich ein mehrdeutiger Gebrauch des Wortes MIB durchgesetzt. So wird von einer MIB als Definition von MIB-Objekten in einer SMI-Datei oder der Ansammlung von Objekt-Instanzen, die in einem Agenten existieren, gesprochen.

3.2.3. Das Kommunikationsmodell

Das SNMP-Protokoll ist ein so genanntes symmetrisches asynchrones Request-Response-Protokoll. Das bedeutet, dass SNMP-Nachrichten ohne vorheriges Handshake gesendet werden. Normalerweise wird SNMP so benutzt, dass die Management Station eine Anfrage an einen Agenten sendet, um Informationen abzurufen oder Objektwerte zu ändern. Der Agent sollte dann mit der angeforderten Information oder einer Bestätigung der Änderung antworten. Die Requests und die zugehörigen Responses werden nicht synchronisiert, sondern die Zuordnung wird durch einen "Request Identifier" (Auftragskennung) gewährleistet. Es ist dabei nicht sichergestellt, dass keine Nachrichten verloren gehen. Einen Sonderstatus nehmen die so genannten

[3] Aktualisierte Definitionen in RFC 2011, 2012, 2013

TRAP-Meldungen ein; sie dienen zur Ereignismeldung von Agenten. Diese Meldungen werden unbestätigt an die Manager versandt.

Um die angesprochenen Operationen auszuführen, sind für das SNMPv2-Protokoll, welches im RFC 3416 [17] beschrieben wird, acht Nachrichtentypen definiert. Diese acht Nachrichtentypen bilden sechs Kommunikationsprozesse.

GET-Request Dieser Request dient der Anforderung eines oder mehrerer anhand der OID bestimmter Objektes. Falls ein Objekt nicht vom Agenten bereitgestellt wird oder ein Referenzierungsfehler vorliegt, so wird ein `noSuchObject`- oder `noSuchInstance`-Fehler mit einer Kopie der angeforderten OIDs zurückgegeben.

GETNEXT-Request Um eine Möglichkeit zu haben, die Objekte eines Agenten anzufordern, ohne das im Agenten implementierte Objektmodell (MIB View) zu kennen, wurde dieser Request eingeführt. Es wird in einem Request nicht die genau angegebene Objektinstanz zurückgegeben, sondern die, die in der lexikographischen Ordnung nächste Instanz. Damit ist es auch möglich, die Indexwerte von Tabellen zur weiteren Referenzierung zu bestimmen.

GETBULK-Request Dieser Request fordert die Werte zu einer Folge von Objekten an. Damit kann mit einem Befehl z.B. eine ganze Tabelle gelesen werden. Begrenzt wird die Anzahl der zurückgegebenen Objekte nur durch die maximale Framelänge des Transportmediums.

SET-Request Dieser Request wird benutzt, um eine Wertänderung einer Objektinstanz auf der verwalteten Station anzufordern.

TRAP Traps sind unbestätigte Ereignismeldungen vom Agenten zu interessierten Management-Stationen. Es gibt sechs definierte Ereignisse allerdings können über diese Ereignisse hinaus auch gerätetypische, nicht standardisierte, Trap-Meldungen erzeugt werden. Falls ein nicht standardisiertes Ereignis gemeldet wird, kann man die Art des Ereignisses durch die Felder *enterprise* und *specific-trap* bestimmen.

INFORM Informs sind im Grunde genommen bestätigte TRAPS und zum Austausch von Informationen zwischen zwei Management-Stationen gedacht.

[18]

Die Protokollbindung von SNMP ist üblicherweise UDP. SNMP ist aber nicht an UDP gebunden, sondern wurde so entworfen, dass es unabhängig vom unterlagerten Transportprotokoll arbeitet. Die Adressierung erfolgt bei Bindung an UDP über die Ports 161 zum Senden an den Agenten und 162 zum Empfangen von Traps. Weitere Protokollbindungen sind in RFC 3417 [19] beschrieben [6, 11, 3, 20].

Um eine Kommunikation von verschiedenen Rechnerarchitekturen zu gewährleisten, sind die BER gewählt worden, um eine einheitliche Bitkodierung zu erreichen.

3.2.4. Das Funktionsmodell

Durch das Managementkonzept von SNMP als integrierter Ansatz und das zugrundeliegende Informationsmodell (vgl. Kap. 3.2.2, S. 12) ist eine Einordnung in die beschriebenen Funktionsteilbereiche der OSI (vgl. Kap. 2.1, S. 5) nur schwer möglich. Durch die Flexibilität und die plattform- und architekturübergreifende Spezifikation des SNMP-Modells ist eine Abdeckung aller Teilbereiche der "OSI Management Functional Areas" möglich. Das Funktionsmodell einer SNMP-Netzwerkimplementierung wird durch die in den Agenten implementierten MIBs gebildet.

3.2.5. Sicherheitsmechanismen in SNMP, insbesondere SNMPv3

SNMP bietet in der ersten Version des Protokolls nur rudimentäre Sicherheitsmechanismen. So wird als Zugriffsberechtigung den `GET` und `SET`-Requests ein so genannter *"Community"*-String beigefügt. Ist dieser mit einem im Agenten hinterlegten *"Community"*-String gleich, so wird die angeforderte Operation bezüglich dieser *"Community"* erlaubt. Durch die Definition von verschiedenen "Communities" ist es möglich, verschiedene Zugriffsrechte zu realisieren. Weitverbreitete Gruppen sind zum Beispiel die "Communities" `private` für den vollen Zugriff und `public` für den lesenden Zugriff auf die Netzwerkressource. Dadurch, dass zum einen in den meisten Implementierungen und Konfigurationen von Agenten die Standard-"communities" nicht variiert werden und zum anderen das Problem besteht, dass die Gruppennamen in den SNMP-Nachrichten im Klartext transportiert werden und daher sehr verwundbar gegenüber dem Erlauschen sind, bietet diese Funktion keine große Sicherheit gegenüber Angriffen.

Als zusätzliche Sicherheitsfunktion bieten Agenten zum Teil ACLs [4] an, in denen die IP-Adressen der zugriffsberechtigten Rechner eingetragen sind.

Da diese Maßnahmen insgesamt keinen ausreichenden Schutz gewährleisten, sind in den Standards zu SNMPv2 und SNMPv3 erweiterte Sicherheitsmechanismen aufgenommen worden.

Die Maßnahmen, die in SNMPv2 eingeführt worden sind, haben sich aber im Gebrauch durch zu hohe Komplexität als nicht praktikabel erwiesen. Daher entstand eine "Arbeitsversion", SNMPv2c (classic), ohne diese Funktionen.

Es ist in SNMPv3 wieder versucht worden, die in SNMPv2 enthaltenen Sicherheitsmechanismen in abgewandelter Form in das Protokoll zu integrieren und gleichzeitig ein Rahmenwerk zu schaffen, auch vorhandene SNMPv1-Implementierungen weiter zu nutzen. Während SNMPv2 inkompatibel zu SNMPv1 war, ist SNMPv3 keine alleinstehende neue Version, die die vorherigen Protokollversionen ersetzt, sondern eine die beiden vorherigen Protokollversionen integrierende Lösung.

Dies wird dadurch erreicht, dass zwischen die Dienstschnittstellen von SNMPv1/v2 und dem Transportprotokoll der SNMPv3-Dienst platziert wird. Somit wird die SNMP-PDU nicht an das Transportprotokoll weitergeleitet, sondern zuerst an den SNMPv3-Dienst. Dieser fügt an die SNMP-PDU einen SMNPv3-Header[5] an und leitet diese Nachricht dann an das Transportprotokoll weiter (vgl. Abb.3.3). Somit ist eine Funktionserweiterung ohne Veränderung der

[4] "Access Control List"

[5] Dieser besteht aus dem "Message Processing Model" und dem "User Security Model" (USM)

vorhandenen SNMP-Dienst-Version möglich. Das impliziert aber auch, dass entweder SMNPv1 oder SNMPv2 als Nachrichtenprotokoll zur Kommunikation mit dem Agenten zusätzlich vorhanden sein muss, da in der Spezifikation zu SNMPv3 keine neue SNMP-PDU definiert wird.

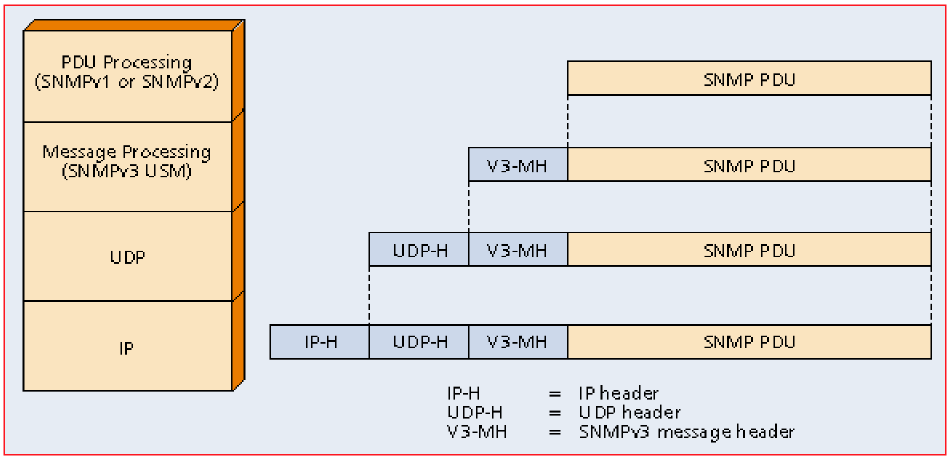

Abbildung 3.3.: PDU-Zusammensetzung SNMPv3 aus [2]

Die Architektur von SNMPv3 besteht aus Subsystemen, oder auch Modulen, die abstrakt durch ihre Serviceschnittstellen definiert sind. Sowohl die SNMP-Verarbeitungseinheit als auch die Anwendungen, die von ihr bedient werden, sind als eine Zusammensetzung von einzelnen Subsystemen definiert. Durch die von einer Maschine implementierten Subsysteme wird ihre jeweilige Rolle bestimmt.

Das Sicherheits-Subsystem von SNMPv3 soll die Authentizität, Vertraulichkeit und Aktualität von Nachrichten sicherstellen. Die Sicherheit von Nachrichten läßt sich wahlweise durch die Zuschaltung von Verschlüsselungsmethoden erhöhen. Es lassen sich weiterhin bestimmte Informationen über das Access-Controll-Modell für Benutzer sperren. Es ist nicht dazu gedacht DoS-Attacken[6] zu erkennen und abzuwehren oder die Verkehrsanalyse zwischen den Konten zu unterbinden.

Die Informationen über die verwendeten Sicherheitsverfahren sind im USM-Feld innerhalb des SNMPv3-Header untergebracht.

Um diese Sicherheitsfunktionen zur Verfügung zu stellen, sind zwei kryptographische Verfahren für das Sicherheitssubsystem USM definiert worden: die Authentisierung und Verschlüsselung. Um diese beiden Verfahren zu ermöglichen, muss jede SNMP-Verarbeitungseinheit zwei Schlüssel halten: einen privaten Schlüssel (privKey) und einen Schlüssel zur Authentisierung (authKey). Weitere Informationen zur Behandlung der Schlüssel sind in RFC 2274 [21] enthalten.

Es werden weiterhin verschiedene Schlüsselsätze für jeden lokalen und entfernten Benutzer, der einen Zugang zu den Management-Informationen haben will, verwaltet. Diese Schlüssel

[6] DoS-Attacke: Denial-of-Service-Attacke; dabei wird der angegriffene Knoten mit Anfragen geflutet, so dass “normale” Anfragen in der Flut der Anfragen untergehen und durch überlaufende Empfangspuffer verworfen werden oder durch die Überlastung der Verarbeitungseinheiten nur stark verzögerte Antworten möglich sind.

können per SNMP-Befehl nicht abgefragt werden.

Die benutzten Authentisierungsalgorithmen sind HMAC-MD5-96 und HMAC-SHA-96, DES werden im CBC-Mode zur Verschlüsselung benutzt.

Um die Zugriffsrechte auf den Knotenpunkten zu regeln, wird in SNMPv3 das so genannte View Based Access Control (VACM) benutzt.

In RFC 2575 [22] sind fünf Elemente erwähnt, die das VACM ausmachen: Gruppen, Sicherheitsebene, Kontext, MIB Views und Zugangskontrolle.

Jeder, der Zugang zu Informationen auf einem Knoten haben möchte, muss in einer Gruppe eingetragen sein. Über diese Gruppe wird dann die Art der Zugangsberechtigung bestimmt. Die Rechte einer Gruppe hängen wiederum von der Sicherheitsebene[7] der Nachricht ab, die die Anfrage enthält. Der Kontext und die MIB Views dienen dazu, die Zugriffsrechte auf bestimmte Teilmengen der Objekte einer lokalen MIB zu beschränken und zu regeln. Die Zugangskontrolle selber bedient sich der vorhergehenden Punkte, um eine Entscheidung zu treffen, welche Rechte für eine Anfrage in Kraft treten sollen [2].

[7]Sicherheitsebenen sind zum Beispiel unverschlüsselt oder verschlüsselt

4. Realisierung des Managementszenarios mittels SNMP

In diesem Kapitel soll die schrittweise Realisierung einer Managementaufgabe mittels SNMP skizziert werden. Dazu wird zuerst der Entwicklungsweg auf Agentenseite beschrieben, um im Anschluss die Möglichkeiten auf der Managerseite zu betrachten.

4.1. Modellierung der MIB

Um ein System um die SNMP-Managementfähigkeit zu erweitern, sind verschiedene Phasen eines Entwicklungsprozesses zu durchlaufen. Die einzelnen und im Folgenden besprochenen Phasen zu solch einem Ziel sind in Abbildung 4.1 dargestellt.

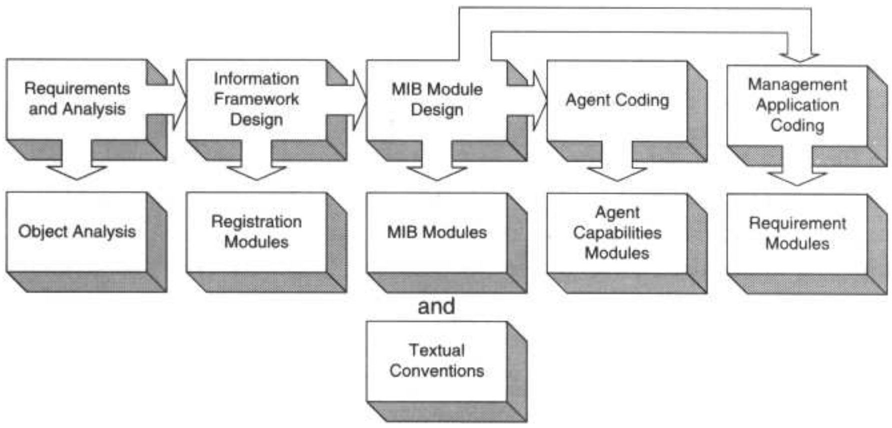

Abbildung 4.1.: MIB Entwicklungsschritte ([3], S. 215)

Die einzelnen Phasen sind von links nach rechts in der oberen Zeile aufgetragen, die resultierenden Dokumente der Phasen jeweils darunter.

4.1.1. Die Objekt/Systemanalyse

Die erste Phase umfasst das Erfassen und Kategorisieren der zu managenden Systemeigenschaften. *“Um ein effektives MIB-Design zu erreichen, muss das System systematisch in spezifische Kategorien unterteilt werden und in einem Registrierungsbaum organisiert werden”* ([3], S.188).

Um eine Systematik im Design zu erreichen, wird in [3] das System in folgende Kategorien unterteilt, um daraus ein MIB-Design zu entwickeln:

Komponenten Diese Kategorie dient dazu eine Hilfe zu geben, um die Schnittstellen und Dienste zu beschreiben, auf die der SNMP-Agent Zugriff haben soll.

Attribute Sind die Eigenschaften eines modellierten Objektes, wie zum Beispiel die Mail-Adresse des Geräteadministrators oder der Standort des Gerätes.

Aktionen Dienen zur Kontrolle des Systems, wie zum Beispiel das Rebooten eines Gerätes oder das Starten und Stoppen von Diensten.

Statistiken Halten nützliche Informationen über das System bereit, die seit einem bestimmten Zeitpunkt angefallen sind, zum Beispiel die Anzahl der versendeten Pakete einer Schnittstelle.

Status Enthält Objekte, die den aktuellen Zustand des Systems anzeigen, wie zum Beispiel den Zustand der Lüfter (an/aus).

4.1.1.1. Komponenten

Die Einteilung des zu managenden Systems oder Gerätes in seine physischen oder logischen Komponenten ist vielleicht der am schwersten zu modellierende Aspekt in der Modellierung einer MIB. Dadurch, dass diese Komponenteneinteilung bei der Auswahl der noch zu erzeugenden anderen Kategorien maßgeblich verwendet werden kann, ist er einer der wichtigsten, weswegen dieser Schritt auch zuerst ausgeführt wird. Man unterteilt Komponenten in physische und logische Komponenten, je nachdem, ob sie direkt mit dem Gerät zu tun haben, wie zum Beispiel die Liste der Steckkarten in einem Gerät, oder ob sie im Gerät implizit enthalten sind, wie zum Beispiel eine Liste der Dienste, die ein Netzwerkserver bereitstellt. Dabei ist eine Trennung der beiden Typen nicht immer einfach vorzunehmen.

Beim Vorgehen der Einteilung bietet sich ein Top-Down Verfahren an, bei dem man zuerst fragt, welche Komponenten ein Gerät enthält. Bei den ermittelten Komponenten stellt sich diese Frage erneut. Dabei ist zu beachten, nicht zu sehr in die Tiefe zu gehen, sondern das Hauptaugenmerk auf den Hauptsystemen und die kontrollierbaren Aspekten zu legen.

Nach dieser Einteilung ist es nötig, zu bestimmen, wie viele Komponenten eines Typs es gibt, um die Anzahl der benötigten Instanzen zu bestimmen, was später bei der Modellierung der MIB wichtig wird. Falls mehrere Instanzen eines Typs exitieren, wird diese Komponente später in Form einer Tabelle modelliert, im Falle einer einzelnen Instanz als Skalar.

4.1.1.2. Attribute

Attribute beschreiben die statischen Eigenschaften des Systems. Damit sind die nicht veränderliche Eigenschaften gemeint.

Um die Wartbarkeit zu verbessern, werden die Attribute analog zu den Komponenten in Gruppen eingeteilt, wozu die Komponenteneinteilung benutzt werden kann.

Attribute können auch dazu verwendet werden, um Objekte bei mehrfachen Objektinstanzen zur Bestimmung zu benutzen. Dabei kann die Bestimmung einer Objektinstanz über die (natürlichen) Attribute eines Objektes gehen, aber auch über die Erzeugung eines zusätzlichen Attributs für das Objekt, durch das eine eindeutige Unterscheidung möglich wird.

4.1.1.3. Aktionen

Da SNMP kein Konzept für die Ausführung von expliziten Aktionen hat, werden diese durch implizite Aktionen, die durch Seiteneffekte ausgelöst werden, modelliert. Dabei wird ein MIB-Objekt auf einen so genannten “trigger value” gesetzt, der dann die gewünschte Aktion ausführt.

Bei der Modellierung einer Aktion ist es wichtig zwischen Aktionszeilen und den eigentlichen Aktionsschritten zu unterscheiden. Beim Design der Aktionsschritte ist es wiederum wichtig, bei der Einteilung der Schritte die richtige Granularität zu wählen. Wenn die Einteilung zu fein ist, kann das zu einer Schwerfälligkeit für Managementanwendungen und Benutzer führen, bei einer zu groben Einteilung zur Unflexibilität.

Perkins [3] stellt auf Seite 195 folgende Faustregeln zur Erzeugung von Aktionen auf:

1. *Erzeuge keine Aktionen, die komplexe zusammenhängende Aktionen sind. Das heißt, vermische nicht Aktionsziele mit Aktionsschritten.*

2. *Mache keinen Unterzustand sichtbar, ohne eine Fehlermeldung für diesen Zustand sichtbar zu machen.*

3. *Zerteile eine Aktion nicht in zu kleine Stücke.*

Albert Einstein sagte einst: “Mache die Dinge so einfach wie möglich, aber nicht zu einfach”.

Des Weiteren müssen Aktionen durch die Verwendung eines verbindungslosen Datagrammdienstes als Kommunikationsmedium so designt sein, dass Mehrfachaufrufe durch verloren gegangene Pakete kompensiert oder beachtet werden können.

4.1.1.4. Statistiken

Statistiken sind Werte von Zählern, die seit einem bestimmten Zeitpunkt angefallen sind, eine Aufzeichnung interessanter Ereignisse eines Systems (zum Beispiel “high peaks”). Dabei ist es wichtig, dass eine Abgrenzung zu den Statusdaten und Attributen vorgenommen wird. Statistikwerte können immer nur steigen, aber nicht fallen, das heißt, es gibt keine fluktuierenden Werte. Ein Beispiel für einen Statistikwert ist die Anzahl der gesendeten Pakete einer Schnittstelle, aber nicht deren momentaner Durchsatz.

Die Bestimmung der Statistikobjekte findet wieder um das schon vorhandene Komponentenmodell statt. Dabei helfen folgende Fragen bei der Bestimmung der Objekte: Welche Ereignisse gibt es um eine Komponente? Wird eine Aktion regelmäßig ausgeführt, die interessant für Leistungs- oder Fehlermanagement ist ?

Auch die Lebenszeit der Werte ist zu beachten. Die Werte können persistent oder flüchtig vorhanden sein. Diese Angabe ist wichtig für die Auswertung durch ein Managementsystem.

4.1.1.5. Status

Die Statusobjekte repräsentieren die aktuelle Situation, in der sich das System gerade befindet. Dabei kann man ebenfalls zwei Typen von Objekten unterscheiden: Zum einen, ob das Objekt eine Art Sensorwert, wie zum Beispiel die Temperatur des Systems, darstellt, oder den Zustand, in dem sich eine Komponente befindet, wie zum Beispiel der Zustand einer Netzwerkschnittstelle (up, down, testing, unkown, dorment). Letztere lassen sich durch Zustandsdiagramme ähnlich Moore- oder Mealy-Automaten ermitteln.

4.1.2. Information Framework Design

Im “Information Framework Design” geht es darum, die gesammelten Informationen aus der Analysephase dazu zu benutzen, das Informationsrahmenwerk und die MIB-Struktur zu bestimmen.

Perkins empfiehlt in [3] nicht alles in einem MIB-Modul abzulegen, sondern ein Rahmenwerk aus Informationsmodulen anzulegen, um eine höhere Wartbarkeit und Erweiterbarkeit zu erreichen. Diese Informationsmodule bestehen aus OID-Registrierungen, “Textual Conventions”, Implementierungsanforderungen, Implementierungsprofilen, SNMPv1-Traps und SNMPv2-Notifications. Diese Module zusammen ergeben dann das Informationsrahmenwerk. Ein Augenmerk sollte auf die Organisation der OID-Infrastruktur gelegt werden. So führt ein ungünstiges Design in der Anfangsphase dazu, dass sich spätere Ergänzungen nur noch schwer in die vorhandene Struktur einfügen lassen. Der Weg sollte also über eine tief strukturierte OID-Struktur gehen.

4.1.3. MIB-Modul Design

Durch die Vorarbeit beim Design des Rahmenwerkes und der Analyse des Systems gestaltet sich die Entwicklung der MIB-Module relativ einfach. Zu achten ist weiterhin auf eine gute Strukturierung der OIDs im Sinne des Informationsrahmenwerks.

Aus der Systemanalyse kann man anhand der gesammelten Daten die MIB-Objekte modellieren.

Dazu hat Perkins [3] auf Seite 200 folgenden Regeln aufgestellt:

- *Alle Komponenten mit einer Anzahl größer als eins sollten in einer Tabelle modelliert werden.*
- *Statistiken, die ansteigende Werte repräsentieren, sind als* `COUNTER` *zu modellieren.*
- *Statistiken, die den höchsten oder niedrigsten Stand repräsentieren, sollten als* `INTEGER` *modelliert werden.*
- *Status, die einen bestimmten Zustand, in dem sich die Komponente befindet, beschreiben, werden als nummerierter* `INTEGER` *modelliert, wobei jeder bezeichnete Wert einen Zustand repräsentiert.*
- *Status, die einen fluktuierenden Wert beschreiben, werden als* `GAUGE` *modelliert.*

- *Attribute können zwei Ausprägungen besitzen.* `OCTET STRINGS` *können zur Repräsentation lesbarer oder binärer Daten benutzt werden,* `INTEGER`*, um Zahlenwerte auszudrücken.*

Diese Regeln scheinen ziemlich einfach zu sein, die Hauptarbeit dabei liegt in der Erstellung einer MIB anhand der vorhergehenden Strukturierung/Analyse und nicht bei der Erstellung der Objektdefinitionen [3].

4.2. Agentenentwicklung

Bei der Entwicklung der Agenten auf ein Zielsystem kann man die in den vorhergehenden Phasen erzeugten MIBs benutzen, um mit Hilfe von MIB-Compilern Quellcode zu erzeugen, um den Agenten dann endgültig zu implementieren.

Dabei kann man den Prozess grundsätzlich in zwei Phasen aufteilen, die von einem Front-End-Compiler und Back-End-Compiler ausgeführt werden (vgl Abb. 4.2).

Der Front-End-Compiler erhält als Eingabe MIBs und erzeugt aus diesen ein Zwischenformat, welches im Normalfall von den Back-End-Compilern zum gewünschten Zielformat verarbeitet wird. Dabei können die Front-End-Compiler noch zusätzlich die Aufgabe der Syntax-Checks der Eingabe übernehmen. Gerade dieser Teil ist wichtig, wenn man MIBs erzeugen möchte, die einfach in Managementanwendungen integrierbar sind.

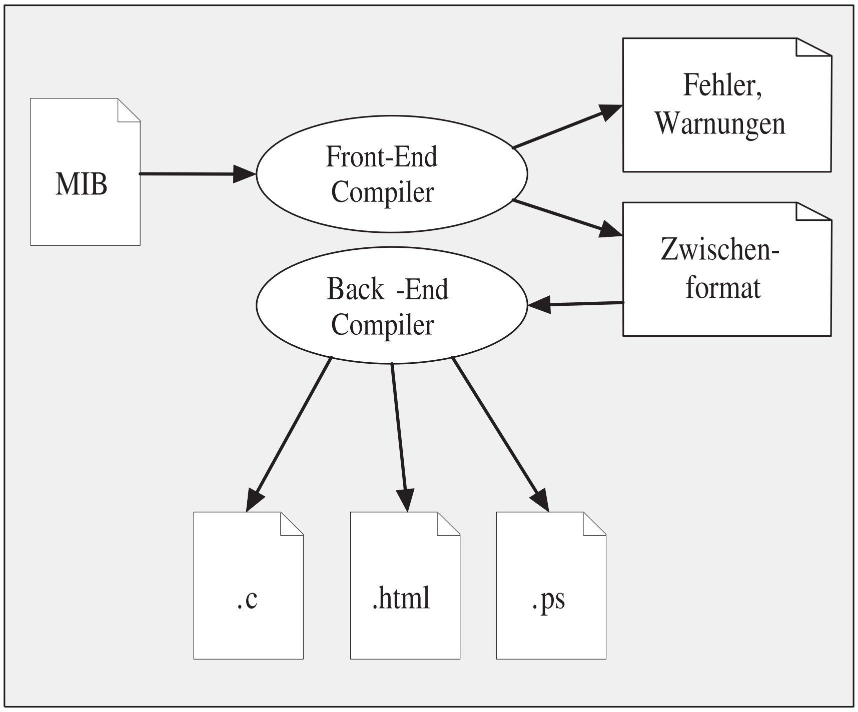

Abbildung 4.2.: Zusammenspiel der Back-End- und Front-End-Phasen beim Kompilieren

Ein Beispiel eines solchen Compilers ist der *SMIC*[1], der eine syntaktische und semantische Überprüfung der MIBs durchführt und zahlreiche Ausgabeformate erzeugt. Ein weiteres Einsatzgebiet der Front-End-Compiler stellen Managementanwendungen dar, die einen solchen Compiler zur Erzeugung des Eingabeformats für die Anwendung benutzen.

Back-End-Compiler dienen wie schon angesprochen dazu, die Zwischenformate in die gewünschten Zielformate zu übersetzen. So kann aus einer MIB zum Beispiel eine HTML-Repräsentation erzeugt werden.

In vielen Fällen sind Front-End- und Back-End-Compiler auch zusammengefügt und bilden zum Beispiel bei Agenten-Entwicklungssystemen einen Compiler, der MIBs in C-Skeleton-Code[2] erzeugt, mit dem dann ein die MIBs unterstützender SNMP-Agent erstellt werden kann.

Ein solcher Prozess zur Entwicklung einer Managementlösung mit eigener MIB bzw. eigenem Agenten ist in Abbildung 4.3 dargestellt.

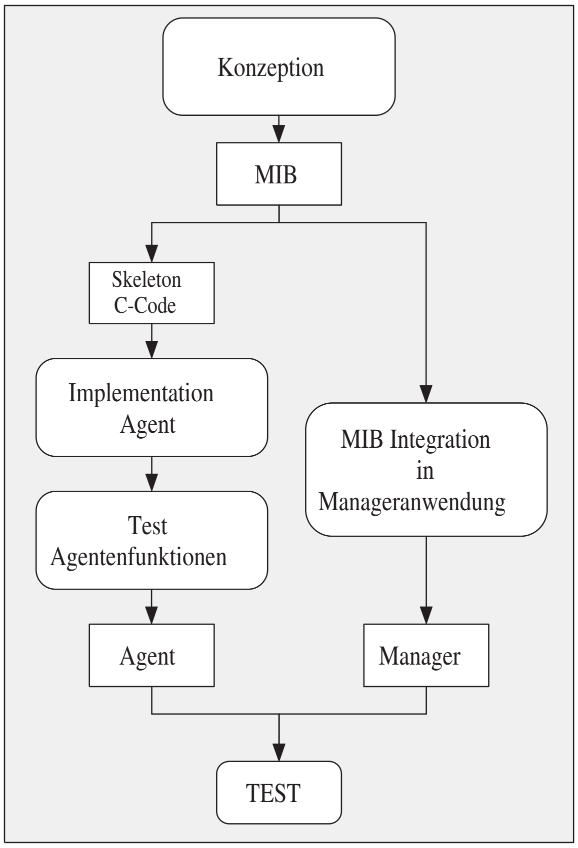

Abbildung 4.3.: Entwicklungsprozess einer SNMP-Netzwerkmanagementlösung

[1]SNMP Management Information Compiler

[2]Quellcode, in dem erzeugte und von der Entwicklungsumgebung unterstützte Funktionen schon vorhanden sind und in dem die aus einer Konfiguration (hier die MIBs) erzeugten Funktionsprototypen noch implementiert werden müssen.

4.3. Manager

Die meisten einfachen Manager, die SNMP-fähig sind, gehen kaum über die Möglichkeiten, die beispielsweise Kommandozeilenbefehle, wie zum Beispiel `snmpwalk, snmpget, snmpgetnext` aus dem *net-snmp*-Paket[3] [23] bieten, hinaus.

Die meisten freien Implementierungen bieten ein gegenüber der Kommandozeile verbessertes Verbindungsmanagement und einen so genannten MIB-Browser, mit dem man die auf den gemanagten Geräten unterstützten Objekte in einer Art Verzeichnisstruktur abfragen kann. Es findet hauptsächlich eine Informationsdarstellung, aber keine Verarbeitung statt. Dies deutet auf ein Hauptproblem in der SNMP-Managementwelt hin. Es existieren kaum einfach zu handhabende Netzwerkmanagementanwendungen ([24], S. 109).

Schönwälder unterscheidet die vorhandenen Managementanwendungen in [24] in fünf Kategorien die im Folgenden kurz erläutert werden:

Generic low-level SNMP Tools Mit diesen Werkzeugen sind die einfachen low-level SNMP Kommandozeilenbefehle ohne semantische Unterstützung gemeint. Bei ihnen ist auch ein grundlegendes Wissen um SNMP/MIB nötig, um die Ausgaben richtig interpretieren zu können.

Generic low-level SNMP APIs Diese APIs sind Werkzeuge und Bibliotheken, die Programmierern eine einfache Schnittstelle bereitstellen, um MIB-Objekte abzurufen un zu manipulieren.

Generic MIB-Browsers Mit einem MIB-Browser kann man durch die MIB-Objekte eines Agenten "browsen". Die meisten MIB-Browser sind aber allgemeine Werkzeuge, die keinerlei semantische Unterstützung bieten und sich auf die Darstellung der Werte beschränken. Somit ist wiederum ein Verständnis der benutzten MIBs vorausgesetzt.

Generic Monitoring Tools In diese Kategorie fallen Werkzeuge wie zum Beispiel MRTG [25], die Statistiken sammeln und von der Norm abweichendes Systemverhalten erkennen können. Allerdings haben einige dieser Werkzeuge noch Einschränkungen, die bei Nichtbeachtung von speziellen MIB-Semantiken zu fehlerhaftem Verhalten führen können.

Generic Management Platforms Diese Kategorie besteht aus Managementwerkzeugen, die eine allgemeine Management-Infrastruktur für die Implementierung einer Managementanwendung bereitstellen. Anwendungen, die auf diesen Werkzeugen aufbauen, nutzen plattformabhängige Schnittstellen und Dienste und enthalten oftmals allgemeine Werkzeuge zur Beobachtung, Ereigniskorrelation oder Topologieerkennung. Beispiele für solche Werkzeuge sind `tkined` [26] oder HP´s OpenView [27].

Im Folgenden werden zwei dieser Managementlösungen kurz betrachtet.

[3] ehemals CMU-SNMP der Carnegie Mellon Universität, stellt einen erweiterbaren Agenten und Kommandozeilenbefehle zur Abfrage bereit

4.3.1. scli

scli [28] ist ein Paket, um schnell und effizient kommandozeilenorientierte Netzwerkmanagementlösungen zu erstellen. Dabei wurde der Ansatz verfolgt, sich von den allgemeinen Ansätzen des Netzwerkmanagements zu lösen und ein Werkzeug bereitzustellen, um spezifische Managementlösungen zu erstellen. Mit spezifisch ist in diesem Zusammenhang gemeint, dass die Werkzeuge die zu manipulierenden Daten semantisch erfassen und die low-level-SNMP und MIB-Details verbergen.

Durch den Einsatz des *smidump*-Compilers, der aus MIBs C-Stub-Code erzeugt, ist es nicht nötig, komplexes SNMP-Wissen als Vorbedingung zur Erstellung einer solchen Lösung zu haben.

4.3.2. scotty

scotty [29] ist eine TCL-API, um Netzwerkmanagement-Software zu implementieren. Durch die Erweiterung mit *TNM*, einer Tcl-Erweiterung, um Netzwerkprotokolle, wie zum Beispiel SNMP, ICMP, DNS, HTTP, SUN RPC, NTP, UDP, zu Netzwerkmanagementinformationen zu nutzen, und *tkined*, einem Netzwerkeditor, ist das Basispaket das zur Zeit wohl komfortabelste und am weitesten entwickelte Open-Source Netzwerkmanagementwerkzeug mit einer graphischen Oberfläche (vgl. Abbildung 4.4).

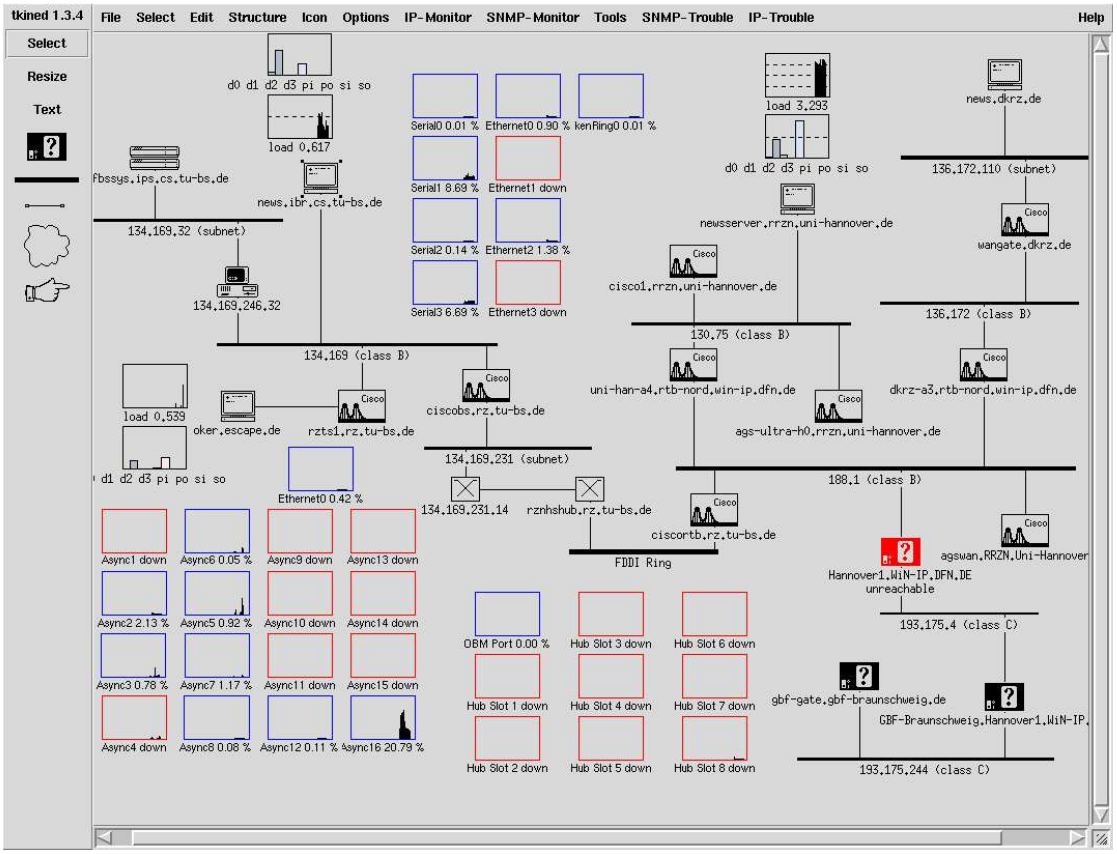

Abbildung 4.4.: tkined map

5. Webservices

5.1. Einleitung

"Ein Webservice ist eine über ein Netzwerk zugängliche Schnittstelle zu Anwendungsfunktionen, die mit Hilfe von Standardtechniken des Internets realisiert wird" ([4], S. 1). Das heißt, man spricht in dem Moment von Webservice, indem man einen entfernten Dienst mit Hilfe einer Kombination von Protokollen wie zum Beispiel HTTP, SMTP, XML oder FTP ansprechen kann.

Der Webservice stellt eine abstrakte Schnittstelle zwischen Anwendung und Benutzer dar. Durch diese Abstraktion wird eine Plattform- und Sprachunabhängigkeit realisiert. Durch die Abstraktion auf Protokollebene und die Verwendung der gängigen Internetprotokolle ist es möglich, dass jede Sprache, die Webservices unterstützt, auf die webserviceanbietende Anwendung zugreifen kann (vgl. Abb. 5.1).

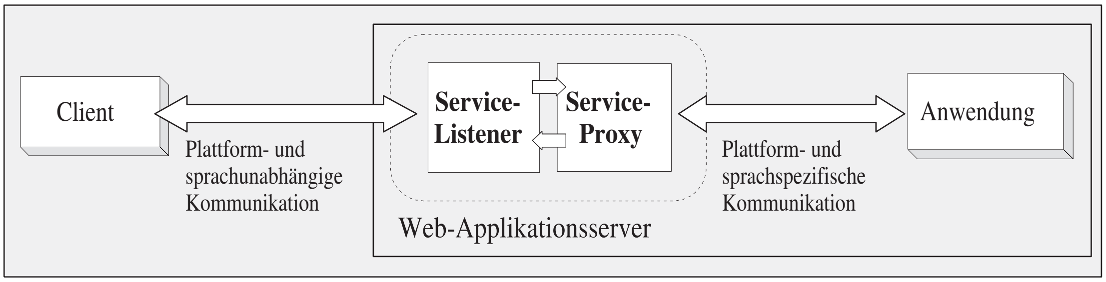

Abbildung 5.1.: Bestandteile und Sprachbeziehungen von Webservices (nach [4] S. 2f.)

Somit ist die Interoperabilität eines der herauszuhebenden Merkmale von Webservices. Eines der Protokolle, um Webservices zu realisieren, ist SOAP (Simple Object Access Protocol). SOAP ist eine Weiterentwicklung von XML-RPC, was gerade im Bereich der komplexen Datenstrukturen wie zum Beispiel Arrays sehr weitschweifig war und mehrfache zusammengefasste Methodenaufrufe, zum Beispiel in Form von Transaktionen, noch gar nicht unterstützte. SOAP bietet im Gegensatz zu XML-RPC mehrere Methodenaufrufe in einer Anfrage, was besonders bei transaktionsbasierten Anwendungen[1] von Vorteil ist.

[1] Anwendung mit hoher Zuverlässigkeit, bei der es von Vorteil ist, wenn entweder alle oder gar kein Methodenaufruf abgearbeitet wird.

5.1.1. Einordnung der Webservice-Instanzen

Wenn man sich den Stapel der Webservice-Techniken anschaut (Abb. 5.2), kann man eine Ähnlichkeit zum TCP/IP-Protokollstapel ziehen. An oberster Stelle steht eine Schicht, die zur Auf-

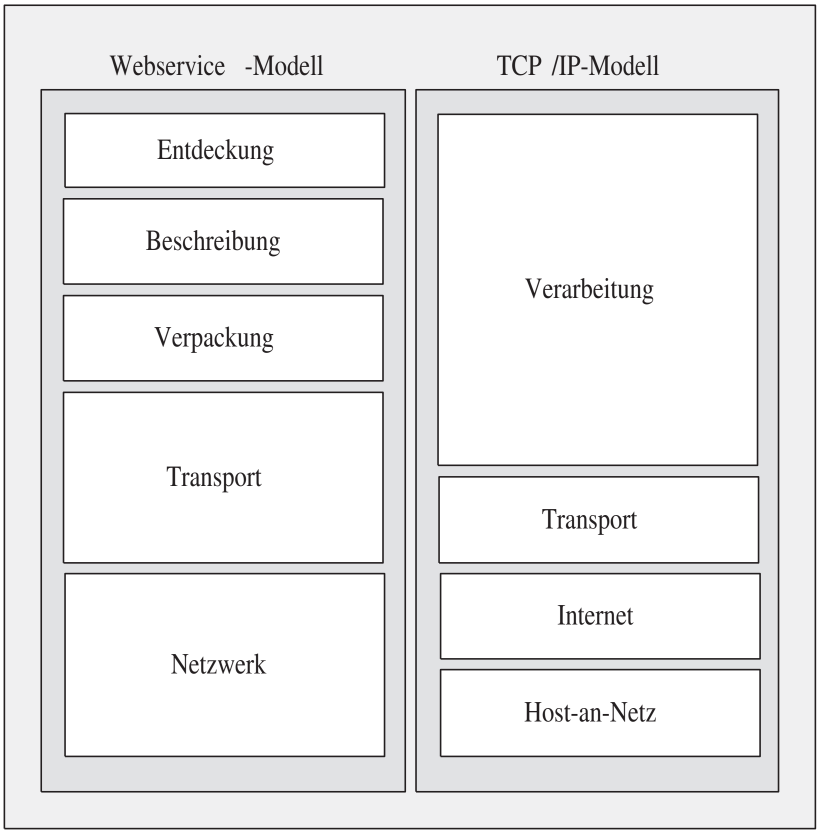

Abbildung 5.2.: Der Stapel der Webservice-Techniken, nach Snell, Tidwell, Kulchenko [4]

findung der Webservices dienen soll. Dazu existieren Technologien wie UDDI (Universal, Description, Discovery and Integration) oder WS-Inspection (Web Service Inspection Language). Beides sind Registraturen, bei denen vorhandene Webservices eingetragen und gesucht werden können. Diese sind allerdings noch in der frühzeitigen Standardisierungs- und Entwicklungsphase. Da bei beiden noch keine vernünftige Kategorisierung gegeben ist und die Dienste sich nur durch die vom Ersteller gegebenen Eigenschaften unterscheiden, ist eine anwendbare Skalierbarkeit zum jetzigen Zeitpunkt noch nicht gegeben. Deswegen wird im Weiteren auch auf diese Schicht nicht weiter eingegangen.

Mit der Beschreibungsschicht wird ein Dienst angeboten, der den Webservice so beschreibt, dass ein Service-Konsument diesen angebotenen Dienst auch nutzen kann. Dazu gibt es wiederum mehrere Sprachen, zum einen WSDL (Webservice Description Language), zum anderen RDF (Resource Description Framework) des W3C (World Wide Web Consortium) und DAML

(DARPA Agent Markup Language). Von diesen hat sich WSDL als die bei den vorhandenen Implementierungen vorherrschende Beschreibungssprache durchgesetzt.

Um die Anwendungsdaten plattform- und sprachunabhängig zur Verfügung zu stellen, ist die Verpackungsschicht vorhanden. Zum Vorgang des Packaging, der Serialisierung oder des Marshalling der Daten gehört unter anderem die Auswahl der akzeptierten Datentypen, die Codierung von Werten usw. XML ist dazu heutzutage die Grundlage der meisten Verpackungsverfahren, da XML einen semantischen Bezug zu den Daten herstellt und außerdem XML-Parser mittlerweile überall vorhanden sind. Unter den Verfahren, die XML als Verpackung der Daten benutzen, haben sich SOAP und XML-RPC als gebräuchliche Formate herausgestellt.

Die Transportschicht umfasst Dienste, die eine Datenübertragung der verpackten Daten gewährleisten sollen. Dazu gehören zum Beispiel TCP, SMTP, HTTP usw. Sie enthält im Webservice-Schichtenmodell sämtliche Protokolle, die auch im Internet-Protokollstapel die Transportschicht und zum Teil die Anwendungsschicht bilden. Somit schließt sich auch die Eingliederung des Webservice-Protokollstapels in den Internet-Protokollstapel an. Unterhalb der Verpackungsschicht kommt der unveränderte Protokollstapel mit Anwendung, Transport, Netzwerk und Verbindungsschicht des Internets zum Einsatz [4].

5.1.2. Gründe für die Nutzung von Webservices

Eine Grundidee von Webservices ist die Möglichkeit, eine Kommunikations-Applikation zu erstellen, ohne irgendetwas über die Architektur der Gegenstelle zu wissen. Aus diesem Grund verwenden Webservices die Datenbeschreibungssprache XML, um ein solches Kommunikationsprotokoll zu entwickeln.

XML hat folgende Vorteile ([30], S. 36):

- *Es ist ein offenes System. Mit XML kann man Daten plattform- und applikationsunabhängig austauschen.*
- *XML ist weitgehend selbstdokumentierend.*
- *Man kann Daten auch ohne vorhergehende Koordination austauschen.*

Die größten Vorteile von XML sind seine Plattformunabhängigkeit und die Selbstdokumentation. Bei einer gut definierten DTD ist es möglich, den Inhalt der Nachricht ohne weitere Hilfsmittel als einem Texteditor zu entziffern. Diese Möglichkeit ausnutzend haben sich schon verschiedene DTDs für die unterschiedlichsten Anwendungen gebildet, wie zum Beispiel die CML (Chemical Markup Language), cXML (CommerceXML9) usw.

Die DTDs haben den Vorteil, dass eine universelle Beschreibungssprache den Austausch von Informationen erheblich vereinfacht, da man sich nicht mehr um systemspezifische Dinge beim Entwurf einer solchen Sprache kümmern muss, sondern nur noch den zu übertragenden Inhalt und die Syntax beachten muss, in der diese Daten dann übertragen werden.

Ein weiterer Nebeneffekt durch die Verwendung von XML ist die Lesbarkeit der Daten durch Menschen ohne weitere Hilfsmittel.

XML ist auch wesentlich flexibler, was die Möglichkeit der Protokollbindung gegenüber einer binären Datenübertragung betrifft. Für die Übertragung von XML gibt es verschiedene

Ansätze ([30], S. 485ff), die weit verbreitete Internet-Protokolle zur Kommunikation der Nachrichten nutzen, wie zum Beispiel XML über FTP, XML über SMTP oder XML über HTTP.

5.2. Konzept und prinzipielle Funktionsweise von SOAP

SOAP ist im Webservices-Protokollstack auf der Verpackungsebene anzuordnen, dient also dazu, die zu übertragenden Daten zu serialisieren. Durch die Verwendung von XML als Serialisierungssprache hat man zwar einen erhöhten Aufwand in diesem Schritt, gewinnt aber gleichzeitig Sprach- und Plattformunabhängigkeit.

SOAP vereint zwei vorhandene XML-Nachrichtenaustausch-Anwendungen. Zum einen RPC (Remote Procedure Call), einer Grundlage für verteiltes Rechnen, zum anderen EDI (Electronic Document Interchange) als Grundlage der automatisierten geschäftlichen Transaktion.

5.2.1. Die SOAP-Nachricht

Wenn Daten zwischen heterogenen Systemen ausgetauscht werden, ist es wichtig, eine gemeinsame Kodierung zu definieren, damit sie jeder Teilnehmer richtig interpretieren kann.

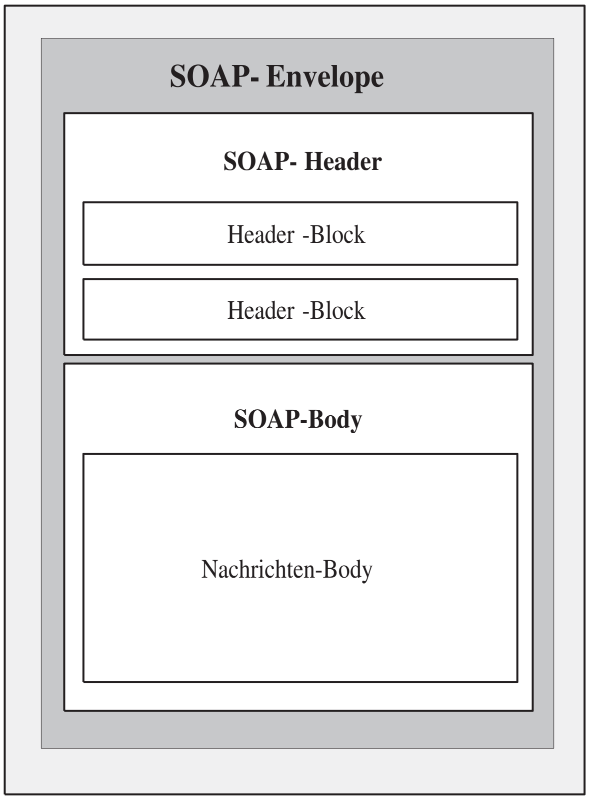

Abbildung 5.3.: Der SOAP-Envelope (vgl. [4], S. 16)

Dazu muss folgendes definiert werden (vgl. [4], S. 15):

- die Information, die ausgetauscht wird
- wie diese Information in XML codiert wird
- wie diese Information versendet wird

SOAP stellt für diese Anforderungen ein Rahmenkonstrukt zur Verfügung.

Eine SOAP-Nachricht ist im Grunde genommen eine “oneway”-Übertragung zwischen SOAP-Knotenpunkten, und zwar von SOAP-Sender zu SOAP-Empfänger. Anwendungen können dann komplexere Konstrukte wie zum Beispiel einen “request/response”-Nachrichtenaustausch mit diesem einfachen Rahmen implementieren.

Der Envelope (Umschlag) einer SOAP-Nachricht besteht aus zwei Teilen: Einem optionalen SOAP-Header (Kopf) und einem SOAP-Body (Hauptteil) (vgl. Abb. 5.3, S. 38).

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
                env:role="http://www.w3.org/2002/12/soap-envelope/role/next"
                env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
              env:role="http://www.w3.org/2002/12/soap-envelope/role/next"
              env:mustUnderstand="true">
   <n:name>Ake Jogvan Wyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
    <p:departing>New York</p:departing>
    <p:arriving>Los Angeles</p:arriving>
    <p:departureDate>2001-12-14</p:departureDate>
    <p:departureTime>late afternoon</p:departureTime>
    <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
    <p:departing>Los Angeles</p:departing>
    <p:arriving>New York</p:arriving>
```

```
28      <p:departureDate>2001−12−20</p:departureDate>
        <p:departureTime>mid−morning</p:departureTime>
30      <p:seatPreference/>
       </p:return>
32    </p:itinerary>
      <q:lodging xmlns:q="http://travelcompany.example.org/reservation/hotels">
34     <q:preference>none</q:preference>
      </q:lodging>
36   </env:Body>
    </env:Envelope>
```

Listing 5.1: Beispiel SOAP-Envelope [9]

Der SOAP-Header beginnt nach der obligatorischen Definition der XML-Version für XML-Dateien mit der Definition des Namespaces für den Envelope der SOAP-Nachricht in Zeile 2 von Listing 5.1. Der Header-Block der Nachricht ist dann von Zeile 2 bis 15 in der Nachricht enthalten.

Der Header enthält zwei Blöcke, die jeweils mit einem eigenen Namespace definiert sind (Zeile 4-9/10-14). Der Inhalt dieser Blöcke ist Teil der Anwendung (hier eine Reiseinformation) und nicht Teil der SOAP-Spezifikation. Der SOAP-Header wurde in Erwartung der vielfältigen Einsatzmöglichkeiten von SOAP geschaffen. Er dient zum einen der Anwendung, nicht nur anwendungsspezifische Nutzdaten zu übertragen, sondern auch Kontrollinformationen, die die Daten in irgendeiner Art und Weise beschreiben. Zum anderen dient der Header aber auch dazu, Kontrollinformationen von oder für so genannten SOAP-Intermediäre[2] zu speichern. Die Headerblöcke können von diesen Knotenpunkten inspiziert, eingefügt, entfernt oder weitergeleitet werden.

Der Hauptteil, der Body, ist ein notwendiger Teil einer SOAP-Nachricht (Zeile 16-38 in Listing 5.1). Im Body ist der primäre Inhalt der Nachricht zwischen Sender und Empfänger abgelegt. Auch hier müssen die Namensräume des Inhalts, sofern nicht schon im Header geschehen, einzeln definiert werden.

Die Trennung zwischen Kopf- und Hauptinformation muss in der Anwendung geschehen. Die Daten im Header können dazu benutzt werden, um auf dem Weg vom Sender zum Empfänger verschiedene Entscheidungen und Manipulationen in den einzelnen SOAP-Knoten zu beeinflussen. Es ist aber laut Spezifikation nicht ausgeschlossen, dass auch der Body der Nachricht verarbeitet[3] wird, obwohl davon abgeraten wird.

5.2.2. Der SOAP-Nachrichtenaustausch

SOAP in der Version 1.2 ist ein einfaches Rahmenwerk, XML-basierende Nachrichten zwischen einem Sender und einem endgültigen Empfänger auszutauschen.

Komplexere Formen des Nachrichtenaustauschs sind zum Beispiel “request-response”-Verfahren. In SOAP 1.1 wurde die Benutzung von RPCs [4] vorgeschlagen, um ein solches Verhalten

[2]SOAP-Knotenpunkt auf dem Weg der Nachricht von Sender zu Empfänger
[3]gelesen, manipuliert
[4]Remote Procedure Calls

zu modellieren. RPCs werden benutzt, um ein bestimmtes programmatisches Verhalten zu modellieren, bei dem die ausgetauschten Nachrichten von einem bestimmten, vorher vereinbarten Format sind.

Es wird von der SOAP 1.2 Spezifikation vorgeschlagen, dass ein größeres Einsatzgebiet im Bereich der “request-response”-Verfahren mit einfachem Austausch von XML-basierten Dokumenten in Form einer bestätigten Kommunikation modelliert wird.

5.2.2.1. SOAP-Remote Procedure Calls

Eines der Entwicklungsziele von SOAP 1.2 war es, die Funktionalität von RPCs in die Erweiterbarkeit und Flexibilität von XML zu kapseln.

Um einen SOAP-RPC einzuleiten, sind folgende Informationen nötig:

1. Die Adresse des SOAP-Zielknotens
2. Der Funktions- oder Methodenname
3. Die Typen und Werte der Eingangs- und Ausgangsdaten der Funktion oder Methode
4. Eine klare Trennung der Argumente, die die Netzressource, welche das Ziel vom entfernten Prozeduraufruf ist, beschreiben, von denen, die die Daten oder Kontrollinformationen für die Zielressource transportieren
5. Das Nachrichtenaustauschmuster, welches benutzt wird, um den RPC zusammen mit der Identifikation des so genannten Web-Service zu befördern
6. Optionale Daten, die als Teil der Headerblöcke befördert werden

Alle diese Informationen werden typischerweise in einer den Webservice beschreibenden formalen oder unformalen Definitionssprache, wie zum Beispiel WSDL (vgl. Kap. 5.3,S. 49), eindeutig beschrieben.

Die Adresse (1. Punkt) ist eine Adresse, anhand derer der endgültige Empfänger erkennen kann, dass die Nachricht an ihn gerichtet ist. Da das Übertragungsprotokoll nicht eindeutig definiert ist, ist die Adresse und deren Mechanismus zur Erkennung transportprotokollabhängig.

Mit den Punkten 4 und 5 soll die Kompatibilität mit den architektonischen Prinzipien des World Wide Web sichergestellt werden.

```
<m:chargeReservation
          env:encodingStyle="http://www.w3.org/2002/12/soap—encoding"
          xmlns:m="http://travelcompany.example.org/">
 <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
  <m:code>FT35ZBQ</m:code>
 </m:reservation>
 <o:creditCard xmlns:o="http://mycompany.example.com/financial">
  <n:name xmlns:n="http://mycompany.example.com/employees">
```

```
     Ake Bogvan Boyvind
18    </n:name>
      <o:number>123456789099999</o:number>
20    <o:expiration>2005-02</o:expiration>
     </o:creditCard>
22  </m:chargeReservation>
   </env:Body>
24 </env:Envelope>
```

Listing 5.2: Beispiel SOAP-RPC [9]

Um dieses zu erreichen, muss die Art der Datenkodierung angegeben werden. Dies geschieht mit dem Attribut `encodingStyle` des Envelopes (Z.11, Listing 5.2). Die Datenkodierung muss das “SOAP Data Model” unterstützen. Eine mögliche Repräsentation ist das “SOAP Encoding” (URI: http://www.w3.org/2002/12/soap-encoding).

Ein RPC-Aufruf enthält eine Struktur, die einen Eintrag für jeden Eingangs- und Eingangs-/Ausgangsparameter der Methode/Funktion enthält. Diese Struktur ist benannt nach dem Namen der addressierten Methode/Funktion. Jeder Parametereintrag sollte wiederum nach dem Parameternamen der Funktion benannt werden. Falls die Parameterliste unvollständig ist, so ist das Verhalten der Anwendung, die den RPC erhält, nicht definiert.

Eine RPC-Antwort besteht wiederum aus einer einzelnen Struktur, die jeden Ausgangs- und Eingangs-/Ausgangsparameter der Funktion/Methode enthält. Für die Namensgebung gelten dieselben Regeln wie für den RPC-Aufruf. Falls die Methode/Funktion einen Rückgabewert hat[5], so muss die Antwort einen result-Tag im Namensraum “http://www.w3.org/2002/12/soap-rpc” enthalten.

5.2.3. SOAP Fehlermeldungen

Fehler, die während der Bearbeitung eines RPCs auftreten, werden durch SOAP-Fehlermeldungen signalisiert, wenn dies möglich ist (protokoll und fehlerabhängig). Eine SOAP-Fehlermeldung ist eine SOAP-Nachricht mit einem einzelnen “env:Fault”-Element[6]. Das “Fault”-Element enthält zwei benötigte Elemente, “env-Code” und “env-Reason”, und optionale Elemente “env:Detail”, “env:Node” und “env:Role” (Listing 5.3).

```
 <env:Body>
  <env:Fault>
   <env:Code>
    <env:Value>env:Sender</env:Value>
    <env:Subcode>
     <env:Value>rpc:BadArguments</env:Value>
    </env:Subcode>
```

[5] der Rückgabewert ist nicht “void” oder leer

[6] env ist der Namensraum des SOAP-Envelope

```
    </env:Code>
    <env:Reason>
     <env:Text xml:lang="en-US">Processing error</env:Text>
     <env:text xml:lang="de-DE">Verarbeitungsfehler</env:Text>
    </env:Reason>
    <env:Detail>
     <e:myFaultDetails xmlns:e="http://travelcompany.example.org/faults">
      <e:message>Name does not match card number</e:message>
      <e:errorcode>999</e:errorcode>
     </e:myFaultDetails>
    </env:Detail>
   </env:Fault>
  </env:Body>
</env:Envelope>
```

Listing 5.3: Beispiel SOAP-RPC Fehlermeldung [9]

Das “Code”-Element enthält wiederum ein notwendiges und ein optionales Element. Das notwendige Element ist das “Value”-Element, welches eine standardisierte SOAP-Fehlermeldung enthält ([31], Kap. 5.4.6). Das optionale Element ist das “Subcode”-Element mit seinem “Value”-Unterelement. In ihm ist eine genauere Beschreibung zur standardisierten Fehlermeldung enthalten. Eine Erläuterung zu RPC-Fehlermeldungen ist in [32], Kapitel 4.4 beschrieben. Jedes “Subcode”-Element kann aber wiederum ein “Subcode”-Element enthalten. Eine solche Modellierung ist vorteilhaft, um einen einheitlichen Mechanismus zu erhalten, multiple Fehlermeldungen zu transportieren. Die SOAP-Knotenpunkte müssen aber nur die standardisierte Fehlermeldung des Top-Level-“Value”-Elementes verstehen.

Das “Reason”-Element ist zur umgangssprachlichen Beschreibung der Fehlermeldung gedacht. Es muss mindestens ein “Text”-Element mit der Fehlermeldung enthalten.

Das “Node”-Element sollte immer dann mit angegeben werden, falls die Meldung nicht vom endgültigen Nachrichtenempfänger generiert wird, sondern von einem Knotenpunkt auf dem Weg vom Sender zum Empfänger. Das Element enthält die URI des Knotens, welcher die Fehlermeldung erzeugt.

Das “Role”-Element gibt die Rolle des Knotens an, welcher die Fehlermeldung erzeugt hat([32], Kap. 2.2).

Das “Detail”-Element enthält anwendungsspezifische Fehlermeldungen, die im Namensraum der Anwendung angegeben werden.

5.2.4. Headerblock-Attribute

Wie schon vorher beschrieben, gibt es drei Typen von SOAP-Knotenpunkten: den Sender, den endgültigen Empfänger und den Intermediär. Ein SOAP-Knoten, der eine Nachricht empfängt, muss diese, wie im SOAP Processing Model ([31], Kap. 2) beschrieben, verarbeiten. Jeder SOAP-Knoten wird durch seine URI identifiziert. Die Headerblock-Attribute bestimmen nun das Verarbeitungsverhalten der SOAP-Knoten. Diese Attribute sind wie der ganze Header optional. Für jeden Headerblock werden einzelne Attribute gesetzt, so dass in einer Nachricht unterschiedliches Prozessverhalten modelliert werden kann. Ein Beispiel ist das Ablegen sämtlicher durchlaufener Knotenpunkte in einem Block, bei gleichzeitiger Nichtbeachtung der restlichen Blöcke.

Wichtig ist zu erwähnen, dass die Verarbeitungsregeln verlangen, den verarbeiteten Block zu entfernen. Diese Regel ist aus Sicherheitsgründen eingeführt worden, um ein Weiterreichen obsoleter oder fehlerhafter Blöcke zu vermeiden. Gleichzeitig soll damit verhindert werden, dass ein Intermediär Annahmen über die "Überlebensfähigkeit" eines Headerblockes macht, dessen "Rolle" er adaptiert hat. Jeder Knoten kann aber Blöcke hinzufügen. Wenn also nur ein lesender Zugriff auf einen Headerblock erfolgen muss, so kann er hinterher wieder in den Kopf der Nachricht kopiert werden.

Im Folgenden werden verschiedene Headerblockattribute beschrieben:

role: Jeder Knotenpunkt, der in eine definierte "Rolle" oder Funktion fällt, muss den Headerblock auswerten. Dabei wird die "Rolle" durch eine URI definiert. Die drei standardisierten "Rollen" und das Auswerteverhalten der SOAP-Knoten sind in Tabelle 5.1 aufgeführt. Neben diesen standardisierten "Rollen" können aber auch beliebig andere definiert werden. Die Blöcke werden dann auf den Knoten verarbeitet, die diese "Rollen" erkennen und nach Definition bearbeiten dürfen ([9] Kap. 3.1, [31] Kap 2.2).

mustUnderstand: Jeder Headerblock, der mit dem Attribut "mustUnderstand=true" gekennzeichnet ist, muss von den nach den "role"-Regeln identifizierten Knotenpunkten verarbeitet werden können[7]. Ist die Verarbeitung eines so gekennzeichneten Blocks nicht möglich, so darf die gesamte Nachricht nicht verarbeitet werden und es muss eine SOAP-Fehlermeldung erzeugt werden. Der Default-Wert ist "false" für dieses Attribut, was bedeutet, dass der Block verarbeitet werden kann, aber nicht muss ([9] Kap 3.2, [31] Kap. 2.4).

relay: Aus dem Grund, dass ein verarbeiteter Headerblock, der nicht verstanden wird, aus dem Headerblock entfernt wird, ist dieses Attribut eingeführt worden. Wenn eine neue Anwendungsfunktion eingeführt wird, welche im Header manifestiert werden soll, so kann der Fall eintreten, dass es auf dem Nachrichtenpfad neben den Knoten, die diese neue Funktion schon verstehen, auch noch unveränderte Knoten gibt. Um zu vermeiden, dass der Block von einem "alten" Knoten entfernt oder eine Fehlermeldung erzeugt wird, muss zum einen das Attribut "mustUnderstand=false" (gegen das Erzeugen der Fehlermeldung) gesetzt werden, zum anderen "relay=true" (gegen das Entfernen des Blocks) ([9] Kap. 3.3, [31] Kap. 2.7).

5.2.5. Die Protokollbindungen von SOAP

SOAP-Nachrichten können über eine Vielzahl von Protokollen übertragen werden. Wie in Abbildung 5.2 gezeigt, schließt das die Benutzung von Protokollen auf der Anwendungsschicht ein. Die Art, wie eine SOAP-Nachricht von Knotenpunkt zu Knotenpunkt übertragen wird, wird

[7] Der SOAP-Knoten kann den Block fehlerfrei "verstehen", wenn die Implementierung der Semantik konform zum XML qualifizierten Namen des äußersten Elementes des Headerblocks ist.

Tabelle 5.1.: Standardisierte role URIs und das Verarbeitungsverhalten der unterschiedlichen SOAP-Knotenpunkte

Kurzname	Name	initialer Sender	Intermediär	Empfänger
Attribut nicht vorhanden	-	X	Nein	Ja
“none”	“http://www.w3.org/2002/12/soap-envelope/role/none”	X	Nein	Nein
“next”	“http://www.w3.org/2002/12/soap-envelope/role/next”	X	Ja	Ja
“ultimateReceiver”	“http://www.w3.org/2002/12/soap-envelope/role/ultimateReceiver”	X	Nein	Ja
X=Regel nicht anwendbar				
“none” wird formal nie verarbeitet; solche Blöcke können Daten enthalten, die für die Verarbeitung anderer Blöcke nötig sind.				

“SOAP-Binding” genannt. “SOAP-Bindings” müssen der SOAP-Bindungsspezifikation in [31], Kapitel 4 entsprechen. Die SOAP-Nachricht selber wird in der Form eines XML-InfoSets [33] übertragen. Die Protokollbindung hat die Aufgabe, das SOAP XML-InfoSet `env:Envelope` in einer Art und Weise zu serialisieren, dass es zum nächsten SOAP-Knoten übertragen und dort ohne Informationsverlust rekonstruiert werden kann.

Dabei ist die ganze Spezifikation flexibel ausgelegt. So können verschiedene Eigenschaften, die das Protokoll, welches die Bindung zwischen den Knotenpunkten herstellt, für die Anwendung nicht bietet, durch Headerblöcke modelliert werden. Diese Art der Modellierung durch Headerblöcke wird “SOAP-Module” genannt.

Durch diese Modellierung ist es nicht erforderlich, dass alle Knotenpunkte in einer SOAP-Verbindung dieselbe Protokollbindung benutzen, sondern es ist für jede Verbindung eine andere möglich. Damit muss eine einheitliche Protokollbindung über die gesamte Nachrichtenstrecke nicht gewährleistet sein, um einen bestimmten Transportdienst zu gewährleisten. Somit sind die “SOAP-Module” eine sinnvolle Einrichtung, bestimmte Verbindungseigenschaften zu modellieren und zu gewährleisten.

5.2.6. HTTP

Die Protokollbindung an HTTP ist neben einem kurzen Vorschlag einer Bindung an SMTP die einzige in die Spezifikation aufgenommene Bindung. HTTP wurde wegen mehrerer Gründe ausgewählt. Zum einen ist es ein weit verbreitetes Protokoll, welches von fast allen Servern unterstützt wird. Zum anderen ist es der Port 80, den HTTP standardmäßig verwendet. Dieser Port ist einer der wenigen, der in den meisten Firewalls geöffnet ist, und somit eine Kommunikation über diesen Port zulässt. Damit, dass nur der eine, schon verwendete Port benutzt wird, erspart man sich Probleme, die durch die dynamische Protokollallokation der verschiedenen ORBs wie CORBA immer wieder auftreten. Durch diese Wahl ist die Kommunikation auch durch Firewalls möglich. Dies führte natürlich zu dem Problem, wie SOAP-Nachrichten erkannt werden können, um sie zum Beispiel zu filtern oder abzublocken. Aus dem Grund wurde das "SOAPAction"-Element im HTTP-Header in der Spezifikation 1.1 eingeführt, wegen Problemen aber in der vorliegenden Version 1.2 wieder abgeschafft, da der Wert des Elementes völlig beliebig und so eine vernünftige Selektion nicht möglich war. In der Version 1.2 erfolgt die Erkennung einer SOAP-Nachricht anhand des HTTP-Header-Feldes "Content-Type".

Webmethoden sind die Methoden `GET, POST, PUT, DELETE`, welche in der Spezifikation zu HTTP 1.1 definiert sind. Die Benutzung dieser Methoden ist explizit in den Spezifikationen zu SOAP aufgenommen worden, um dem Anwendungsentwickler eine zur Internetarchitektur kompatible Protokollbindung zu gewährleisten. Bei der Benutzung von HTTP als Transportprotokoll der Nachrichten beschränkt sich die SOAP 1.2 Spezifikation explizit auf die `POST`- und `GET`-Methoden. Dabei wird `POST` im "request-response"-Nachrichtenaustausch eingesetzt und `GET` als "response"-Verfahren.

5.2.6.1. HTTP-Header

Beim Transport einer SOAP-Nachricht im Rumpf einer HTTP-Nachricht sind laut Spezifikation folgende Angaben im HTTP-Header zu machen [32]:

HTTP-METHOD: als Methode muss, da nicht weiter spezifiziert, `GET` oder `POST` angegeben werden (mehr dazu s.u.).

Request URI: Die Adresse des Empfängers

Content-Type: Sollte bei SOAP-Nachrichten auf `application/soap+xml` gesetzt werden, nur wenn das XML-InfoSet leer ist, dann darf diese Angabe entfallen (siehe `GET`, Kap. 5.2.6.2).

Accept: Diese Angabe ist optional und gibt den akzeptierten Rückgabetyp an.

Des Weiteren sind zusätzliche Header-Felder nach RFC 2616 [34] zugelassen.

5.2.6.2. GET

Bei einem HTTP-GET-Request ist zu beachten, dass keine über die Adressierung hinausgehenden Informationen vom Sender zum Empfänger übermittelt werden. Dies wird in einem exemplarischen GET-Request in Listing 5.4 gezeigt. In der Adressierung ist natürlich die normale HTTP GET-Codierung von Eingabedaten möglich (in diesem Fall wird der Funktion "reservations" die Variable "code" mit übergeben).

```
GET /travelcompany.example.org/reservations?code=FT35ZBQ HTTP/1.1
Host: travelcompany.example.org
Accept: text/html, application/soap+xml
```

Listing 5.4: SOAP: HTTP GET Request [9]

Der `Accept`-Teil des Kopfes gibt die akzeptierten Rückgabeformate an, die der Webserver zurückliefern kann. In diesem Fall sind das HTML-Dateien oder XML-kodierte SOAP-Dateien.

Die Antwort könnte wie in Listing 5.5 aussehen.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
               env:role="http://www.w3.org/2002/12/soap-envelope/role/next"
               env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-30T16:25:00.000-05:00</m:dateAndTime>
  </m:reservation>
 </env:Header>
 <env:Body>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:x="http://travelcompany.example.org/vocab#"
          env:encodingStyle="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <x:ReservationRequest
       rdf:about="http://travelcompany.example.org/reservations?code=FT35ZBQ">
    <x:passenger>Ake Jogvan Ayvind</x:passenger>
     <x:outbound>
      <x:TravelRequest>
       <x:to>LAX</x:to>
       <x:from>LGA</x:from>
       <x:date>2001-12-14</x:date>
      </x:TravelRequest>
     </x:outbound>
     <x:return>
      <x:TravelRequest>
      <x:to>JFK</x:to>
      <x:from>LAX</x:from>
      <x:date>2001-12-20</x:date>
     </x:TravelRequest>
    </x:return>
   </x:ReservationRequest>
  </rdf:RDF>
 </env:Body>
```

```
</env:Envelope>
```

Listing 5.5: SOAP: HTTP GET-Response [9]

5.2.6.3. POST

Um RPCs auszuführen, ist die Verwendung von HTTP-POST nötig. Listing 5.6 zeigt einen solchen Request. Gegenüber der GET Methode ist hier ein kompletter “SOAP-Envelope” im “BODY” der Nachricht enthalten. Die einzige Anforderung an die Adresse ist, dass es eine gültige URI [35] sein muss, sonst gibt es keine formalen Restriktionen an die Adresse. Es ist aber von der SOAP-Spezifikation angeraten, innerhalb der URI den Methodenaufruf zu kodieren (wie in Listing 5.6).

Wenn SOAP-Nachrichten mittels HTTP übertragen werden, ist es erforderlich, den Typ der Nachricht mit `application/soap+xml` anzugeben.

```
POST /Reservations?code=FT35ZBQ HTTP/1.1
Host: travelcompany.example.org
Content—Type: application/soap+xml; charset="utf—8"
Content—Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap—envelope" >
 <env:Header>
  <t:transaction xmlns:t="http://thirdparty.example.org/transaction"
                env:encodingStyle="http://example.com/encoding"
                env:mustUnderstand="true" >5</t:transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2002/12/soap—encoding"
      xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
```

Listing 5.6: SOAP: HTTP POST-Request (gekürzt) [9]

Die Antwort auf solch einen Request, sieht ähnlich aus wie die Antwort auf den GET-Request in Listing 5.5.

5.2.6.4. Fehlermeldungen

Wenn während der Bearbeitung der Anfrage ein Fehler auftritt, so ist, wie schon erwähnt, eine SOAP-Fehlermeldung zurückzugeben. Die Spezifikation der “HTTP-Binding” von SOAP verlangt, dass die Fehlermeldung in einer HTTP-Fehlermeldung vom Typ `"HTTP 500 Internal Server Error"` transportiert wird. Weitere Fehlermeldungen, die beim Bearbeiten der HTTP-Nachricht auftreten können, sind genauer in [32], Kapitel 7.5.1.2 beschrieben ([4, 9, 31, 32]).

5.2.7. SOAP-Encoding

Ein Ziel von Webservices ist es, Plattform- und Sprachunabhängigkeit zu gewährleisten. Deswegen bezeichnet die SOAP-Spezifikation den SOAP-Codierungsstil als "ein einfaches Typensystem, das eine Erweiterung der gemeinsamen Merkmale ist, die in den Typensystemen von Programmiersprachen, Datenbanken und halbstrukturierten Daten angetroffen werden, unabhängig von den kleineren Unterschieden, die zwischen diesen Umgebungen existieren" ([4], S. 29).

Diese Spezifikation sagt aus, dass die Art des Codierungsstils vollkommen freigestellt ist. Es kann innerhalb des SOAP-Envelopes ein beliebiges XML-Dokument transportiert werden.

Um eine gewisse Standardisierung anzubieten, bietet die SOAP-Spezifikation das schon erwähnte SOAP-Encoding Format an. Das SOAP-Encoding definiert drei Arten den Datentyp eines Accessors auszudrücken:

- Es kann die Typdefinition nach der XML-Schema-Spezifikation in jedem Datentyp einzeln erfolgen.

```
<person>
  <name xsi:type="xsd:string">Max Mustermann</name>
<person>
```

Listing 5.7: SOAP-Encoding Variante mit expliziter Typangabe

- Es kann eine Angabe erfolgen, welche auf ein XML-Schema-Dokument verweist, in dem die Datentypen für jeden Accessor festgelegt sind.
- Es kann ein Verweis auf eine andere Art von Schema-Dokument erfolgen, welche den Datentyp innerhalb des Dokumentes festlegt.

Diese freie Art Datentypen festzulegen, führte natürlich dazu, dass frühe Implementierungen der SOAP-Spezifikation gerade diesen Punkt frei auslegten. So wurde bei den Implementierungen von IBM, gSOAP und des Webservers Apache die explizite Angabe (Punkt 1) gewählt, um die Datentypen zu definieren, während bei der Implementierung von Microsoft auf die Verwendung von Schemas gesetzt wurde. Das heißt, dass vorhandene Implementierungen in diesem Punkt keinesfalls vollständig der Spezifikation entsprechen und eine vollständige Kompatibilität, die ja eine Zielvorgabe darstellt, noch nicht gegeben ist.

5.3. Die Webservice Description Language

Eines der Schlüsselelemente eines Webservices ist die Selbstbeschreibung der angebotenen Dienste. Eine Anwendung oder Dienst stellt bestimmte Funktionalitäten zur Verfügung. Dies kann beinhalten, dass der Ablauf mit bestimmten Optionen oder auch Parametern beinflusst werden kann. Das heißt, dass der Dienstnehmer diese Optionen in Form von Informationen der Anwendung mitteilen können muss. Nach Ablauf der Bearbeitung durch die Anwendung muss eine Rückgabe der angeforderten Informationen möglich sein.

Konventionelle Ansätze haben in diesen Punkten auf genau spezifizierte Ablauf- und Kommunikationsprotokolle gesetzt. Gerade die Bereitstellung der Information, der Serialisierung der Daten, hat es oft schwer gemacht, Plattform- und Sprachunabhängigkeit zu schaffen.

Mit WSDL ist es nun möglich, die angebotenen Dienste so zu spezifizieren, dass eine Informationsabfrage ohne weiteres möglich ist. Dazu wird die den Dienst beschreibende WSDL-Datei abrufbereit auf einen Server gelegt. Wenn ein Client jetzt den Dienst benutzen möchte, so muss er nur die in der WSDL-Datei angegebenen Spezifikationen einhalten.

Eine WSDL-Datei setzt sich aus den fünf Blöcken “`definition, types, message, portType` und `binding` zusammen. Zur Verdeutlichung ist in Listing 5.8 als Beispiel eine WSDL-Beschreibung angegeben, in der zwei Dienste angeboten werden, zum einen ein einfacher “Hello World”-Dienst, zum anderen eine Additionsroutine.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Service"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="urn:hellonamespace"
    xmlns:tns="urn:hellonamespace"
    xmlns:SOAP-ENV="http://www.w3.org/2002/06/soap-envelope"
    xmlns:SOAP-ENC="http://www.w3.org/2002/06/soap-encoding"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ns="urn:hellonamespace">

<types>
 <schema
     xmlns="http://www.w3.org/2001/XMLSchema"
     targetNamespace="urn:hellonamespace"
     xmlns:SOAP-ENV="http://www.w3.org/2002/06/soap-envelope"
     xmlns:SOAP-ENC="http://www.w3.org/2002/06/soap-encoding"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:ns="urn:hellonamespace">
 </schema>
</types>

<message name="greetRequest">
 <part name="myname" type="xsd:string"/>
```

```
</message>

<message name="greetResponse">
 <part name="result" type="xsd:string"/>
</message>

<message name="addRequest">
 <part name="a" type="xsd:double"/>
 <part name="b" type="xsd:double"/>
</message>

<message name="addResponse">
 <part name="result" type="xsd:double"/>
</message>

<portType name="ServicePortType">
 <operation name="greet">
  <documentation>Service definition of function ns__greet</documentation>
  <input message="tns:greetRequest"/>
  <output message="tns:greetResponse"/>
 </operation>
 <operation name="add">
  <documentation>Service definition of function ns__add</documentation>
  <input message="tns:addRequest"/>
  <output message="tns:addResponse"/>
 </operation>
</portType>

<binding name="ServiceBinding" type="tns:ServicePortType">
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="greet">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded"
             namespace="urn:hellonamespace"
             encodingStyle="http://www.w3.org/2002/06/soap-encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded"
             namespace="urn:hellonamespace"
             encodingStyle="http://www.w3.org/2002/06/soap-encoding"/>
  </output>
 </operation>
 <operation name="add">
```

```
72    <SOAP:operation soapAction=""/>
      <input>
74     <SOAP:body use="encoded"
                  namespace="urn:hellonamespace"
76                encodingStyle="http://www.w3.org/2002/06/soap-encoding"/>
      </input>
78    <output>
       <SOAP:body use="encoded"
80                namespace="urn:hellonamespace"
                  encodingStyle="http://www.w3.org/2002/06/soap-encoding"/>
82    </output>
     </operation>
84  </binding>

86  <service name="Service">
     <documentation>gSOAP 2.1.10d generated service definition</documentation>
88   <port name="ServicePort" binding="tns:ServiceBinding">
      <SOAP:address location="http://location/Service.cgi"/>
90   </port>
    </service>
92
    </definitions>
```

Listing 5.8: WSDL Beispiel

Eine WSDL-Beschreibung beginnt mit einem `definitions`-Tag, dessen Attribute sich dem `name`-Attribut mit dem Namen der Anwendung und den zu definierenden Namensräumen zusammensetzt (Listing 5.8, Zeile 2-12).

In dem folgenden Element werden die benutzten Datentypen der Nachrichten des Dienstes spezifiziert. Das kann zum einen explizit in der Beschreibung mit einem `types`-Block (Listing 5.8, Zeile 14-22) geschehen, zum anderen bietet die WSDL-Spezifikation aber auch das `import`-Tag an, um externe Schema-Definitionen zu importieren. Gerade letztere Möglichkeit wird aber noch nicht von vielen WSDL- benutzenden Werkzeugen unterstützt ([4], S. 95).

Nach der Beschreibung der benutzten Datentypen erfolgt die Beschreibung der Webservice-Schnittstelle (Listing 5.8, Zeile 25-55). Dazu werden zuerst in den `message`-Blöcken die Eingangs- und Ausgangsdatentypen der Methoden des Webservices beschrieben. Nach dieser Beschreibung werden die so erzeugten Nachrichtenblöcke in der Beschreibung der einzelnen Methoden verwendet. Die Beschreibung der Methoden eines Webservices findet im `portType`-Block statt (Listing 5.8, Zeile 44-55). In diesem `portType`-Block befindet sich zu jeder Methode ein `operation`-Block, der neben der Beschreibung der Methode in den `documentation`-Tags die in den `message`-Blöcken erzeugten Eingangs- und Ausgangsnachrichten einer Methode enthält.

Im Anschluss an die Beschreibung der Methoden findet die Beschreibung der Schnittstelle zur Außenwelt des Services statt. Dies wird analog zu der SOAP-Schnittstelle auch hier "Binding" genannt. Im `binding`-Block (Listing 5.8, Zeile 56-77) findet zum einen die Beschreibung der Bindung an das Transportprotokoll, hier HTTP, zum anderen die Art der Verpackung der über das Transportprotokoll transportierten Nachricht, hier SOAP, statt.

Das letzte Element in der Beschreibung eines Webservices stellt die Ortsangabe dar, die es erst ermöglicht, den angebotenen Dienst zu nutzen. Dies erfolgt im `service`-Block (Listing 5.8, Zeile 79-84). In ihm wird mit dem Element `port` die Adresse des Dienstes beschrieben, unter der er erreichbar ist. Es ist möglich, an dieser Stelle mehrere Adressen anzugeben, um zum Beispiel die Verfügbarkeit zu erhöhen [4, 36, 37].

6. Realisierung des Managementszenarios mittels Webservices

In diesem Kapitel soll der Weg der Entwicklung einer Managementlösung mit Mitteln, die mit den Webservices zur Verfügung stehen, beschrieben werden. Dazu ist in der Anfangsphase analog zur Entwicklung einer SNMP-Lösung in Kapitel 4.1.1 eine Analyse des vorhandenen Problems nötig. Der große Unterschied ist die Modellierung der Managementinformationen und deren Zugriffe.

6.1. Konzept des Prototypen

Auf der Grundlage des eingeschränkten Anforderungsmodells aus Kap. 2.2.1 (S. 8) ist der aus drei Teilen bestehende Prototyp eines Webservice-Managementansatzes entstanden. Die Anforderungen an diesen Prototypen bestanden im wesentlichen in der Konfiguration und Beobachtung der in einem Gerät vorhandenen Schnittstellen.

Nach der Objektanalysephase, die, wie schon beschrieben, analog zur Analysephase in Kapitel 4.1.1 erfolgen kann, und nach der Festlegung der Organisationsstruktur der Kommunikationspartner, kommt die für die Entwicklung einer Webservice-Management-Lösung wichtigste Phase, nämlich die Abbildung dieser ermittelten Objekte auf Webservice-Datenstrukturen und Zugriffsfunktionen. In Abbildung 6.1 ist der Ablauf der Modellierung eines Webservices zu sehen.

Das Ergebnis dieser Phase ist eine Webservice-Beschreibung, zum Beispiel WSDL, anhand derer dann parallel die Implementierungsphasen von Client- und Serveranwendungen erfolgen können.

Im Folgenden wird der Webseviceansatz anhand der Systematik der Modellierung eines integrierten Managementansatzes aus Kapitel 2.1 beschrieben.

6.1.1. Organisationsmodell

Um eine, den Anforderungen genügende Stuktur zu bilden, wurden die Aufgaben wie in Abbildung 6.2 zu sehen aufgeteilt. Der **Agent** ist für Zugriff, Bereitstellung und Modellierung der Managementinformationen auf dem zu managenden Gerät zuständig. Dazu implementiert er die Dienste, die mit den Namensräumen `WS_std_stdDefs`, `WS_std_Interface`, `WS_std_Route` und `WS_std_System` gekennzeichnet sind. Neben diesen Diensten implementiert der Agent die Funktion `WS_std_Event::registerEventhandler()` die, wie später noch beschrieben, Teil des Ereignismanagements des Agenten ist.

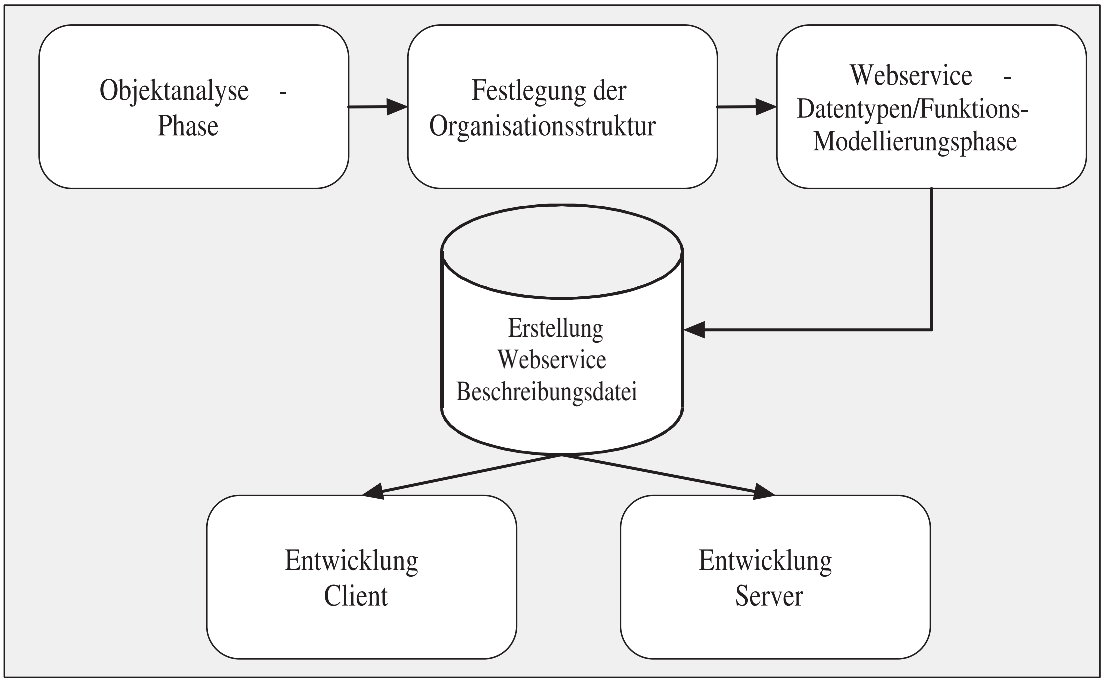

Abbildung 6.1.: Schematischer Entwicklungsverlauf eines Webservices

Der **Manager** ist eine reine Umsetzung der Webservice-Funktionen auf die Kommandozeile und benutzt dementsprechend die Funktionen der importierten Namensbereiche.

Der **Eventreceiver** ist für das Empfangen von möglichen Ereignismeldungen eines Agenten zuständig. Dazu implementiert er die Funktion `WS_std_Event::signalEvent()` und benutzt `WS_std_Event::registerEventhandler()`.

Die Kommunikationsbeziehung zwischen Agent und Manager ist eine reine Client-Server-Beziehung, in der der Agent als Dienstanbieter und der Manager als reiner Dienstnehmer auftritt. Dabei implementiert der Agent sämtliche Funktionen, mit denen der Manager mit Hilfe von Parametern das Gerät, auf dem der Agent läuft, managen kann. Mit diesem Konzept sollte es bei klar definierten Diensten zu keinen signifikanten Misskonfigurationen des Gerätes kommen, da der Agent für eine (Validitäts-)Überprüfung der eingehenden Daten (Parameter) verantwortlich ist und im Fehlerfall mit einer dezidierten `SOAP-FAULT`-Fehlermeldung (Kap. 5.2.3, S. 42) antworten sollte. Der Agent übernimmt somit die lokalen Managementaufgaben und Funktionen eines Gerätes, den globalen Netzwerkzusammenhang wiederum muss ein geeigneter Manager herstellen.

Die Kommunikationsbeziehung zwischen Eventreceiver und Agent ist als Peer-to-Peer-Beziehung modelliert worden. Dies hat den Hauptgrund darin, dass Ereignismeldungen in diesem Managementmodell bestätigte Meldungen sind. Beide Kommunikationspartner bieten gegenseitig Funktionsschnittstellen an, um diese Funktionalität zu gewährleisten. Der Agent bietet, wie später noch genauer beschrieben wird, dem Eventreceiver neben der Möglichkeit sich für den ei-

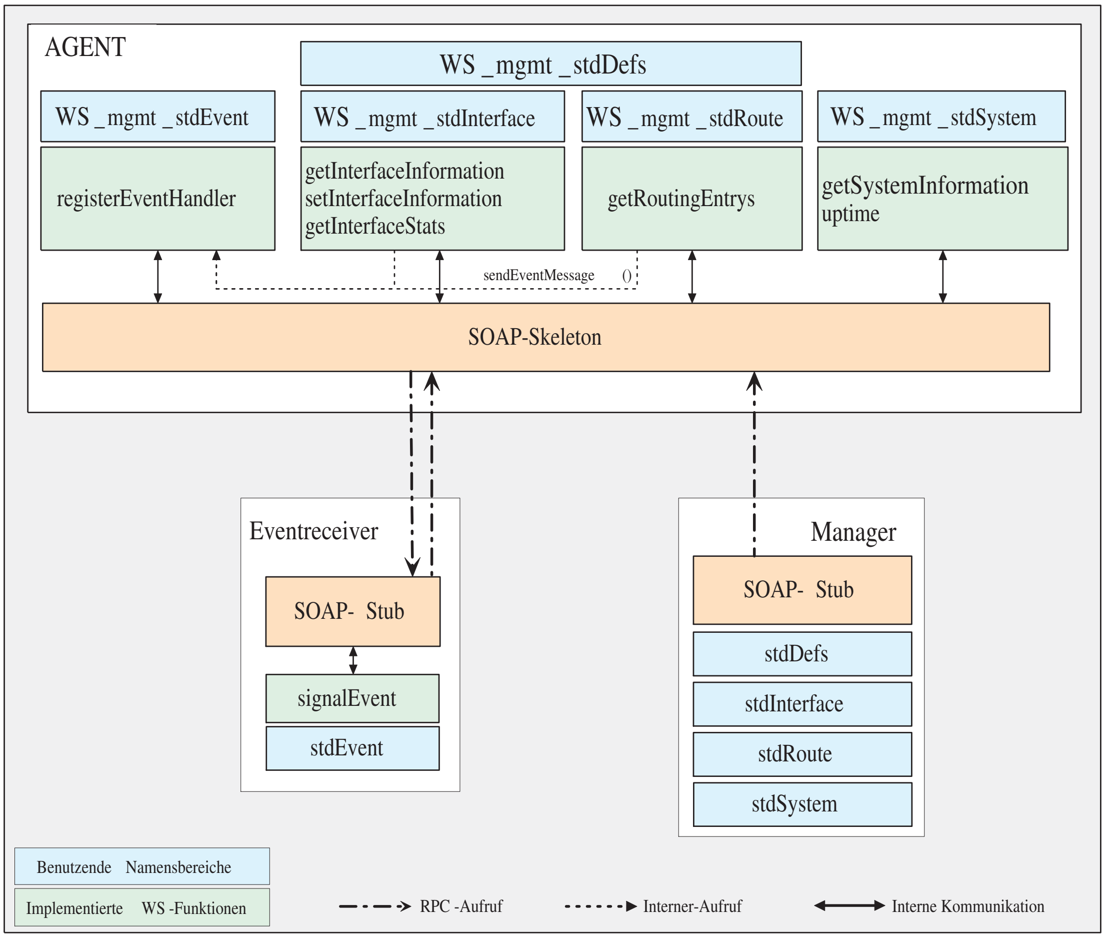

Abbildung 6.2.: Strukturmodell der Webservice Lösung

gentlichen Empfang der Daten zu registrieren, die Möglichkeit sich nur für bestimmte Ereignisse einzuschreiben. Der Eventreceiver wiederum muss dem Agenten eine Funktion zum Mitteilen der Ereignismeldungen bieten. Für die Interpretation der Ereignisse ist der Eventreceiver zuständig.

6.1.2. Die Webservice Funktions-Prototypen / Informationsmodell

Neben der reinen Strukturierung der generellen Aufgaben in einem Webservice-Managementmodell ist es nötig, eine Strukturierung der zu managenden Informationen vorzunehmen. Da der Agent einen großen Anteil am Management des Gerätes hat, ist zuerst eine generelle Einteilung in Aufgabenbereiche nötig, in denen man dann die managementspezifischen Funktionen des Aufgabenbereichs festlegen kann.

Um eine Strukturierung der Funktionen in Aufgabenbereichen zu erreichen, sind die Funktionen in verschiedene Namensbereiche unterteilt worden.

Die Namensbereiche und die Funktionen sind in der folgenden Aufzählung erklärt. Eine ausführliche Beschreibung der Funktionen findet in den Kapiteln 6.2.2 und 6.2.3 statt:

WS_mgmt_stdDefs Im Bereich dieses Namensraums wurden Strukturen und Definitionen festgelegt, die in allen anderen Namensbereichen gemeinsam benutzt werden können. In diesen Bereich fällt zum Beispiel die Definition einer Adress-Struktur, die in den anderen Bereichen genutzt wird (vgl. Anhang B.2.1, S. 99).

WS_mgmt_stdSystem Dieser Namensraum wurde eingeführt, um generelle Informationen über das gemanagten Gerätes zu modellieren. Mit den Funktionen `uptime()` und `getSystemInformation()` kann auf diese Informationen lesend zugegriffen werden.

WS_mgmt_stdInterface Mit den Funktionen dieses Namensraums findet das Management der Schnittstellen eines Systems statt. Die Funktionen `getInterfaceInformation()` und `getInterfaceStats()` geben Auskunft über die aktuellen Statistiken und Zustand der Schnittstellen, mit `setInterfaceInformation()` können die Schnittstellen des Gerätes konfiguriert werden.

WS_mgmt_stdRoute Dieser Namensbereich enthält die Funktion `getRoutingEntrys()`, mit dem Informationen der aktuellen Routingtabelle erfragt werden können.

WS_mgmt_stdEvent Dieser Namensbereich umfasst das Management der Ereignisse und der damit zusammenhängenden Meldungen. Mit der Funktion `registerEventHandler()` kann man sich für den Empfang von Ereignismeldungen des Agenten registrieren. Voraussetzung für den Empfang dieser Nachrichten ist die Implementierung der Funktion `signalEvent()`.

Die eindeutige Identifizierung und Beschreibung von Managementinformationen findet durch die WSDL-Beschreibung der durch den Agenten bereitgestellten Informationen statt.

6.1.3. Das Kommunikationsmodell

Als Kommunikationsprotokoll ist SOAP mit seinen Möglichkeiten, wie in Kapitel 5.2 beschrieben, im Rahmen der Webservice-Architektur gewählt worden. Durch die Verwendung von XML als Transportsprache ergibt sich für die Serialisierung und Deserialisierung der Daten eine andere Struktur als in klassischen Übertragungssystemen. Durch die Eigenschaft von XML, die enthaltenen Daten mit Hilfe von Elementen auch semantisch zu beschreiben und für Menschen lesbar zu halten, ergibt sich auch für das Kommunikationsdesign die Voraussetzung, die Daten in einer geeigneten Art und Weise zu modellieren, die diesen Voraussetzungen entsprechen. Durch die Verwendung von SOAP tritt für den Entwickler die eigentliche Kommunikation in den Hintergrund (bei SOAP durch die Protokollbindung gewährleistet) und die Anordnung der Daten in den Vordergrund. Da man mit der Verwendung von XML auch nicht an die üblichen Datentypenbeschränkungen gebunden ist, ist bei der Modellierung der Daten größtmöglicher Wert auf den semantischen Zusammenhang und die Beschreibung der Daten in einer Nachricht gelegt worden.

6.1.4. Das Funktionsmodell

Das Funktionsmodell deckt in dem beschriebenen Prototyp Teilbereiche des Konfigurations-, Fault- und Performancemanagements im Bezug auf die Schnittstellen- und Routinginformationen ab.

Das Konfigurationsmanagement ist der am weitesten ausgeprägte Bereich, auf beide Managementteilbereiche kann lesend zugegriffen werden, auf die Schnittstelleninformationen auch schreibend.

Im Performancemanagement ist ein Beispiel in Bezug auf die Schnittstellen modelliert worden.

Das Faultmanagement umfasst die Signalisierung von Änderungen in der Konfiguration der Schnittstellen und Routingdaten.

6.2. Modellierung des Agenten / Eventreceivers

Aus den Anforderungen des Szenarios und den Unterteilungen der Namensbereiche ist mit Webservice-Mitteln der Prototyp eines Agenten für eine managementfähige Lösung entwickelt worden.

Da der Agent und der Eventhandler durch ihre Kommunikation untereinander in einem direkten Abhängigkeitsverhältnis stehen, werden diese beiden Teile der Lösung hier zusammen beschrieben.

Als Programmiersprache für diesen Prototyp ist C gewählt worden. Für die Entwicklung eines Webservices in C gibt es die *gSOAP* Compiler-Werkzeuge für C/C++.

6.2.1. gSOAP

gSOAP ist ein Werkzeug, um mit C/C++ auf eine einfache Art und Weise Webservices mit SOAP/XML zu entwickeln. Dazu bietet gSOAP zum einen eine SOAP-API, um die Kommunikationsleistung zu erbringen, und zum anderen einen Compiler, um die nötigen Dateien dafür zu generieren. Im Gegensatz zu den meisten SOAP-APIs, die einen SOAP-zentrierten Ansatz verfolgen, das heißt, dass man seine Anwendung speziell auf die SOAP-spezifischen Konstrukte anpassen muss, verfolgt die gSOAP-API einen transparenten Ansatz, der irrelevante SOAP-spezifische Daten vor dem Benutzer verbirgt. Dies führt dazu, dass man, nachdem man sich ein wenig in das Erstellen der Definitionsdateien eingearbeitet hat, die gSOAP-API ähnlich wie die Standard-Berkley-Socket-Bibliothek benutzen kann, da die Definitionsdateien sehr ähnlich den C-Header Dateien sind.

Zu diesem Zweck werden aus einer Definitionsdatei, die ähnlich der C/C++ Headerdateien ist, ein SOAP-Kommunikations-Skeleton und eine WSDL-Datei erzeugt (vgl. Abb. 6.3 S. 60). In diesem Skeleton sind alle in der Definitionsdatei beschrieben Funktionen als Funktionsprototypen enthalten, die dann mit den zur Implementierung des Dienstes nötigen Funktionalitäten gefüllt werden können.

Nach der Implementierung können die Programmdateien mit dem Skeleton zusammen mittels eines C-Compilers zu einem lauffähigen Programm kompiliert werden.

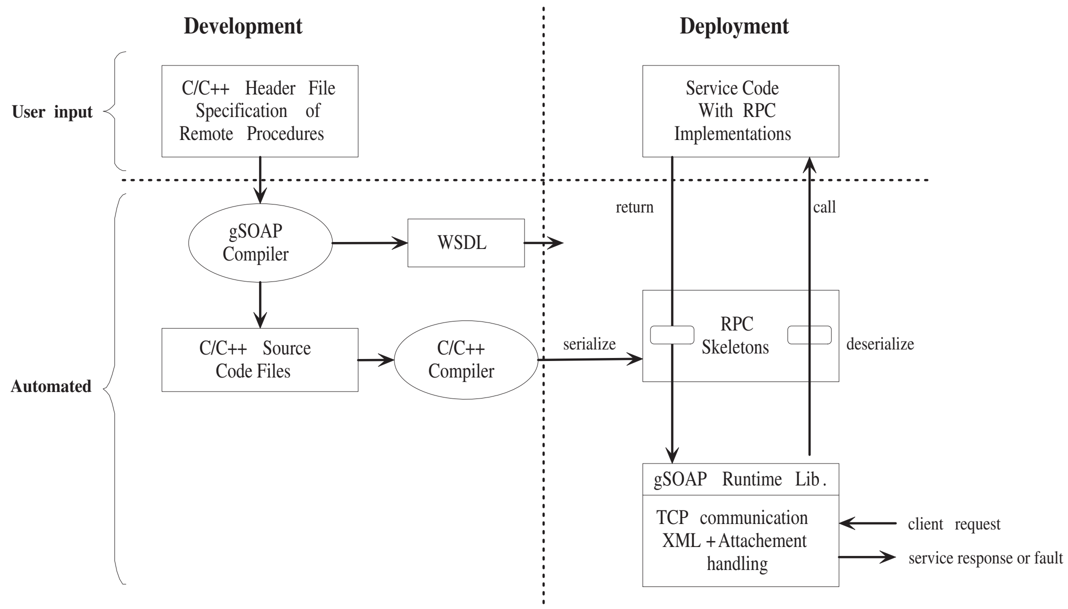

Abbildung 6.3.: Erzeugung eines Webservices mit Hilfe von gSOAP [5]

Bei der Entwicklung einer Client-Lösung kann man die Erstellung einer Definitionsdatei einsparen, da aus einer den Webservice beschreibende WSDL-Datei mittles des WSDL-Importers `wsdlcpp` eine solche Datei erzeugt werden kann. Der Entwicklungsprozess einer Client-Applikation ist in Abbildung 6.4 dargestellt [38].

6.2.2. Implementierung des Agenten

Die Implementierung des Agenten beginnt mit der Spezifizierung und Erstellung der Webservice-Funktions-Prototypen. In den vorhergehenden Schritten wurden Namensräume und Funktionen festgelegt und können zu einer Eingabedatei für den gSOAP-Compiler zusammengefügt werden. Die dazu erstellte Datei ist `devmgrSOAP.h` im Anhang B.1. Durch die Kompilierung mit *soapcpp2* werden die zur Entwicklung benötigten C-Header-Dateien erstellt, unter anderem die Datei `SOAPdevmgrStub.h`, die sämtliche SOAP-Prototypen enthält. Die Zusammenhänge der einzelnen Programmdateien sind in Abbildung 6.5, Seite 62 abgebildet. Zu beachten ist in diesem Fall, dass nicht nur die gSOAP-Dateien, die zur Erstellung eines Servers nötig sind, eingebunden werden, sondern durch den Aufruf der `signalEvent()`-Funktion in den *eventreceivern* auch die gSOAP-Client Dateien nötig sind. Die Datei `stdsoap2.c` ist eine Bibiliotheksdatei von gSOAP, die wiederum von den erzeugten Stub- und Skeleton-Dateien benötigt wird.

Bei der Definition der Webservice-Prototypen wurde bei der Entwicklung ein Schwerpunkt darauf gelegt, dass Datentypen und Funktionsparameter in der späteren XML-Serialisierung einen menschlich lesbaren, semantischen Zusammenhang ergeben. Unter anderem deswegen

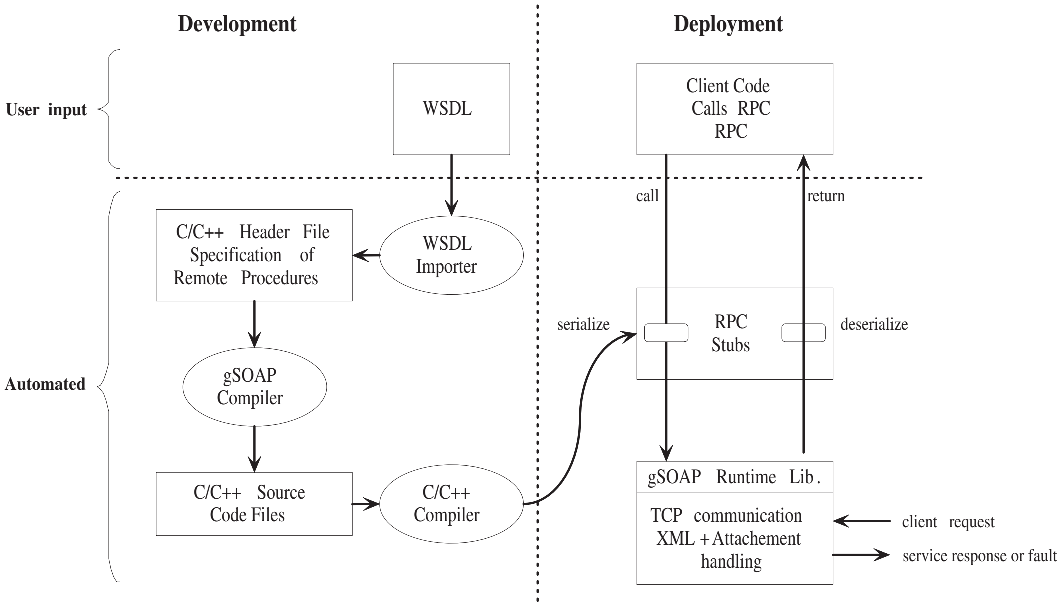

Abbildung 6.4.: Erzeugung eines Webservice-Clients mit Hilfe von gSOAP [5]

wurde von der Verwendung generischer Datentypen und Felder abgesehen und zugunsten von funktionstypischen, spezifischen Strukturen entschieden.

Ein Beispiel mit einer SOAP-Nachrichtenumsetzung ist die Modellierung eines `getSystemInformation()`-Aufrufes. Das Ergebnis eines solchen Requests und seiner Modellierung ist in den Listings 6.1, 6.2 und 6.3 abgebildet.

```
struct nssys__sysInf{
  xsd__string   description 1:1;
  xsd__duration uptime      0:1;
  xsd__duration idletime    0:1;
};
```

Listing 6.1: Ausschnitt aus `devmrgSOAP.h`

```
  <element name="sysInf" type="nssys:sysInf"/>
  <complexType name="sysInf">
   <sequence>
    <element name="description" type="xsd:string" minOccurs="1" maxOccurs="1"
        nillable="true"/>
```

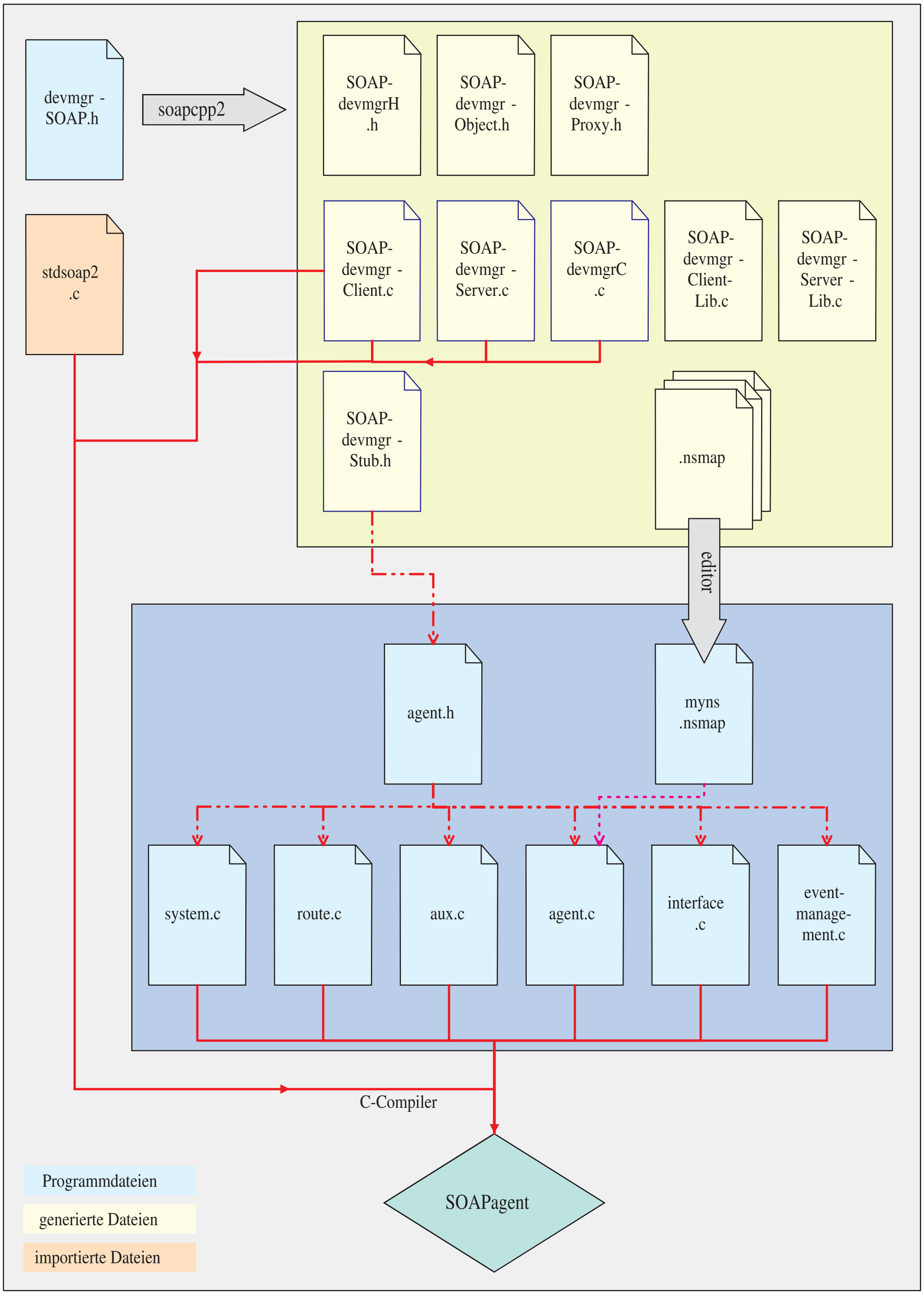

Abbildung 6.5.: Zusammenhänge der Programmdateien des Agenten

```
    <element name="uptime" type="xsd:duration" minOccurs="0" maxOccurs="1"
        nillable="true"/>
    <element name="idletime" type="xsd:duration" minOccurs="0" maxOccurs="1"
        nillable="true"/>
  </sequence>
</complexType>
```

Listing 6.2: Ausschnitt aus der Webservice Definitionsdatei `nssys.wsdl`

```
        <description xsi:type="xsd:string">Linux 2.4.20</description>
        <uptime xsi:type="xsd:duration">PT6H30M43S</uptime>
        <idletime xsi:type="xsd:duration">PT4H44M41S</idletime>
```

Listing 6.3: SOAP-Ausschnitt eines getSytemInformation Aufrufes

Da der Agent wie im Organisationsmodell beschrieben zwei verschiedene Kommunikationsbeziehungen unterhält, werden zuerst die Managementfunktionen für die Schnittstellen erläutert, im Anschluss daran die Funktionen der Ereignisbehandlung. Nach der vollständigen Funktionsbeschreibung findet eine Erläuterung der Abläufe im Agenten statt.

6.2.2.1. Die Webservice-Funktionen und Datentypen des Agenten

Die Webservice-Funktionen bestehen bis auf einen Fall in reinen atomaren Funktionen mit Ein- und Ausgabeparametern. Auf den einzigen komplexeren Fall, die Ereignisbehandlung, wird im Anschluss genauer eingegangen. Eine Beschreibung der Datenstrukturen und Funktionen ist in den folgenden beiden Aufzählungen gegeben:

Die Datenstrukuren:

WS_mgmt_stdDef:address Ist ein Datentyp, der als Attribut den Typ der enthaltenen Adresse besitzt. Die Adresse wird laut XML-Präferenz in menschlich lesbarer Form als Zeichenkette abgelegt. Es existieren in diesem Agenten zwei Adresstypen, zum einen die IPv4-Adresse in der üblichen Schreibweise, zum anderen die Ethernet-Adresse in der durch Doppelpunkte getrennten Hexadezimalschreibweise. Ein Beispiel für diese beiden Typen ist hier gegeben:

```
<address xsi:type="ns:address" type="IP">
<addr xsi:type="xsd:string">192.168.2.1</addr></address>

<link-address xsi:type="ns:address" type="ETH">
<addr xsi:type="xsd:string">00:50:56:c0:00:08</addr></link-address>
```

Listing 6.4: Beispiel: WS_mgmt_stdDef address-Strukturen

WS_mgmt_stdSystem:sysInf Ist eine einfache Datenstruktur, die Informationen über das System enthält, auf dem der Agent läuft. Es besteht aus einem `description`-Feld, in das Systeminformationen eingetragen werden können, und zwei Feldern, in denen die Uptime und die Idletime[1] des Systems abgelesen werden können. Die Zeiteinheiten sind nach ISO 8601 [39] und XML-Schema [40] kodiert.

```
<nssys:sysInf SOAP-ENV:encodingStyle="http://www.w3.org/2002/12/soap-
    encoding">
 <description xsi:type="xsd:string">Linux 2.4.20</description>
 <uptime xsi:type="xsd:duration">PT15H8M37S</uptime>
 <idletime xsi:type="xsd:duration">PT11H54M26S</idletime>
</nssys:sysInf>
```

Listing 6.5: Beispiel: WS_mgmt_stdSystem sysInf-Struktur

WS_mgmt_stdInterface:interfaceDescription Die `interfaceDescription`-Struktur enthält Parameter zum Beschreiben der Eigenschaften einer Schnittstelle. Genauere Erläuterungen zu den einzelnen Einträgen in dem Feld sind in `devmgrSOAP.h`, S.95 direkt kommentiert.

```
<item xsi:type="nsif:interfaceDescription">
 <type xsi:type="xsd:int">IF-ETHERNET</type>
 <name xsi:type="xsd:string">eth0</name>
 <mtu xsi:type="xsd:int">1500</mtu>
 <flag xsi:type="xsd:string">UP</flag>
 <flag xsi:type="xsd:string">BROADCAST</flag>
 <flag xsi:type="xsd:string">MULTICAST</flag>
 <address xsi:type="ns:address" type="IP">
  <addr xsi:type="xsd:string">192.168.0.1</addr>
 </address>
 <netmask xsi:type="ns:address" type="IP">
  <addr xsi:type="xsd:string">255.255.255.0</addr>
 </netmask>
 <broadcast xsi:type="ns:address" type="IP">
  <addr xsi:type="xsd:string">192.168.0.255</addr>
 </broadcast>
 <dst-address xsi:type="ns:address" xsi:nil="true"/>
 <link-address xsi:type="ns:address" type="ETH">
  <addr xsi:type="xsd:string">00:e0:7d:c9:ac:48</addr>
 </link-address>
</item>
```

Listing 6.6: Beispiel: WS_mgmt_stdInterface interfaceDescription-Struktur

[1] Die Zeit, in der der Prozessor im Idle-Status seit Systemstart war.

WS_mgmt_stdInterface:interfaceStats Die `interfaceStats`-Struktur enthält Parameter zum Beschreiben des aktuellen Schnittstellenstatistiken. Genauere Erläuterungen zu den einzelnen Einträgen in dem Feld sind in `devmgrSOAP.h`, S.95 direkt kommentiert.

```
<item xsi:type="nsif:interfaceStats">
 <ifname xsi:type="xsd:string">lo</ifname>
 <rx-bytes xsi:type="xsd:int">43343146</rx-bytes>
 <rx-packets xsi:type="xsd:int">424651</rx-packets>
 <rx-errors xsi:type="xsd:int">0</rx-errors>
 <rx-drop xsi:type="xsd:int">0</rx-drop>
 <rx-fifo xsi:type="xsd:int">0</rx-fifo>
 <rx-frame xsi:type="xsd:int">0</rx-frame>
 <rx-compressed xsi:type="xsd:int">0</rx-compressed>
 <rx-multicast xsi:type="xsd:int">0</rx-multicast>
 <tx-bytes xsi:type="xsd:int">43343146</tx-bytes>
 <tx-packets xsi:type="xsd:int">424651</tx-packets>
 <tx-errors xsi:type="xsd:int">0</tx-errors>
 <tx-drop xsi:type="xsd:int">0</tx-drop>
 <tx-fifo xsi:type="xsd:int">0</tx-fifo>
 <tx-colls xsi:type="xsd:int">0</tx-colls>
 <tx-carrier xsi:type="xsd:int">0</tx-carrier>
 <tx-compressed xsi:type="xsd:int">0</tx-compressed>
</item>
```

Listing 6.7: Beispiel: Ausschnitt aus einer WS_mgmt_stdInterface interfaceStats-Struktur

WS_mgmt_stdRoute:routingInformation Diese Datenstruktur dient der Übermittlung der Einträge aus der Routingtabelle. Genauere Erläuterungen zu den einzelnen Einträgen in dem Feld sind in `devmgrSOAP.h`, S.95 direkt kommentiert.

```
<item xsi:type="nsrt:routingInformation">
 <iface xsi:type="xsd:string">eth0</iface>
 <dst-addr xsi:type="ns:address" type="IP">
  <addr xsi:type="xsd:string">169.254.0.0</addr>
 </dst-addr>
 <gw-addr xsi:type="ns:address" type="IP">
  <addr xsi:type="xsd:string">0.0.0.0</addr>
 </gw-addr>
 <nm-addr xsi:type="ns:address" type="IP">
  <addr xsi:type="xsd:string">255.255.0.0</addr>
 </nm-addr>
 <flag xsi:type="xsd:string">UP</flag>
 <mtu xsi:type="xsd:int">40</mtu>
 <metric xsi:type="xsd:int">0</metric>
```

```
  <refcnt xsi:type="xsd:int">0</refcnt>
  <use xsi:type="xsd:int">0</use>
  <window xsi:type="xsd:int">0</window>
  <irtt xsi:type="xsd:int">0</irtt>
</item>
```

Listing 6.8: Beispiel: Ausschnitt aus einer WS_mgmt_stdRoute routingInformation-Struktur

Die Webservice-Funktionen[2]:

WS_mgmt_stdSystem:getSystemInformation() Diese Funktion hat keine Eingabeparameter und liefert eine `sysInf`-Struktur mit der aktuellen Uptime und Idletime.

WS_mgmt_stdSystem:uptime() Diese Funktion liefert die Uptime als Zeichenkette.

WS_mgmt_stdInterface:getInterfaceInformation() Diese Funktion bietet eine Möglichkeit, Schnittstelleninformationen zu empfangen. Bleibt die Eingabestruktur `in` leer, so werden alle vorhandenen Schnittstelleneinträge zurückgegeben. Falls in der Eingabestruktur Angaben gemacht werden, so werden nur mit diesen Angaben übereinstimmende Schnittstelleneinträge zurückgegeben. Im vorliegenden Prototypen wird nur der Schnittstellenname zur Selektion unterstützt.

WS_mgmt_stdInterface:setInterfaceInformation() Diese Funktion bietet eine Möglichkeit, die Schnittstelleninformationen zu modifizieren. Im Parameter `in`, werden die zu setzenden Eigenschaften der Schnittstelle, die über den Namen identifiziert wird, übertragen. Das Ergebnis der Funktion ist eine `interfacelist`-Struktur, die mit den gesetzten Werten zur Bestätigung gefüllt ist. Falls während der Bearbeitung ein Fehler auftritt, so wird ein SOAP-FAULT mit Details zur Fehlerursache zurückgegeben.

WS_mgmt_stdInterface:getInterfaceStats() Diese Funktion gibt die angeforderten Schnittstellenstatus eines Gerätes anhand eines `interfaceStats`-Feldes zurück. Die Funktion hat einen Eingabeparameter in Form einer Zeichenkette. Falls diese Zeichenkette keine Zeichen (also Länge Null) enthält, so werden die Informationen von allen auf dem Gerät vorhandenen Schnittstellen zurückgegeben. Anderenfalls stellt der Eingangsparameter `ifname` einen Schnittstellennamen einer Schnittstelle dar, deren Informationen dann zurückgegeben werden. Falls keine Statusinformationen gefunden werden, so wird ein Feld der Länge Null zurückgegeben.

WS_mgmt_stdRoute:getRoutingEntrys() Funktioniert wie die Interfaceroutine `getinterfaceInformation()`, nur dass in diesem Fall die Einträge der Routingtabelle zurückgegeben werden.

Der Normalfall der Kommunikation zwischen Manager und Agent ist ein einfaches Request-Response-Verfahren, wie in Abbildung 6.6 (S. 67 linke Seite) zu sehen ist.

[2]für eine genauere Beschreibung der Funktionsprototypen und Parameter ist in Anhang B.1, Seite 95, die Datei `devmgerSOAP.h`

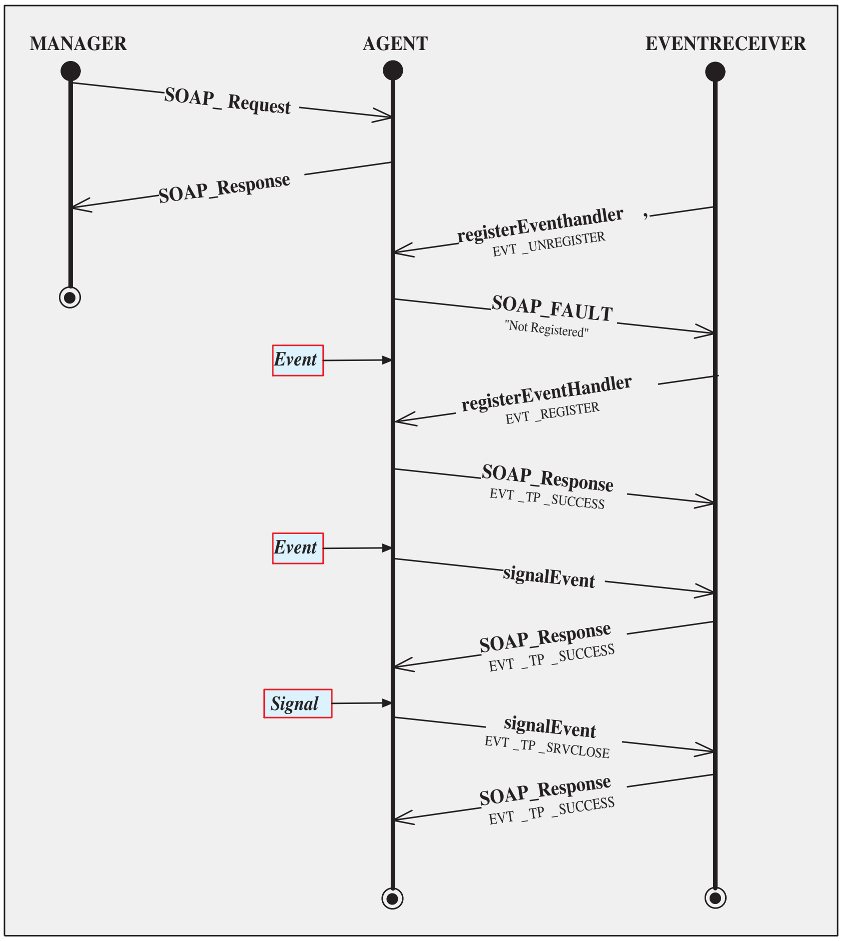

Abbildung 6.6.: Kommunikationsdiagramm der drei Managementkomponenten

6.2.2.2. Die Webservice-Funktionen und Datentypen der Ereignisbehandlung

Die Besonderheit in der Funktion der Ereignisbehandlung liegt, wie schon angesprochen, in der Kommunikationsform. Es sind beide Partner sowohl Webservice-Dienstanbieter als auch Dienstnutzer. Somit liegt eine Peer-to-Peer-Beziehung vor, in der beide Partner Funktionen bereitstellen.

Die Ereignisbehandlung erfordert für die eindeutige Identifikation der Teilnehmer in der SOAP-Nachricht einen Header-Eintrag, in dem eine vom Agenten beim Registrieren vergebene Identifikationsnummer vermerkt ist. Dazu ist das SOAP-Header-Element `connection-ID` in der zentralen gSOAP-Definitionsdatei `devmgrSOAP.h` und der zum Eventreceiver gehörenden Definitionsdatei-Datei `mynsevt.h` als `SOAP_ENV__Header` definiert worden.

Aus dieser Definition wird zur Laufzeit folgender Header-Block erzeugt:

```
<SOAP—ENV:Header>
 <connection—ID xsi:type="xsd:int">123</connection—ID>
</SOAP—ENV:Header>
```

Listing 6.9: Header-Block mit connection-ID Element

Die Ereignisbehandlung besteht aus folgenden Funktionen und Strukturen:

WS_mgmt_stdEvent:registerEventHandler() Diese Funktion wird vom Agenten bereitgestellt und dient den Ereignisempfängern dazu, sich beim Agenten zum Empfang dieser Meldungen anzumelden und den Empfangsstatus zu verändern. Bei der ersten Anmeldung wird in der Rückantwort ein Header-Block mitgeschickt, in dem eine Identifikationsnummer für dieses Kommunikationsverhältnis vorgemerkt ist. Dieser Header mit der Nummer muss bei den folgenden Operationen immer bis zur Abmeldung als Identifikation mitgeschickt werden. Da über diese Funktion sowohl An- als auch Abmeldung realisiert werden, ist dieser Mechanismus als Sicherheit gedacht, um unabsichtliche Abmeldungen durch falsche Parameter von anderen Kommunikationsverbindungen des Agenten zu vermeiden.

Die Parameter des Dienstes bestehen aus Hostname des Eventreceivers, Portnummer des Eventreceivers und den beiden Feldern zu den Registrierungsparametern Aktion (register, unregister), `enum regAction`, und Typ, `enum regType`, der Registrierung. Mit dem Typ der Aktion kann man festlegen, welche Arten von Meldungen durch die Aktion erfasst werden sollen. So kann neben dem Empfang aller Ereignisse (`EVT_REGTYPE_ALL`) auch der Empfang einzelner Teilbereiche, die der Agent überwacht, wie zum Beispiel die Routingereignisse (`EVT_REGTYPE_RT`) oder die Schnittstellenereignisse (`EVT_REGTYPE_IF`), ausgewählt werden. Der Rückgabeparameter der Funktion gibt an, ob die Registrierungsoperation erfolgreich war oder nicht.

Bei fehlerhaften Parametern wird von dieser Funktion eine SOAP-FAULT-Nachricht zurückgesendet.

WS_mgmt_stdEvent:signalEvent() Diese Funktion wird vom Empfänger, bereitgestellt. Sie dient dem Agenten dazu, die aufgetretenen Ereignisse in Textform den Empfängern mitzuteilen. Dazu wird mit `error` die Fehlerbeschreibung und mit `msgtype` die Art des Fehlers übergeben. Der Rückgabeparameter dient als reine Bestätigung, ob die Nachricht angekommen ist.

Der Ablauf einer solchen Kommunikation zwischen Agent und Eventreceiver ist in Abbildung 6.6 auf der rechten Seite zu sehen. Der erste, vom Eventreceiver ausgelöste Request stellt eine fehlerhafte Anforderung dar, da als Aktion ein Abmeldevorgang ohne vorherige Anmeldung initiiert wurde. Daher antwortet der Agent mit einer SOAP-Fehlermeldung. Die nachfolgende Ereignisauslösung bewirkt kein Versenden von Nachrichten, da noch kein Empfänger solcher Meldungen beim Agenten eingetragen ist.

Um als Eventreceiver in eine Kommunikationsbeziehung mit dem Agenten zu treten, muss er sich zuerst, wie im nächsten Schritt angedeutet, beim Agenten als Empfänger solcher Nachrichten registrieren. Neben dem Erfolg dieser Aktion bekommt der Eventreceiver eine VerbindungsID zugewiesen, die für die nächsten Operationen wichtig ist. Diese Identifikationsnummer ist wichtig, damit nicht ein anderer Benutzer des Webservices durch eine unachtsame Angabe der Parameter die Verbindungparameter verändert oder die Verbindung ganz beendet. Vom Zeitpunkt der Registrierung an können Veränderungen nur noch vorgenommen werden, wenn das Tripel ID, Hostname, Portnummer mit dem Eintrag auf dem Agenten übereinstimmen.

Das nächste Ereignis bewirkt nun, dass alle an dieser Ereignisart interessierten Empfänger eine Nachricht geschickt bekommen.

In den letzten beiden Schritten ist angedeutet, was passiert, wenn der Agent ein Signal zum Beenden, z.B. durch das Systemsignal `SIGKILL`, bekommt. Daraufhin verschickt er an alle Eventreceiver eine Nachricht vom Typ `EVT_TP_SRVCLOSE`, was den Empfängern signalisiert, dass die Kommunikationsbeziehung beendet wird. Diese quittieren dann den Empfang und der Agent beendet sich.

6.2.2.3. Aufbau und Funktionsweise des Agenten

Der Agent besteht aus der Headerdatei `agent.h` und den Programmdateien `agent.c`, `interface.c`, `route.c`, `eventmanagement.c`, `system.c` und `aux.c`. Um das Hauptaugenmerk bei der Entwicklung auf die SOAP-Funktionsprototypen zu legen, werden zur Schnittstellenansteuerung Funktionen der *libdnet*[3] benutzt. Diese Bibliothek kapselt die Systemfunktionen zum Setzen und Empfangen von Schnittstelleninformationen in einer API.

Die zentrale Steuerung des Agenten findet innerhalb einer Schleife in der `main()`-Funktion in `agent.c` statt. In dieser Datei befinden sich auch die Aufrufe der Webservice-Funktionen durch die Funktionen `nssys__uptime`, `nssys__getSystemInformation`, `nsif__getInterfaceInformation`, `nsif__setInterfaceInformation`, `nsif__getInterfaceStats`, `nsrt__getRoutingEntrys` und `nsevt__registerEventHandler`.

Die Webservice-Funktionen ihrerseits bedienen sich Funktionen, die in den jeweiligen Programmdateien zum Bearbeiten der Anforderungen abgelegt sind[4].

Der Programmablauf des Agenten ist schematisch in Abbildung 6.7 dargestellt. Nach der erfolgreichen Auswertung der Kommandozeilenparameter werden die globalen Datenstrukturen für die Ereigniskontrolle innerhalb der `init()`-Funktion initialisiert.

Nach der Initialisierung der Datenstrukturen findet die Initialisierung des SOAP-Sockets durch `soap_init()` und `soap_bind()` statt.

Nach der erfolgreichen Initialisierungsphase tritt das Programm in eine Endlosschleife ein, die durch die Signale *SIGINT*, *SIGHUP*, *SIGKILL* beendet werden kann. Dies führt zur kontrollierten Schließung der Kommunikationsverbindungen und zur Beendigung des Agenten. Beim Beenden des Agenten wird dieses Ereignis allen registrierten Eventreceivern durch eine `signalEvent()`-Meldung des Typs `EVT_TP_SRVCLOSE` mitgeteilt. Der Empfang sollte dann von den Eventreceivern bestätigt werden.

[3] http://libdnet.sourceforge.net

[4] Die Hilfsfunktionen zum Namensraum `WS_std_Interface` sind zum Beispiel in `interface.c` enthalten.

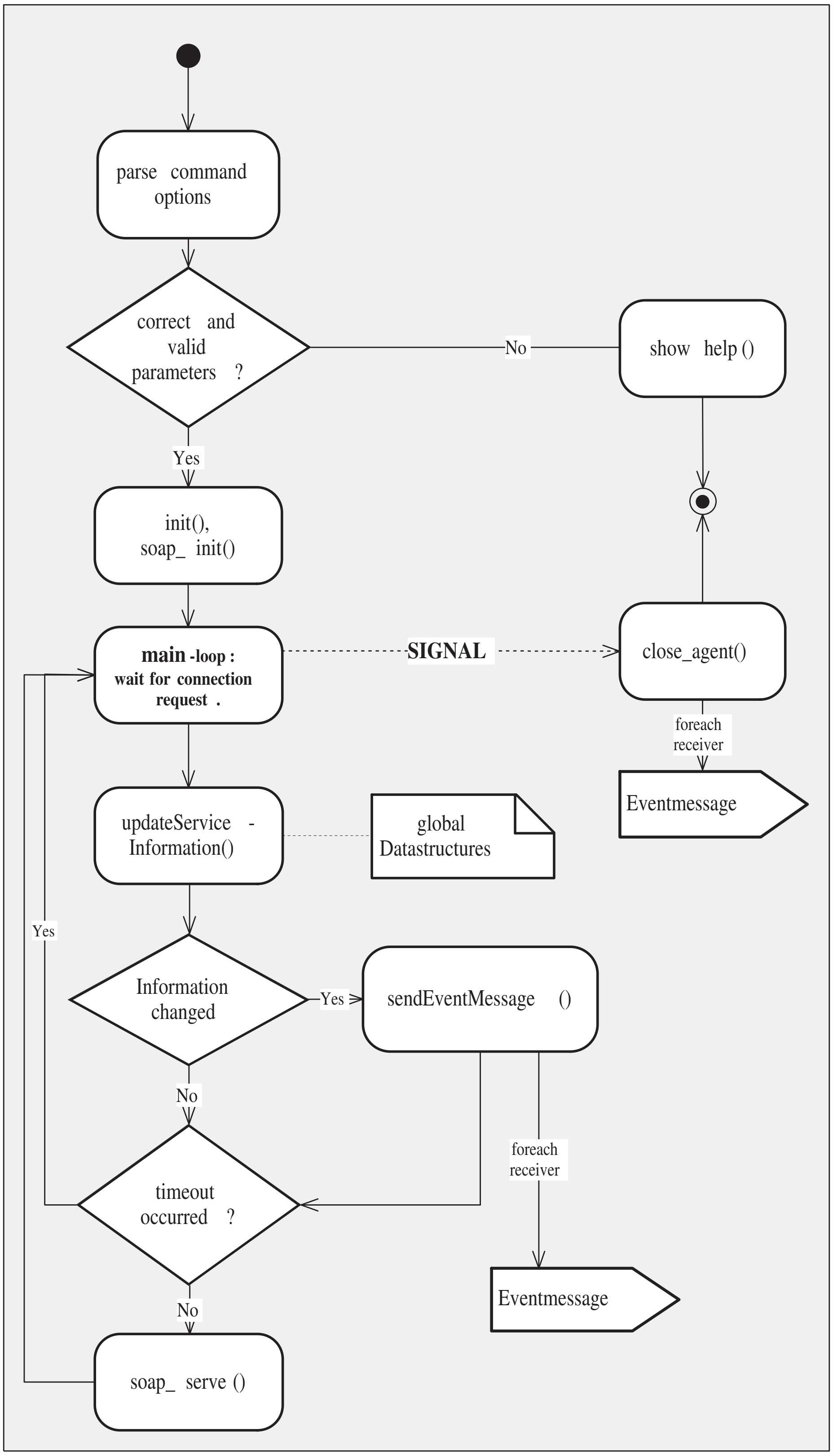

Abbildung 6.7.: Hauptprogrammablauf des Agenten

In der Schleife wird bis zum Ablauf eines Zeitlimits auf eingehende Dienstanforderungen gewartet. Nach Ablauf des Zeitlimits oder bei eingehender Anforderung findet eine Aktualisierung der globalen Datenstrukturen statt und falls eine Anforderung eingegangen ist, wird diese durch `soap_serve()` bedient. Diese Funktion der gSOAP-API ruft wiederum die Webservice-Funktion auf, die im Request angefordert wurde, und übernimmt die Übermittlung und Serialisierung des Ereignisses der Funktion. In Abbildung 6.8 ist der Verlauf eines Requests genauer dargestellt.

Für die Erkennung von Konfigurationsveränderungen des Gerätes existieren drei globale Felder[5]. Bei jedem Aufruf der `update`-Funktionen werden diese Datenstrukturen an die aktuellen Verhältnisse angepasst und die Veränderungen als Ereignis an registrierte Eventreceiver versandt.

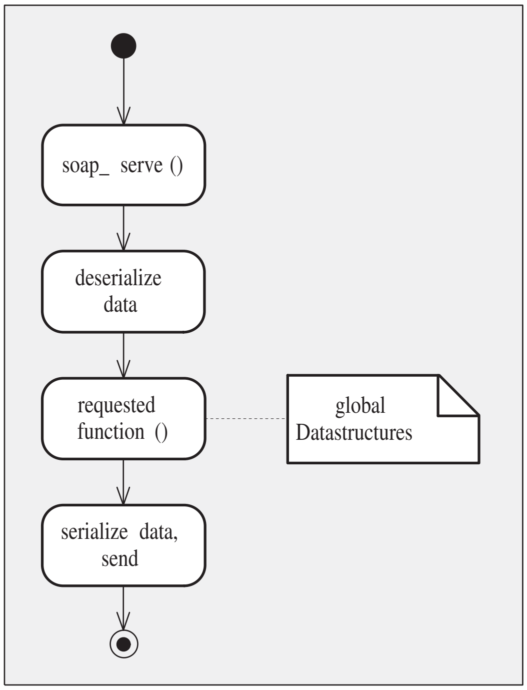

Abbildung 6.8.: Ablauf eines Requests an den Agenten

Für das Ereignismanagement ist die globale Struktur `regarr` zuständig, in der die registrierten Eventreceiver abgelegt werden.

Beim Registrieren oder Abmelden werden zuerst die angegebenen Parameter auf Validität überprüft und danach in die Struktur eingetragen. Zusätzlich wird bei einer Neueintragung eines Empfängers diesem eine neue Verbindungsidentifikation zugeteilt.

Beim Versand von Ereignismeldungen durch `sendEventMessage()` wird die übergebene Nachricht an alle Empfänger übermittelt, die sich für den Empfang einer solchen Nachricht registriert haben. Falls beim Versand der Nachrichten ein Fehler auftritt, wird der Fehlereintrag

[5] `ifDescr, ifStat und rtInf`

der Verbindung um eins erhöht. Hat eine Vebindung fünf aufeinanderfolgende Fehlversuche erzeugt, so wird die Verbindung aus dem Registrierungsfeld gelöscht.

6.2.3. Implementierung des Eventreceivers

Normalerweise würde die Implementierung des Eventmanagers so beginnen, dass man anhand der vom Agenten erzeugten `.wsdl` Beschreibungen mittels `wsdlcpp` aus dem gSOAP Packet eine Header-Datei erzeugt, mit der dann wiederum die Client-Stubs und Server-Skeletons mit `soapcpp2` gebildet werden können. Leider generiert `wsdlcpp` zur Zeit noch eine fehlerhafte Datenstruktur bei den Rückgabetypen, so dass dieser automatische Weg nicht eingeschlagen werden kann.

Die WSDL-Beschreibung in Listing 6.10 führt zur fehlerhaften Datenstruktur in Listing 6.11 und der darausfolgenden Fehlermeldung des Agenten beim Kontaktieren mit der Funktion, dargestellt in Listing 6.12. Dieser Fehler tritt bei den Datentypen des Ausgangsparameters auf. Ändert man den Datentyp wie in `devmgrSOAP.h` (vgl. Anh. B.1) auf `xsd__int`, ist die Definition wieder richtig und der Agent kann fehlerfrei kontaktiert werden.

```
<message name="registerErrorHandlerRequest">
 <part name="portnumber" type="xsd:int"/>
 <part name="hostname" type="xsd:string"/>
</message>

<message name="registerErrorHandlerResponse">
 <part name="result" type="xsd:int"/>
</message>

<portType name="ServicePortType">
 <operation name="registerErrorHandler">
  <documentation>
   Service definition of function nserr__registerErrorHandler
  </documentation>
  <input message="tns:registerErrorHandlerRequest"/>
  <output message="tns:registerErrorHandlerResponse"/>
 </operation>
</portType>
```

Listing 6.10: WSDL-Beschreibung

```
struct nserr__registerErrorHandlerResponse {
        xsd__int _result;
};

nserr__registerErrorHandler( xsd__int portnumber, xsd__string hostname, struct
    nserr__registerErrorHandlerResponse * out );
```

Listing 6.11: Erzeugte Datenstruktur durch `WSDLCPP`

```
SOAP FAULT: SOAP—ENV:Sender
"Data type '' mismatch in element 'SOAP—RPC:result'"
```

Listing 6.12: Resultierende Meldung durch den Agenten

Durch diesen Generierungsfehler kann man zwar die Headerdatei für den Stub-Compiler durch `wsdlcpp` automatisch erstellen lassen, aber die Ausgangsdatentypen müssen danach korrigiert werden. Nach der Erstellung der gSOAP-Header Datei (`mynsevt.h`) kann wie beim Agenten durch den Stub-Compiler `soapcpp2` der Stub für die SOAP-Funktionen erstellt werden.

Der Eventreceiver registriert sich nach der Kommandozeilenauswertung und Initialisierung des SOAP-Stubs zuerst beim Agenten und wartet dann in einer Endlosschleife auf eingehende `signalEvent`-Requests. Die Daten dieses Requests werden dann in der Standardausgabe ausgegeben.

Das Beenden des Programms geschieht auch hier, wie im Agenten, durch Signale oder durch die Beendigung des Agenten, mit dem der Receiver verbunden ist.

Alle diese Funktionen sind in der Datei `eventreceiver.c` enthalten.

6.3. Modellierung des Managers

Da das Hauptaugenmerk in der Entwicklung des Prototypen auf den Webservice-Funktionen lag, wurde der Manager als einfaches Kommandozeilenprogramm realisiert, ähnlich wie die "low level tools" aus Kapitel 4.3. Durch die Übernahme von vielen Funktionen, wie zum Beispiel der Validitätsprüfung der Argumente, ist die Modellierung und Implementierung des Managers relativ einfach geworden. Im hier vorliegenden Fall ist der Manager somit eine reine Übersetzungseinheit zwischen Kommandozeile und Webservice-Funktionen.

Die Funktionen des Managers beschränken sich somit auf das Auslesen der Kommandozeile, die Ausführung der Webservice-Funktionen und die Darstellung der Ergebnisse.

Als gSOAP-Header-Datei wurde die Header-Datei des Agenten benutzt (`devmgrSOAP.h`) und die Funktionen sind in der Datei `manager.c` realisiert.

6.4. Limitierungen von gSOAP

Da das gSOAP-Paket immer noch in ständiger Entwicklung ist, sind einige SOAP-typische Eigenschaften in der vorliegenden Version 2.3rev2 noch nicht modellierbar. So können nicht mehrere RPC-Aufrufe innerhalb eines SOAP-Envelopes stattfinden. Diese Möglichkeit würde es erlauben, voneinander abhängige Funktionen, wie zum Beispiel Authentifizierung der Benutzer und die Anforderung auf Modifikation von Daten innerhalb eines SOAP-Envelopes, zu bündeln.

Ein weiteres kleines Manko ist, dass es nur einen Ausgangsparameter und eine strikte Trennung von Ein- und Ausgangsparametern gibt. Der SOAP-Standard erlaubt auch Ein/Ausgabeparameter. Dadurch, dass nur ein Ausgabeparameter existiert, müssen die meisten Rückgaben als Strukturen modelliert werden, was wiederum zu einer um die Strukturdefinition vergrößerten Beschreibungsdatei führt.

Eine weitere Einschränkung zur spezifizierten SOAP-Nachrichtenmodellierung muss man bei der Verwendung von gSOAP machen. gSOAP unterstützt nur Datentypen, die vor der Kompilierung bekannt sind. Das hat zur Folge, dass die C-Konstrukte `union` und `void`-Zeiger auf unbekannte Datentypen nicht modelliert werden können.

6.5. Fazit und Ausblick

Mit der weiteren Entwicklung von gSOAP ist es eventuell möglich, mehrere Funktionsaufrufe in einem SOAP-Envelope zu transportieren. Damit sind effektive transaktionsbasierte Abläufe möglich. Mit diesem Mittel könnte man vernünftige Zugriffskontrollmechanismen wie zum Beispiel Anmelden, Funktionsaufrufe, Abmelden innerhalb einer Nachricht ermöglichen.

Das damit zusammenhängende Sicherheitskonzept von SOAP ist ebenfalls noch in der Standardisierungsphase, so dass vorhandene Modellierungen das Problem haben, keinerlei Sicherheit gegenüber Attacken zu bieten. Die in diesem Fall eingeführte Verbindungsnummer kann nur unbeabsichtigte Operationen verhindern, gegenüber absichtlichen Versuchen, das System zu kompromittieren, ist es nutzlos. Ein großes Problem in der Sicherheit der Webservices stellt im Moment noch die Klartextlesbarkeit der XML-Serialisierung der Daten dar. Dieses Problem kann man zur Zeit mit der Übertragung der Nachrichten über SSH oder SSL durch ihre Kommunikationsverschlüsselung beheben, die am meisten genutzte und in der Spezifikation zu SOAP auch einzig genauer beschriebene Protokollbindung ist zur Zeit HTTP. Da der Webserver Apache sowohl über ein Modul für gSOAP als auch über ein Modul für SSL verfügt, kann mit der Bindung an HTTP trotzdem eine sichere Kommunikation modelliert werden. Neben diesen Möglichkeiten könnte aber auch die Bindung an BEEP[6] (RFC 3081) diese Probleme lösen. Die Sicherheitsfunktionen von Webservices sind gerade noch in der Definitions- und Standardisierungsphase und werden somit in Zukunft interessant für die weitere Entwicklung.

[6]Blocks Extensible Exchange Protocol: BEEP ist ein “grundlegender Baukasten” für verbindungsorientierte Applikationsprotokolle. Aspekte wie die unabhängige Flusskontrolle mehrerer Kanäle innerhalb einer Verbindung, Authentisierung der Kommunikationspartner, Kompression, Verschlüsselung und weitere zukünftige Protokolleigenschaften können durch “BEEP Profiles” kombiniert werden. Eine Studienarbeit von Ingo Paschke am Instiut für Betriebssysteme und Rechnerverbund untersucht diese Bindung unter dem Titel “Implementierung und Evaluierung eines SOAP/BEEP Mappings”.

Mit den jetzt vorhandenen Funktionen und der Erweiterung des Funktionsumfanges auf die Fehlererkennung in den Statusinformationen und die Konfiguration der Routingtabelle ist ein effektives Management der Schnittstellen eines Gerätes möglich. Somit eignet sich dieses Hilfsmittel sehr gut zum Schnittstellenmanagement in einer Netzwerklandschaft. Weiteres Entwicklungpotential bietet die Entwicklung einer Möglichkeit, die Fehlermeldungen zu automatisieren und damit neben den menschlich lesbaren Informationen auch maschinenlesbare Informationen anzubieten. Mit diesem Entwicklungsschritt könnte man die Entwicklung von verteilten Netzwerkmanagementstrukturen beginnen und das Routing und die Vorauswertung von managementrelevanten Daten durch Intermediäre steuern.

7. Webservices und SNMP im Vergleich

In diesem Kapitel werden die mit den Webservice-Methoden gemachten Erfahrungen mit den Erfahrungen im Bezug auf die Modellierung einer Managementaufgabe in SNMP verglichen. Dazu werden zuerst die Unterschiede der beiden Architekturen aufgezeigt, um dann im Folgenden einen Vergleich der Implementierungsumstände vorzunehmen.

7.1. Vergleich der Managementarchitekturen

Die beiden Architekturen, SNMP und das Konzept des Prototypen, kann man insofern vergleichen, da SNMP einen integrierten Managementansatz darstellt und das Managementkonzept des Webserviceprotoypen ebenfalls auf eine solche Modellierung abzielt. Deswegen wird der Vergleich der beiden Ansätze systematisch anhand der in Kapitel 2.1 aufgestellten Strukturierung vorgenommen.

7.1.1. Die Organisationsmodelle

In der jetzigen Ausprägung des Webservice-Prototypen und des SNMP-Modells unterscheiden sich die Modelle kaum. Beide Modelle haben eine Agent-Manager-Beziehung zum Management der Daten und Informationen. Um Ereignisse mitzuteilen, wird eine dritte Instanz benötigt, um diese Meldungen zu empfangen. Bezieht man aber die Möglichkeiten, die Webservices bzw. SOAP bieten, in die Betrachtung ein, so kann man signifikante Unterschiede feststellen.

Bei SNMP hat der Agent eine rein informationshaltende Funktion, sämtliche Managementprozesse laufen auf den Managementstationen ab. Dies führt dazu, dass auch die komplette Logik des Managements in die Managementstationen verlagert ist. Diese Modellierung hatte den Ursprung darin, die Geräte so wenig wie möglich mit Managementaufgaben zu belasten. Dies führt aber wiederum zu dem Problem, dass gerätetypische Managementoperationen vom Hersteller nur dokumentiert werden und dann auf eine richtige Implementierung dieser Operationen in den Managern hoffen. Das hat wiederum dazu geführt, dass die einzelnen Hersteller proprietäre Managementsoftware für die Geräte erstellt haben (zum Beispiel HP Web Jetadmin für HP-Drucker). Dies widerspricht aber dem Gedanken der integrierten Managementumgebung, für die SNMP geschaffen wurde.

Der Webserviceansatz setzt dahingegen auf eine höhere Intelligenz und Vielseitigkeit des Agenten. Der Manager wird in diesem Modell zu einer Einheit, die sich auf die Bearbeitung der managementrelevanten Funktionen konzentrieren kann. Der Managemententwickler soll sich nicht auf die korrekte Art und Weise der jeweiligen Geräteimplementierungen des Informationsmodells konzentrieren, sondern vielmehr auf die komplexen Zusammenhänge, die eine Netzwerkkonfiguration bieten kann, um die darin enthaltenen Managementaufgaben zu lösen. Der Agent stellt Funktionen zum Management bereit, die der Manager benutzen kann. Die Validität und Korrektheit der Eingabedaten muss der Agent prüfen. Dies führt wiederum zu einem erhöhten Prozessaufwand des Agenten.

Natürlich kann auch ein SNMP-Agent Intelligent modelliert werden. Dies führt aber durch das Informationsmodell zu erheblich größeren Implementierungsaufwand und daraus resultierenden Prozessaufwand als bei den Webservices. Dieser höhere Prozessaufwand wiederspräche aber dem SNMP-Grundsatz zur Modellierung von Agenten, die Systeme mit sowenig Prozesslast wie möglich zu belasten.

Während SNMP weitgehend nur die Rolle des Managers, Ereignisbehandlers und des Agenten kennt, hat das Webservicemodell noch einen möglicherweise entscheidenden Interakteur mehr: den Intermediär. Dieser befindet sich zwischen den Parteien und bietet die Möglichkeit, Managementaufgaben von zum Beispiel Teilnetzen in untergeordnete Intermediäre auszulagern und so ein verteiltes Managementmodell zu modellieren. Dazu kann der Headerblock (vgl. Kapitel 5.2.4) einer SOAP-Nachricht verwendet werden. Durch die Möglichkeit, Nachrichten über bestimmte Pfade zum Empfänger zu leiten, zum Beispiel mit einer Intermediärkette, ist es möglich, dass diese Intermediäre die Fehlermeldungen eines Subnetzes akkumulieren und diese ausgewertet an den Manager weiterschicken. Somit könnte man eine Lastverteilung im Management von großen Netzwerken in Bezug auf das Management einrichten. SNMP kennt solche Verfahren nur in Bezug auf SNMP-Proxys, die allerdings nur eingeschränkt für diese Aufgaben zu nutzen sind.

7.1.2. Die Informationsmodelle

Die Informationsmodelle der beiden Architekturen unterscheiden sich grundlegend. Schon die Identifikation der Managementinformationen beruht auf völlig verschiedenen Konzepten. Während das SNMP-Modell von einer zentralen Registrierung und hierarchischen Ordnung der Managementinformationen ausgeht, sind die Webservice-Managementinformationen in Namensräume eingeteilt. Diese Namensbereiche kann man in SNMP mit einzelnen MIB-Modulen gleichsetzen.

Auch in der Beschreibung der Informationen gibt es Unterschiede. Die Organisation der Informationen bei SNMP läuft über die SMI in den MIBs. Diese klar strukturierten Informationsbeschreibungen haben aber einen Nachteil, der in der Anfangszeit von SNMP entstanden ist. Man hat sich damals entschieden, einen eingeschränkten Teil von ASN.1 als Datenbeschreibungssprache zu benutzen. Heute stoßen einige dieser fest definierten Datentypen an ihre Grenzen. So ist zum Beispiel mit der Zeit ein Problem dadurch entstanden, dass man für die Information `ifSpeed (mgmt.mib-2.2.2.1.5)`, der Schnittstellengeschwindigkeit, in der Interfacesgruppe der MIB-II als Datentyp Integer[1] verwendet hat, welcher mit einer Größe von 32bit eine Maximalbegrenzung der zu beschreibenden Geschwindigkeit darstellt. Dies führt aber bei den heute üblichen schnellen Schnittstellen dazu, dass dieses Feld überlaufen kann. Als Lösung hat man in RFC 2863 [41] vorgeschlagen, die Angabe in zwei Feldern übergreifend zu kodieren.

[1]eigentlich `Gauge/Gauge32`, ist aber ein beschränkter `INTEGER/Integer32` (Siehe Kapitel 3.2.2.1)

An dieser Stelle tritt eine für einen Standardisierungsprozess zwar wünschenswerte Eigenschaft zutage, gleichzeitig wird aber eine Anpassung von Datentypen im SNMP-Rahmenwerk verhindert. Einmal im Registrierungsbaum registrierte Managementinformationen dürfen nicht mehr geändert werden (vgl. [3], S. 24). Zudem führt die Vorgabe, dass nur komplette MIBs von Agenten implementiert werden sollten, immer wieder zu fehlerhaften Implementierungen, wenn zum Beispiel Geräte nicht den gesamten, in der MIB modellierten Funktionsumfang besitzen. Es gibt zwar die `AGENT-CAPABILTIES`- und das `Module-Compliance`-Makros, um vom Standard abweichende Implementierungen zu beschreiben, aber diese werden in den seltensten Fällen von den Herstellern mit den Geräten distributiert, noch werden diese Makros von vielen SNMP-Management-Werkzeugen unterstützt (Es müsste für jedes Gerät ein eigenes Konstrukt ladbar bzw. definierbar sein).

Bei den XML-Schema-Datentypen, die von SOAP verwendet werden, gibt es solche Grenzen nicht, auch die Einschränkung auf eine geringe Anzahl von Datentypen, wie im Fall von SNMP auf vier primitive Datentypen, gibt es im XML-Schema nicht. Somit ist dieses Format im Bezug auf die langfristige Modellierung von Datentypen wesentlich flexibler.

Informationen im Webservice-Rahmenwerk werden durch die Namensbereiche eindeutig referenziert. Durch die Modellierung der Informationen in Datenstrukturen ergibt sich ein wesentlich höherer Zusammenhang von abhängigen Informationen untereinander. Die Beschreibung der Informationen an sich ist mit den MIBs allerdings wesentlich datenorientierter als mit den WSDL-Beschreibungen der SOAP-Services. Hegering u.a. ([1], S. 111) kritisieren aber in diesem Zusammenhang mit dem Objektidentifizierungsbaum eine mangelnde Strukturierung der Daten, fehlende Verfeinerungsmöglichkeiten, fehlende Wiederverwendbarkeit von Objektdefinitionen und durch die zahlreichen MIB-Definitionen im Internetbereich eine Verteilung der Ressourcen auf getrennte Bereiche in Teilbäumen.

Während in den MIBs eine reine Beschreibung der modellierten Daten stattfindet, stellt die WSDL-Beschreibung eines Dienstes die vollständige Beschreibung auch aller Kommunikationsparameter dar. Da das Webservice-Modell funktionsorientiert modelliert werden muss, muss in den WSDL-Dateien auch beschrieben sein, wie man auf diese Managementinformationen zugreifen kann. Dies ist bei SNMP standardisiert geregelt durch die Protokolloperationen. Diese haben zwar den Vorteil, einen einfachen standardisierten Zugriff auf die Managementinformationen zu bieten, allerdings stößt dieses Konzept an seine Grenzen, wenn kompliziertere Zusammenhänge modelliert werden sollen. Ein Beispiel für einen solchen Fall stellen Tabellen und der Zugriff auf einzelne Elemente der Tabelle dar. Dies kann in Webservices über Zugriffsfunktionen wesentlich einfacher geregelt werden.

Ein weiterer Nachteil von SNMP ist die fehlende Möglichkeit, Aktionen zu modellieren (vgl. [2], S. 187). Zwar kann man Aktionen auf Geräten modellieren, indem man Trigger-Objekte anlegt, bei deren Setzung Aktionen ausgelöst werden, aber eine vernünftige parametrisierbare Möglichkeit bietet SNMP in diesem Fall nicht. Bei Verwendung von SOAP ist die Verwendung von Funktionen unerlässlich und Bestandteil der Modellierung. Damit stellt SOAP mit der funktionsbasierten Modellierung ein wesentlich mächtigeres Werkzeug zur Lösung vom Managementprozessen dar.

Wenn man das Informationsmodell der beiden Ansätze betrachtet, darf man die Behandlung von Ereignissen nicht vergessen. Dies wird in SNMP durch die so genannten Traps gelöst. Auch hier gibt es wieder ein paar wenige vordefinierte Standard-Traps. Wenn ein anderes Ereignis ausgelöst wird, so ist es nur über eine OID und die zugeordnete Fehlernummer möglich, die Ursache herauszufinden. Dies stellt eine grundlegende Schwäche im SNMP Rahmenwerk dar (vgl [3], S. 121). In dem hier vorgestellten Prototypen ist zwar keine maschinenlesbare Version der Ereignismeldung implementiert, aber es ist ohne Probleme möglich, bei schwerwiegenden Problemen komplexe Datenstrukturen, die den Fehler beschreiben, zu übermitteln.

7.1.3. Die Kommunikationsmodelle

In diesem Punkt unterscheiden sich beide Systeme zwar vordergründig wenig, in der Konsequenz der sich ergebenden Modellierungsmöglichkeiten lässt sich aber auch hier ein großer Unterschied feststellen. Das Kommunikationsmodell von SNMP ist ausgelegt auf den Zugriff auf die Managementinformationen durch die OIDs. Durch die Modellierung der Managementinformationen in Variablen ist es möglich, mit einem standardisierten Set von Zugriffsfunktionen zu arbeiten (Kap. 3.2.3). Die Übertragung der Daten findet durch die BER kodiert in binärer maschinenlesbarer Form statt.

Durch die Übertragung der Daten in Form von XML-Dokumenten im Webservice-Rahmenwerk in Verbindung mit SOAP ist die reine übertragene Datenmenge wesentlich größer. Durch die Serialisierung und Deserialisierung von XML-Nachrichten ist der Prozessaufwand gegenüber SNMP auch in diesem Teil der Bearbeitung eines Managementrequests höher. Gerade in diesem Bereich ist es wichtig auf effiziente Algorithmen zurückzugreifen, wie sie Chiu, Govindaraju und Bramley in [42] beschreiben. In diesem Zusammenhang beschreiben sie auch, dass man den Speicherbedarf während der Serialisierung nicht vernachlässigen darf, der vier bis zehnmal höher sein kann als der maschinentypische Bedarf.

Auch ist es durch die Modellierung der Daten in Datenstrukturen in den Namensräumen nur umständlich möglich, gemeinsame Zugriffsfunktionen auf Daten verschiedener Namensräume zu modellieren. Dies stellt ein entscheidendes Manko dar, wenn generische Werkzeuge zum Management benutzt werden sollen, die einheitliche Zugriffsfunktionen auf die Managementinformationen benötigen, um möglichst viele Geräte zu unterstützen.

Die Bindung von SNMP an UDP als verbindungslosen Datagrammdienst verursacht gerade im Bereich der Ereignissignalisierung mit Traps Probleme in einem stark überlasteten Netz, wenn Pakete verloren gehen. So kann es sein, dass die Überlastungsmeldung nie ankommt. Der Prototyp ist deswegen gerade in diesem Bereich in Form eines bestätigten Dienstes modelliert, um diesen Fall zu umgehen.

In der Spezifikation zu SOAP ist zwar die Bindung an HTTP beschrieben, allerdings ausdrücklich nur als Beispiel. Es kann jedes beliebige Übertragungsprotokoll zur Übertragung des SOAP-Envelopes benutzt werden.

Den größten Unterschied in den Modellierungsmöglichkeiten bietet aber der schon beschriebene Intermediär. Da es eine solche Instanz in SNMP nicht gibt, bieten Webservices gerade im Bereich der zukünftigen Entwicklung von verteilten Managementumgebungen ein weitaus flexibleres und größeres Potential.

7.1.4. Die Funktionsmodelle

Ein Vergleich der Funktionsmodelle ist zwar durchführbar, erbringt aber für die Fragestellung dieser Arbeit keine weiterbringenden Ergebnisse, da beide Konzepte alle der in Kapitel 2.1 beschriebenen Teilbereiche ausfüllen können. Es sind somit nur implementierungsabhängige Unterschiede festzustellen, die im Rahmen dieser Bewertung nicht von Belang sind, da die Implementierung des Webservice-Ansatzes nur eine prototypische Implementierung darstellt und somit kein Vergleich zu den jahrelang entwickelten MIBs ist.

7.2. Vergleich der Implementierungsmöglichkeiten

Beim Vergleich der Implementierung eines Managementproblems in beiden Konzepten fallen die ersten Unterschiede bei der Modellierung der Managementdaten nach Vorgabe des Informationsmodells auf. Die Analysephase ist in beiden Modellen gleich. Durch den grundlegenden Unterschied der Informations- und Kommunikationsmodelle (s.o.) ist der größte Unterschied in der Modellierungsphase zu finden.

Die reale Implementierung ist mit SOAP und den aktuellen Hilfsmitteln einfacher durchzuführen, da es wesentlich mehr und differenziertere Hilfsmittel zur Entwicklung von SOAP-Webservices gibt als zur Entwicklung von SNMP-Lösungen. SOAP hat in den meisten Programmiersprachen zumindest durch Prekompiler schon Einzug gehalten. Die einzigen freien Möglichkeiten, SNMP-basierte Lösungen zu entwickeln, sind *scotty* und *net-snmp*. Ersteres ist allerdings ein Hilfsmittel, um nur auf der Managerseite zu Lösungen zu gelangen, während man bei *net-snmp* den Agenten erweitern und Managementapplikationen entwickeln kann. Zudem sind beide ausschließlich an die Programmiersprache C/C++ gebunden. Freie Hilfsmittel, um “standalone”-Lösungen wie mit gSOAP zu entwickeln, waren in Literatur und auf den einschlägigen Webseiten zu SNMP nur vereinzelt zu finden.

Im weiteren Implementierungsverlauf ist beim Webservice-Modell durch die erhöhten Anforderungen an den Agenten festzustellen, dass hier wesentlich mehr Arbeitsaufwand notwendig ist als beim Manager. Da dies im Fall von SNMP aber ausschließlich in den Manager verlagert wird, ist der Nettoaufwand der Implementierung, wenn man an die Entwicklung von Manager und Agent als Komplettlösung denkt, in beiden Fällen gleich. Bei der Entwicklung nur eines Teils fallen die Unterschiede dagegen deutlich ins Gewicht.

7.3. Vergleich der Leistungsverhalten

Um die Leistung des Protoypen gegenüber einer SNMP-Implementierung zu testen, wurde ein Leistungstest durchgeführt. Die Bewertung des Vergleichs der Leistungsverhalten von verschiedenen Programmen gestaltet sich problematisch, wenn es wie in diesem Fall um zwei Implementierungen mit grundsätzlich verschiedenem Umfang geht. Zum einen um den Prototypen, der die grundlegenden Funktionen zum Management von Schnittstellen implementiert, zum anderen um den hochentwickelten Agenten und die dazugehörige API der SNMP Lösung `net-snmp`[2].

[2] `net-snmp` ist aus cmu-snmp entstanden und stellt eine freie Entwicklungs-Api mit dazughörigem Agenten und Kommandozeilenprogrammen dar (*http://net-snmp.sourceforge.net/*).

Die Testumgebung bestand aus einem Testnetzwerk mit zwei Rechnern, einem Athlon 1500XP (Obelix) und einem Pentium 133 (Idefix), die über eine 100MBit Ethernet-Verbindung vernetzt sind. Gerade die Verwendung des Pentium-Rechners erwies sich im Test als besonders sinnvoll, da gerade hier Skalierbarkeitsunterschiede der Implementierungen bemerkbar wurden. Beide Rechner laufen unter dem Betriebssystem Linux.

Es wurde das Leistungsverhalten der Kommunikations- und Vearbeitungssysteme durchgeführt. Aus diesem Grund wurde während des Tests auf Ausgaben der Anfrage-Ergebnisse verzichtet. Der Basistest bestand aus der Messung der Dauer eines Requests an den jeweiligen Agenten. Um die Skalierbarkeit zu testen, wurde diese Messung mit verschiedenen Request-Anzahlen durchgeführt.

Im Folgenden werden die verschiedenen Einzeltests erläutert:

WS-InterfaceInf Dieser Test besteht aus dem Aufruf der Webservice-Funktion `getInterfaceInformation()`

WS-InterfaceStats Dieser Test besteht aus dem Aufruf der Webservice-Funktion `getInterfaceStats()`

Ws-Routing Dieser Test besteht aus dem Aufruf der Webservice-Funktion `getRoutingInformation()`

SNMP-Get Dieser Test empfängt zuerst mit einem GET-Request die Anzahl n der Zeilen der `mib-2.interfaces.ifTable` und dann mit n aufeinderfolgenden GET-Requests die Elemente `ifType, ifDescr, ifMtu, ifOperStatus, ifAdminStatus, ifPhysAddress`.

SNMP-GetBulk Dieser Test empfängt zuerst mit einem GET-Request die Anzahl n der Zeilen der `mib-2.interfaces.ifTable` und dann mit einem GETBULK-Request den Rest der gesamten Tabelle.

SNMP-GetInterface Mit diesem Kommando sollen die gleichen Informationen wie bei der Webservice-Funktion `interfaceHandling()` empfangen werden. Dazu werden zuerst die benötigten Informationen mit einem GET/GETBULK-Request wie in SNMP-Get angefordert, danach wird mit einem GETNEXT-Request über die erste Spalte der `ipAddrTable` aus der `ip`-Gruppe der MIB-II die Zeilenanzahl bestimmt, um im letzten Request die erforderlichen Informationen der Tabelle mit einem GETBULK-Request zu empfangen.

SNMP-GetRoute Mit diesem Kommando sollen die gleichen Informationen wie bei der Webservice-Funktion `routeHandling()` empfangen werden. Dazu wird mit einem GETNEXT-Request über die erste Spalte der `ipRouteTabel` aus der `ip`-Gruppe der MIB-II die Zeilenanzahl bestimmt, um im letzten Request die erforderlichen Informationen der Tabelle mit einem GETBULK-Request zu empfangen.

Beim Vergleich der beiden Implementierungen auf lokaler Ebene konnte kein signifikanter Unterschied festgestellt werden, da die Abarbeitungszeiten nahezu identisch sind. Wenn man

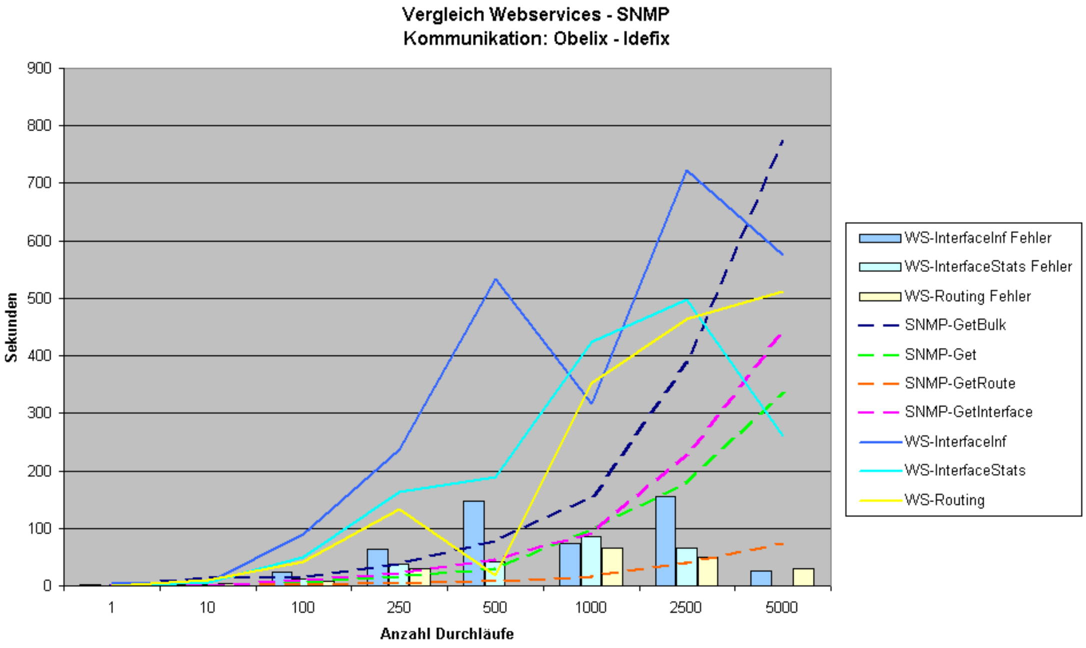

Abbildung 7.1.: Vergleich der beiden Implementierungen lokal auf dem Athlon

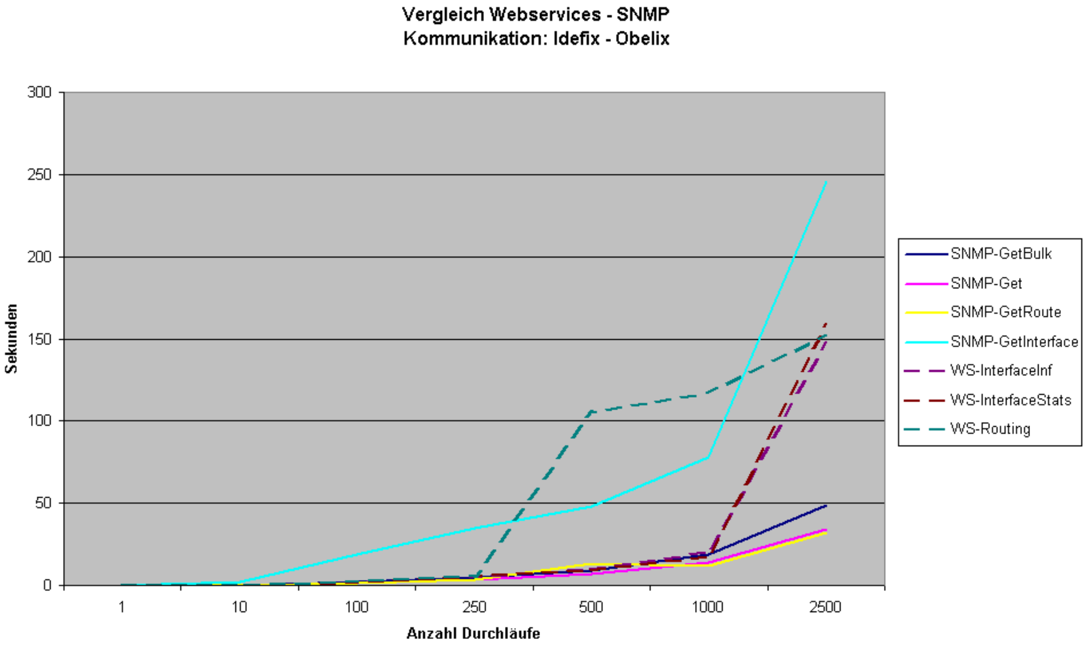

Abbildung 7.2.: Vergleich der beiden Implementierungen, der Pentium als Manager, der Athlon als Agent

aber den Vergleich des Tests über das Netzwerk zu einem anderen Kommunikationspartner betrachtet, fällt zunächst auf, dass der erwartete Performancevorteil von SNMP nicht merkbar auftritt. Dies kann man dadurch erklären, dass der net-SNMP-Agent wesentlich mehr Funktionalitäten implementiert und bereitstellt als der Webservice-Prototyp. Man kann aber im Bezug auf die Skalierbarkeit feststellen, dass die net-SNMP-Implementierung in diesem Bereich wesentlich besser arbeitet, als der Webservice-Prototyp, wie in Abbildung 7.2 zu sehen ist. Bemerkbar macht sich dies durch die hohen Antwortzeiten des Webservices und die angefallenen Fehler während des Tests. Abschließend muss man feststellen, dass eine direkte Vergleichbarkeit und Zuordnung von Unterschieden durch die verschiedenen Entwicklungsstufen der Implementierungen in diesem Rahmen nicht gegeben ist.

7.4. Zusammenfassung der Ergebnisse

In Tabelle 7.1 sind die Ergebnisse des Vergleiches zusammengefasst dargestellt.

Modell		SNMP		Webservices
Organisations-modell	o	Agent-Manager-Beziehung	o	Agent-Manager-Beziehung
	o	Agent hat rein informations-haltende Funktion	o	Agent implementiert lokale Managementfunktionen eigenständig
	o	Manager implementiert die komplette Managementlogik	o	Manager implementiert die globale Netzwerksicht
	o	Proxies als Zwischeninstanz	+	Intermediär zur Skalierung vorhanden
	+	Agent hat kleinen Prozess-aufwand	-	hoher Prozessaufwand des Agenten
	-	doppeldeutige, unvollständige RFCs ([3] S.4)	-	ohne Standardisierung gefahr der Überflutung durch proprietäre Webservices
Informations-modell	-	Datenstrukturen nur in Form von Tabellen möglich	+	zusammengesetzte Daten-strukturen modellierbar
	o	hierarchische zentrale Registrierung der Informationen in einem Objektidentifizierungsbaum	o	Einteilung in Namespaces
	-	festgelegte Anzahl von Datentypen	+	frei definierbare Datentypen ohne Bit-Grenzen
	-	keine direkte Möglichkeit Aktionen (RPCs) zu modellieren	+	RPCs Grundkonzept der Modellierung
	-	`MODULE-COMPLIANCE`-, `AGENT-CAPABILITIES`-Makros werden von wenigen Herstellern veröffentlicht, nur in wenige Manager Integrierbar	+	zu jedem Webservice gehört eine ihn beschreibende Definitionsdatei (WSDL)
Kommunikations-modell	+	standardisierte Zugriffsfunktionen	-	Namensraumabhängige individuelle Zugriffsfunktionen
	+	geringer Kommunikations-aufwand	-	hoher Kommunikations-aufwand (Prozessaufwand, Speicherbedarf)
	+	geringe Nachrichtengröße	-	umfangreiche Nachrichten-größe

Fortsetzung auf der nächsten Seite

Modell		SNMP		Webservices
Kommunikations-modell	o	binäres Übertragungsformat, nur von speziellen Empfängern lesbar	+	menschlich lesbares Übertragungsformat, in vielen bestehenden Anwendungen integriert
	o	Protokollbindung primär an UDP	+	freie Wahl der Protokollbindung, durch HTTP “firewall-freundlich”
	-	Durch das Informationsmodell sind Transaktionen nur schwer modellierbar	+	Transaktionen einfach Modellierbar
Implementierungs-möglichkeiten	o	wenige freie Entwicklungs-umgebungen	+	Unterstützung in fast allen Programmiersprachen
Bewertungssymbole: + positiv, - negativ, o neutral				

Tabelle 7.1.: Zusammenfassung und Gegenüberstellung der Eigenschaften der beiden Verfahren

8. Zusammenfassung und Ausblick

Webservices bieten eine funktionsorientierte moderne Datenmodellierung, die gerade durch die Verwendung von XML eine schnelle Verbreitung finden dürfte. Die Einstiegsbarriere zur Entwicklung von Managementlösungen ist durch die semantische Abbildung in menschlich lesbarer Form wesentlich niedriger als in das technisch orientierte SNMP-Protokoll. Diese Barriere mag auch ein Grund dafür sein, dass es bis heute relativ wenige qualitativ hochwertige Werkzeuge zum Management von Rechnernetzen mittels SNMP gibt, die über das Entwicklungsstadium hinausgetreten sind. Ein weiterer Grund, dass die Zugangsbarriere zu Webservices niedriger als im Fall von SNMP sein kann, liegt zum einen in der Integration von SOAP in heute schon relativ viele Programmiersprachen, zum anderen in der damit verbundenen Verbreitung der Basistechnologie. Es dürfte heutzutage relativ viele Entwickler geben, die sich zumindest ein wenig im XML-Umfeld auskennen und denen es somit leichter fallen dürfte die SOAP-Technologie zu verstehen als die SNMP-Technologie. Auch die Tatsache, dass die Managementlogik vom Agenten selbst ausgeführt wird, entlastet den Entwickler des Managers.

Im Bereich des verteilten Netzwerkmanagements mit Teilnetzmanagern, wie im Strukturmodell skizziert, liegt ein großes Potential des Webservice-Ansatzes, da dieser eine einfachere Modellierung durch die weitere Instanz der Intermediäre bietet. Auch ist die Modellierung von komplexen Vorgängen wie Transaktionen in SOAP kein Problem.

Bei all den Vorteilen, die Webservices im Bezug auf die Datenmodellierung bieten, muss man berücksichtigen, dass der Prozessaufwand des Agenten und das Kommunikationsvolumen um ein erhebliches Maß gesteigert wird. Auch die Entwicklung von generischen Werkzeugen für Webservices, die im Fall von SNMP ohne Probleme neue MIBs integrieren, dürfte nur mit erheblichen Mehraufwand möglich sein. Weiterhin steht der heute große Verbreitungsgrad von SNMP-Implementierungen auf Geräten und der hohe Standardisierungsgrad dieses Rahmenwerkes einem neuen Webservice-Management-Rahmenwerk entgegen.

Außerdem muss bei der Bewertung berücksichtigt werden, dass durch die lange Praxiserfahrung, die mit SNMP gesammelt wurde, zahlreichere Schwächen aufgedeckt und dokumentiert sind, als bei einem neuen, noch in der Standardisierung befindlichen Ansatz.

Die weitere Verfolgung des Webserviceansatzes bietet gerade im Bereich des verteilten, automatisierten Netzwerkmanagements ein hohes Entwicklungspotential. Wie schnell sich ein neuer Ansatz gegenüber SNMP etablieren kann, liegt im Wesentlichen in der Bereitschaft der Gerätehersteller, den erhöhten Entwicklungsaufwand des Agenten mit zu tragen.

Literaturverzeichnis

[1] HEGERING, Heinz-Gerd ; ABECK, Sebastian ; NEUMAIR, Bernhard: *Integriertes Management vernetzter Systeme*. erste Auflage. Heidelberg : dpunkt-Verlag, 1999. – ISBN 3-932588-16-9

[2] STALLINGS, William: SNMPv3: A Security Enhancement for SNMP / IEEE. 1998 (4.Quartal 1998). – IEEE Communication Surveys & Tutorials. http://www.comsoc.org/livepubs/surveys/public/4q98issue/stallings.html

[3] PERKINS, David ; MCGINNIS, Evan: *Understanding SNMP MIBS*. erste Auflage. Upper Saddle River, New Jersey : Prentice-Hall, 1997. – ISBN 0-13-437708-7

[4] SNELL, James ; TIDWELL, Doug ; KULCHENKO, Pavel: *Webservice-Programmierung mit SOAP*. erste Auflage. Köln : O'Reilly, 2002. – ISBN 3-89721-159-9

[5] VAN ENGELEN, Robert A.: *gSOAP - Howto*. http://www.cs.fsu.edu/∼engelen/soap.html: Florida State University, Department of Computer Science

[6] ROSE, Marshall T.: *The Simple Book*. erste. Englewood Cliffs, New Jersey : Prentice-Hall, 1991. – ISBN 0-13-812611-9

[7] SMITH, R. ; WRIGHT, F. ; HASTINGS, T. ; GYLLENSKOG, J.: Printer-MIB / IETF, Network Working Group. 1995. – RFC 1759

[8] MCCLOGHRIE, K. ; ROSE, M.: Management Information Base for Network Management of TCP/IP-based internets: MIB-II / IETF, Network Working Group. 1991. – RFC 1213

[9] MITRA, Nilo: SOAP Version 1.2 Part 0: Primer / W3C. 2003. – W3C Recommendation. http://www.w3.org/TR/2003/REC-soap12-part0-20030624

[10] STALLINGS, William: *SNMP, SNMPv2, and RMON*. zweite Auflage. Reading, Massachusetts : Addison Wesley, 1996. – ISBN 0-201-63479-1

[11] TANENBAUM, Andrew S.: *Computernetzwerke*. dritte Auflage. München : Prentice-Hall, 2000. – ISBN 3-8273-7011-6

[12] CASE, J. ; FEDOR, M. ; SCHOFFSTALL, M. ; DAVIN, J.: A Simple Network Management Protocol (SNMP) / IETF, Network Working Group. 1990. – RFC 1157

[13] MCCLOGHRIE, K. ; ROSE, M.: Structure and Identification of Management Information / IETF, Network Working Group. 1990. – RFC 1155

[14] STALLINGS, William: *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. dritte Auflagr. Reading, Massachusetts : Addison Wesley Longman, Inc., 1999. – ISBN 0-201-48534-6

[15] MCCLOGHRIE, K. ; PERKINS, D. ; SCHÖNWÄLDER, J. ; CASE, J. ; MCCLOGHRIE, K. ; ROSE, M. ; WALDBUSSER, S.: Structure of Management Information Version 2 (SMIv2) / IETF, Network Working Group. 1999. – RFC 2578

[16] MCCLOGHRIE, K. ; PERKINS, D. ; SCHOENWAELDER, J.: Textual Conventions for SMIv2 / IETF, Network Working Group. 1999. – RFC 2579

[17] PRESUHN, R.: Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP) / IETF, Network Working Group. 2002. – RFC 3416

[18] CASE, J. ; MCCLOGHRIE, K. ; ROSE, M. ; WALDBUSSER, S.: Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2) / IETF, Network Working Group. 1993. – RFC 1448

[19] PRESUHN, R.: Transport Mappings for the Simple Network Management Protocol (SNMP) / IETF, Network Working Group. 2002. – RFC 3417

[20] JANSSEN, Rainer ; SCHOTT, Wolfgang: *SNMP Konzepte-Verfahren-Plattformen*. erste Auflage. Bergheim : DATACOM, 1993. – ISBN 3-89238-077-5

[21] BLUMENTHAL, U.: User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) / IETF, Network Working Group. 1998. – RFC 2274

[22] WIJNEN, B. ; PRESUHN, R. ; MCCLOGHRIE, K.: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) / IETF, Network Working Group. 1999. – RFC 2274

[23] *net-snmp - Various tools relating to the Simple Network Management Protocol*. http://net-snmp.sourceforge.net

[24] SCHÖNWÄLDER, Jürgen: Specific Simple Network Management Tools. In: *LISA 2001 15th System Administration Conference*. San Diego (California) : The USENIX Association, Dezember 2001, S. 109–119

[25] OETIKER, T.: MRTG - Multi Router Traffic Grapher. In: *Tweltfth Conference on Large Installation System Administration (LISA XII)*. Boston : The USENIX Association, November 1998

[26] SCHÖNWÄLDER, J. ; LANGENDÖRFER, H.: How To Keep Track of Your Network Configuration. In: *Seventh Conference on Large Installation System Administration (LISA VII)*. Monterey (California) : The USENIX Association, November 1993, S. 189–193

[27] BLOMMERS, John: *OpenView Network Node Manager: Designing and Implementing and Enterprise Solution*. first Edition. Prentice-Hall PTR, 2000. – ISBN 0-13019-849-8

[28] Institut für Betriebssysteme und Rechnerverbund , TU Braunschweig: *scli - SNMP Command Line Interface*. – http://www.ibr.cs.tu-bs.de/projects/scli/

[29] SCHÖNWÄLDER, Jürgen: *Scotty - Tcl Extensions for Network Management Applications*. http://www.ibr.cs.tu-bs.de/projects/scotty/: Institut für Betriebssysteme und Rechnerverbund , TU Braunschweig

[30] DIV: *XML Professionell*. erste Auflage. Bonn : MITP-Verlag, 2000. – ISBN 3-82660-633-7

[31] GUDGIN, Martin ; HADLEY, Marc ; MENDELSOHN, Noah ; MOREAU, Jean-Jacques ; NIELSEN, Henrik F.: SOAP Version 1.2 Part 1: Messaging Framework / W3C. 2003. – W3C Recommendation. http://www.w3.org/TR/2003/REC-soap12-part1-20030624

[32] GUDGIN, Martin ; HADLEY, Marc ; MENDELSOHN, Noah ; MOREAU, Jean-Jacques ; NIELSEN, Henrik F.: SOAP Version 1.2 Part 2: Adjuncts / W3C. 2003. – W3C Recommendation. http://www.w3.org/TR/2003/REC-soap12-part2-20030624

[33] COWAN, John ; TOBIN, Richard: – Forschungsbericht

[34] FIELDING, R. ; GETTYS, J. ; MOGUL, J. C. ; FRYSTYK, H. ; BERNERS-LEE, T.: Hypertext Transfer Protocol – HTTP/1.1 / IETF, Network Working Group. 1997. – RFC 2616

[35] BERNERS-LEE, T. ; FIELDING, R. ; IRVINE, U.C. ; MASINTER, L.: Uniform Resource Identifiers (URI): Generic Syntax / IETF, Network Working Group. 1998. – RFC 2396

[36] CHINNICI, Roberto ; GUDGIN, Martin ; MOREAU, Jean-Jacques ; WEERAWARANA, Sanjiva: Web Services Description Language (WSDL) Version 1.2 / W3C. 2002. – W3C Working Draft. http://www.w3.org/TR/2002/WD-wsdl12-20020709

[37] MOREAU, Jean-Jacques ; SCHLIMMER, Jeffrey: Web Services Description Language (WSDL) Version 1.2: Bindings / W3C. 2002. – W3C Working Draft. http://www.w3.org/TR/2002/WD-wsdl12-bindings-20020709

[38] VAN ENGELEN, Robert A.: *The gSOAP Stub and Skeleton Compiler for C and C++ 2.2.1*. http://www.cs.fsu.edu/∼engelen/soap.html: Florida State University, Department of Computer Science, Dezember 2002

[39] ISO: Data elements and interchange formats - Information interchange - Representation of dates and times / ISO. 2000 (8601:2002). – International Standard

[40] PAUL V. BIRON, Ashok M.: XML Schema Part 2: Datatypes / W3C. 2001. – W3C Candidate Recommendation. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502

[41] MCCLOGHRIE, K. ; KASTENHOLZ, F.: The Interfaces Group MIB / IETF, Network Working Group. 2000. – RFC 2863

[42] CHIU, Kenneth ; GOVINDARAJU, Madhusudhan ; BRAMLEY, Randall: Investigating the Limits of SOAP Performance for Scientific Computing. In: *Proceedings of The Eleventh International Symposium on High Performance Distributed Computing*. Edinburgh : IEEE Computer Society Press, Juli 2002, S. 246–254

A. Anwendung des Prototypen

Die drei Programmteile Agent, Manager und Eventreceiver können mit dem Makefile aus dem Hauptverzeichnis kompiliert werden. Entwicklungsumgebung des Prototypen war Linux in der Kernelversion 2.4.20 und dem gcc 3.2.2. Zur Kompilierung ist neben den Standard-C-Bibliotheken eine Installation des gSOAP-Compilers[1] und der *libdnet*[2] notwendig.

A.1. Benutzung

Der Prototyp besteht aus drei Programmen in den Verzeichnissen `agent`, `manager` und `eventreceiver`. In diesen befinden sich das Programm für den Agenten, *SOAPagent*, für den Manager, *SOAPmanager*, und den Ereignisempfänger, *SOAPeventreceiver*.

Alle Programme geben mit der Option `-?` einen Hilfetext zu ihren Kommandozeilenoptionen aus. Die Standardports, auf denen der Agent und der Eventreceiver auf eingehende Nachrichten warten, sind Port 10080 für den Agenten und Port 10081 für den Eventreceiver. Diese Ports sind neben dem Hostnamen (`localhost`) in allen drei Programmen als Default gesetzt. Falls eine Kommunikation über das Netzwerk angestrebt wird, sollte als Hostname immer die IP-Adresse des Rechners als Hostname angegeben werden, um eine korrekte Kommunikation zu gewährleisten.

Nach dem Start des Agenten können beide anderen Programme mit dem Agenten kommunizieren.

A.2. Befehlsbeispiele des Managers

Abfragen der System-Statistiken eines Agenten:

```
./SOAPmanager -s
```

Bei den Optionen für das Routing (-r) und die Schnittstellenkonfiguration `-i` kann mit Option `-c` angegeben werden, was mit den restlichen Optionen der Kommandozeile gemacht werden soll. Standardmäßig ist als Subkommando `GET` eingestellt, welches ein Abfragen der Informationen bewirkt.

Somit haben die beiden folgenden Kommandos denselben Effekt:

```
./SOAPmanager -r
./SOAPmanager -r -c GET
```

[1] http://www.cs.fsu.edu/~engelen/soap.html
[2] http://libdnet.sourceforge.net

Mit der zusätzlichen Angabe des Schnittstellennamens werden nur Einträge dieser Schnittstelle angefordert:

```
./SOAPmanager -r
./SOAPmanager -r --ifname eth0
```

Falls man Schnittstelleninformationen abrufen möchte, muss man hinter der Option `-i` zusätzlich die Art der Information angeben, ob man den Status der Schnittstellen oder deren Konfigurationsinformationen erhalten möchte:

```
./SOAPmanager -i STAT
./SOAPmanager -i INF
```

Das Setzen von Konfigurationseinstellungen funktioniert im Prototypen nur mit den Schnittstelleninformationen. Im Folgenden ein Kommando, um eine Schnittstelle in den UP-Zustand zu versetzen:

```
./SOAPmanager -i INF -c SET --ifname eth1 --flag UP
```

A.3. Befehlsbeispiele des Agenten

Der Agent besitzt nur zwei Optionen zum Angeben des Hostnamens `-h` und des Ports `-p` an den der Socket des Agenten gebunden werden soll.
Um den Agenten mit den Standardoptionen zu starten, sind beide Aufrufe äquivalent:

```
./SOAPagent
./SOAPagent -h localhost -p 10080
```

A.4. Befehlsbeispiele des Eventreceivers

Der Eventreceiver besitzt neben den Optionen für die eigenen Socketparamter Hostname (`-h`) und Portnummer (`-p`) auch die Option für die Socketparameter des Angenten (`-s, -r`). Falls der Agent auf `localhost` mit dem Defaultport läuft, ist eine start ohne Angabe von Optionen möglich.
Falls auf Hostname:`192.168.0.1`, Port `10090` ein Agent läuft, ist folgender Aufruf nötig

```
./SOAPeventreceiver -s 192.168.0.1 -r 10090
```

Eine weitere Option ermöglicht es die Art der zu empfangenden Ereignismeldungen einzuschränken. Mit der Option `-e` kann man den Typ der Erignismeldungen bestimmen. Man kann eine Auswahl zwischen Schnittstellenereignisse und Routingereignissen treffen.

Im Folgenden ist diese Auswahl mit den Schnittstellenerignissen und dann den Routingereignissen dargestellt.

```
./SOAPeventreceiver -e IF
./SOAPeventreceiver -e RT
```

B. Dateien des Webservice Prototypen

B.1. devmgrSOAP.h

Diese Header-Datei ist keine reine C-Header Datei, sondern enthält Ausgangsdefinitionen für den Stub-Compiler `soapcpp2`, um daraus die SOAP-Stub-Dateien zu erzeugen.

```
typedef char *xsd__string;
typedef double xsd__double;
typedef int xsd__int;
typedef char *xsd__address;
typedef short xsd__short;
typedef char *xsd__duration;


enum ns__addrType {IP=1,ETH=2};

struct ns__address
{
  @ enum ns__addrType type;           /* Type of the address */
  xsd__string addr;                   /* Address encoded as String */
};


enum nsif__ifType {IF_NONE=0,IF_OTHER=1,IF_ETHERNET=6,
                 IF_LOOPBACK=24,IF_VIRTUAL=53}; /* other,
                                                Ethernet,
                                                software loopback,
                                                proprietary virtual/internal */



struct nsif__interfaceDescription
{
  enum nsif__ifType type;             /* Interface type   */
```

```
  xsd__string       name;             /* Interface name   */
  xsd__int          mtu;              /* Interface MTU    */
  int               __sizeflags;      /* Flagarray Size   */
  xsd__string       *flag;            /* Interface flagarray */
  struct ns__address *address;        /* Interface address */
  struct ns__address *netmask;        /* Interface Netmask */
  struct ns__address *broadcast;      /* Broadcast Address */
  struct ns__address *dst_address;    /* point-to-point dst */
  struct ns__address *link_address;   /* link-layer address */
  int               __sizealias;      /* Aliasarray size  */
  struct ns__address **alias;         /* Interface aliasarray */

};


struct interfacelist
{
  struct nsif__interfaceDescription *__ptr;
  int __size;
  int __offset;
};



struct nsif__interfaceStats
{
  xsd__string    ifname;            /* Interface name   */
  xsd__int       rx_bytes     0:1;  /* Received Bytes   */
  xsd__int       rx_packets   0:1;  /* Received Packets */
  xsd__int       rx_errors    0:1;  /* Number of errors on ingoing traffic */
  xsd__int       rx_drop      0:1;  /* Number of dropped packets */
  xsd__int       rx_fifo      0:1;  /* Number of ingoing FIFO buffer errors */
  xsd__int       rx_frame     0:1;  /* Number of ingoing packet framing errors
      .*/
  xsd__int       rx_compressed 0:1; /* Number of compressed packets received */
  xsd__int       rx_multicast 0:1;  /* Number of received multicast frames */
  xsd__int       tx_bytes     0:1;  /* Transmitted Bytes   */
  xsd__int       tx_packets   0:1;  /* Transmitted Packets */
  xsd__int       tx_errors    0:1;  /* Number of errors on outgoing traffic */
  xsd__int       tx_drop      0:1;  /* Number of dropped packets */
  xsd__int       tx_fifo      0:1;  /* Number of outgoing FIFO buffer errors */
  xsd__int       tx_colls     0:1;  /* Number of collisions detected on the
      interface */
  xsd__int       tx_carrier   0:1;  /* Number of carrier losses detected by the
```

```
      device driver */
  xsd__int      tx_compressed 0:1; /* Number of compressed packets transmitted
       */
};


struct ifstatlist
{
  struct nsif__interfaceStats *__ptr;
  int __size;
  int __offset;
};



struct nsrt__routingInformation
{
  xsd__string       iface;               /* Interface Name */
  struct ns__address *dst_addr;          /* Destination address */
  struct ns__address *gw_addr;           /* Gateway address */
  struct ns__address *nm_addr;           /* Netmask */
  int               __sizeflags;         /* Flagarray Size */
  xsd__string      *flag;                /* Interface flagarray */
  xsd__int         mtu;                  /* Route maximum transfer unit */
  xsd__int         metric;               /* Route metric */
  xsd__int         refcnt;               /* Number of references to this route
      . */
  xsd__int         use;                  /* Count of lookups for the route. */
  xsd__int         window;               /* TCP window size */
  xsd__int         irtt;                 /* initial round trip time */
};


/* FLAGS for nsrt__routingInformation
   UP        (route is up)
   HOST      (target is a host)
   GATEWAY   (use gateway)
   REINSTATE (reinstate route for dynamic routing)
   DYNAMIC   (dynamically installed by daemon or redirect)
   MODIFIED  (modified from routing daemon or redirect)
   ADDRCONF  (installed by addrconf)
   CACHE     (cache entry)
   REJECT    (reject route)(!)
*/
```

```
struct routelist
{
  struct nsrt__routingInformation *__ptr;
  int __size;
  int __offset;
};


struct nssys__sysInf{
  xsd__string   description 1:1;
  xsd__duration uptime      0:1;
  xsd__duration idletime    0:1;
};


int nssys__uptime(xsd__string *result);
int nssys__getSystemInformation(struct nssys__sysInf *result);
int nsif__getInterfaceInformation(struct nsif__interfaceDescription in,struct
    interfacelist *result);
int nsif__setInterfaceInformation(struct nsif__interfaceDescription in,struct
    interfacelist *result);
int nsif__getInterfaceStats(xsd__string ifname,struct ifstatlist *result);
int nsrt__getRoutingEntrys(struct nsrt__routingInformation in,struct routelist
     *result);



struct SOAP_ENV__Header
{ xsd__int connection_ID;
};


enum nsevt__type {EVT_TP_SUCCESS=0,EVT_TP_IF=1,EVT_TP_ROUTE=2,EVT_TP_SRVCLOSE
    =3,EVT_TP_DEFAULT=4,EVT_TP_FAULT=5};
enum nsevt__regType{EVT_REGTYPE_NONE=0,EVT_REGTYPE_IF=2,EVT_REGTYPE_RT=3,
    EVT_REGTYPE_ALL=4,EVT_REGTYPE_FW=5};

int nsevt__registerEventHandler(xsd__int portnumber,xsd__string hostname,enum
```

```
    nsevt__regAction regact,enum nsevt__regType tp,xsd__int *result);

enum nsevt__regAction{EVT_REGISTER=1,EVT_UNREGISTER=2};

// importing



int nsevt__signalEvent(xsd__string error,enum nsevt__type msgtype,enum
    nsevt__type *deliverySuccess);




// defining namespaces for the stub compiler

//gsoap nsevt service namespace: urn:WS_mgmt_stdEvent
//gsoap ns service namespace: urn:WS_mgmt_stdDefs
//gsoap nsif service namespace: urn:WS_mgmt_stdInterface
//gsoap nsrt service namespace: urn:WS_mgmt_stdRoute
//gsoap nssys service namespace: urn:WS_mgmt_stdSystem
```

Listing B.1: devmgrSOAP.h

B.2. Webservice Definitionen

B.2.1. ns.xsd

Diese XML-Schema Definition enthält die Strukturen des Namensbereiches `WS_mgmt_stdDefs`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/* ns
   Generated by gSOAP 2.3 rev 2 from devmgrSOAP.h
   Copyright (C) 2001-2003 Genivia inc.
   All Rights Reserved.
*/-->
 <schema targetNamespace="urn:WS_mgmt_stdDefs"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="urn:WS_mgmt_stdDefs"
xmlns:nssys="urn:WS_mgmt_stdSystem"
xmlns:nsif="urn:WS_mgmt_stdInterface"
xmlns:nsrt="urn:WS_mgmt_stdRoute"
xmlns:nsevt="urn:WS_mgmt_stdEvent"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">

<element name="addrType" type="ns:addrType"/>
<simpleType name="addrType">
 <restriction base="xsd:string">
  <enumeration value="IP"/>
  <enumeration value="ETH"/>
 </restriction>
</simpleType>

<element name="address" type="ns:address"/>
<complexType name="address">
 <sequence>
  <element name="addr" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
      ="true"/>
 </sequence>
 <attribute name="type" type="ns:addrType" use="required"/>
</complexType>

<complexType name="ArrayOfinterfaceDescription">
 <complexContent>
  <restriction base="SOAP-ENC:Array">
   <sequence>
    <element name="item" type="nsif:interfaceDescription" minOccurs="0"
        maxOccurs="unbounded"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>

<complexType name="ArrayOfinterfaceStats">
 <complexContent>
  <restriction base="SOAP-ENC:Array">
   <sequence>
    <element name="item" type="nsif:interfaceStats" minOccurs="0" maxOccurs="
        unbounded"/>
```

```
      </sequence>
    </restriction>
  </complexContent>
</complexType>

<complexType name="ArrayOfroutingInformation">
  <complexContent>
    <restriction base="SOAP-ENC:Array">
      <sequence>
        <element name="item" type="nsrt:routingInformation" minOccurs="0"
            maxOccurs="unbounded"/>
      </sequence>
    </restriction>
  </complexContent>
</complexType>

</schema>
```

Listing B.2: ns.xsd

B.2.2. nsif.wsdl

Diese WSDL-Definition enthält die Definitionen des Namensbereiches `WS_mgmt_stdInterface`.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Service"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 targetNamespace="urn:WS_mgmt_stdInterface"
 xmlns:tns="urn:WS_mgmt_stdInterface"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent">

<types>
```

```
<schema targetNamespace="urn:WS_mgmt_stdDefs"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <element name="addrType" type="ns:addrType"/>
 <simpleType name="addrType">
  <restriction base="xsd:string">
   <enumeration value="IP"/>
   <enumeration value="ETH"/>
  </restriction>
 </simpleType>

 <element name="address" type="ns:address"/>
 <complexType name="address">
  <sequence>
   <element name="addr" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
       ="true"/>
  </sequence>
  <attribute name="type" type="ns:addrType" use="required"/>
 </complexType>

</schema>
<schema targetNamespace="urn:WS_mgmt_stdSystem"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
```

```
 attributeFormDefault="unqualified">

 <element name="sysInf" type="nssys:sysInf"/>
 <complexType name="sysInf">
  <sequence>
   <element name="description" type="xsd:string" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="uptime" type="xsd:duration" minOccurs="0" maxOccurs="1"
       nillable="true"/>
   <element name="idletime" type="xsd:duration" minOccurs="0" maxOccurs="1"
       nillable="true"/>
  </sequence>
 </complexType>

</schema>
<schema targetNamespace="urn:WS_mgmt_stdInterface"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <element name="ifType" type="nsif:ifType"/>
 <simpleType name="ifType">
  <restriction base="xsd:string">
   <enumeration value="IF-NONE"/>
   <enumeration value="IF-OTHER"/>
   <enumeration value="IF-ETHERNET"/>
   <enumeration value="IF-LOOPBACK"/>
   <enumeration value="IF-VIRTUAL"/>
  </restriction>
 </simpleType>

 <element name="interfaceDescription" type="nsif:interfaceDescription"/>
 <complexType name="interfaceDescription">
  <sequence>
   <element name="type" type="nsif:ifType" minOccurs="1" maxOccurs="1"/>
```

```
    <element name="name" type="xsd:string" minOccurs="1" maxOccurs="1" nillable="true"/>
    <element name="mtu" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="flag" type="xsd:string" minOccurs="1" maxOccurs="unbounded" nillable="true"/>
    <element name="address" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
    <element name="netmask" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
    <element name="broadcast" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
    <element name="dst-address" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
    <element name="link-address" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
    <element name="alias" type="ns:address" minOccurs="1" maxOccurs="unbounded" nillable="true"/>
   </sequence>
  </complexType>

  <complexType name="ArrayOfinterfaceDescription">
   <complexContent>
    <restriction base="SOAP-ENC:Array">
     <sequence>
      <element name="item" type="nsif:interfaceDescription" minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
     <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsif:interfaceDescription[]"/>
    </restriction>
   </complexContent>
  </complexType>

  <element name="interfaceStats" type="nsif:interfaceStats"/>
  <complexType name="interfaceStats">
   <sequence>
    <element name="ifname" type="xsd:string" minOccurs="1" maxOccurs="1" nillable="true"/>
    <element name="rx-bytes" type="xsd:int" minOccurs="0" maxOccurs="1"/>
    <element name="rx-packets" type="xsd:int" minOccurs="0" maxOccurs="1"/>
    <element name="rx-errors" type="xsd:int" minOccurs="0" maxOccurs="1"/>
    <element name="rx-drop" type="xsd:int" minOccurs="0" maxOccurs="1"/>
    <element name="rx-fifo" type="xsd:int" minOccurs="0" maxOccurs="1"/>
    <element name="rx-frame" type="xsd:int" minOccurs="0" maxOccurs="1"/>
```

```
   <element name="rx-compressed" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-multicast" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-bytes" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-packets" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-errors" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-drop" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-fifo" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-colls" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-carrier" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-compressed" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  </sequence>
 </complexType>

 <complexType name="ArrayOfinterfaceStats">
  <complexContent>
   <restriction base="SOAP-ENC:Array">
    <sequence>
     <element name="item" type="nsif:interfaceStats" minOccurs="0" maxOccurs="
         unbounded"/>
    </sequence>
    <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsif:interfaceStats
        []"/>
   </restriction>
  </complexContent>
 </complexType>

 <complexType name="ArrayOfroutingInformation">
  <complexContent>
   <restriction base="SOAP-ENC:Array">
    <sequence>
     <element name="item" type="nsrt:routingInformation" minOccurs="0"
         maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsrt:
        routingInformation[]"/>
   </restriction>
  </complexContent>
 </complexType>

</schema>
<schema targetNamespace="urn:WS_mgmt_stdRoute"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <element name="routingInformation" type="nsrt:routingInformation"/>
 <complexType name="routingInformation">
  <sequence>
   <element name="iface" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
       ="true"/>
   <element name="dst-addr" type="ns:address" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="gw-addr" type="ns:address" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="nm-addr" type="ns:address" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="flag" type="xsd:string" minOccurs="1" maxOccurs="unbounded"
       nillable="true"/>
   <element name="mtu" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="metric" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="refcnt" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="use" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="window" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="irtt" type="xsd:int" minOccurs="1" maxOccurs="1"/>
  </sequence>
 </complexType>

</schema>
<schema targetNamespace="urn:WS_mgmt_stdEvent"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
```

```
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <element name="type" type="nsevt:type"/>
  <simpleType name="type">
   <restriction base="xsd:string">
    <enumeration value="EVT-TP-SUCCESS"/>
    <enumeration value="EVT-TP-IF"/>
    <enumeration value="EVT-TP-ROUTE"/>
    <enumeration value="EVT-TP-SRVCLOSE"/>
    <enumeration value="EVT-TP-DEFAULT"/>
    <enumeration value="EVT-TP-FAULT"/>
   </restriction>
  </simpleType>

  <element name="regType" type="nsevt:regType"/>
  <simpleType name="regType">
   <restriction base="xsd:string">
    <enumeration value="EVT-REGTYPE-NONE"/>
    <enumeration value="EVT-REGTYPE-IF"/>
    <enumeration value="EVT-REGTYPE-RT"/>
    <enumeration value="EVT-REGTYPE-ALL"/>
    <enumeration value="EVT-REGTYPE-FW"/>
   </restriction>
  </simpleType>

  <element name="regAction" type="nsevt:regAction"/>
  <simpleType name="regAction">
   <list>
    <restriction base="xsd:string">
     <enumeration value="EVT-REGISTER"/>
     <enumeration value="EVT-UNREGISTER"/>
    </restriction>
   </list>
  </simpleType>

 </schema>
</types>

<message name="getInterfaceInformationRequest">
 <part name="in" type="nsif:interfaceDescription"/>
</message>

<message name="getInterfaceInformationResponse">
```

```
 <part name="result" type="nsif:ArrayOfinterfaceDescription"/>
</message>

<message name="setInterfaceInformationRequest">
 <part name="in" type="nsif:interfaceDescription"/>
</message>

<message name="setInterfaceInformationResponse">
 <part name="result" type="nsif:ArrayOfinterfaceDescription"/>
</message>

<message name="getInterfaceStatsRequest">
 <part name="ifname" type="xsd:string"/>
</message>

<message name="getInterfaceStatsResponse">
 <part name="result" type="nsif:ArrayOfinterfaceStats"/>
</message>

<message name="ServiceHeader">
 <part name="connection-ID" type="xsd:int"/>
</message>

<portType name="ServicePortType">
 <operation name="getInterfaceInformation">
  <documentation>Service definition of function nsif__getInterfaceInformation</
      documentation>
  <input message="tns:getInterfaceInformationRequest"/>
  <output message="tns:getInterfaceInformationResponse"/>
 </operation>
 <operation name="setInterfaceInformation">
  <documentation>Service definition of function nsif__setInterfaceInformation</
      documentation>
  <input message="tns:setInterfaceInformationRequest"/>
  <output message="tns:setInterfaceInformationResponse"/>
 </operation>
 <operation name="getInterfaceStats">
  <documentation>Service definition of function nsif__getInterfaceStats</
      documentation>
  <input message="tns:getInterfaceStatsRequest"/>
  <output message="tns:getInterfaceStatsResponse"/>
 </operation>
</portType>
```

```
<binding name="ServiceBinding" type="tns:ServicePortType">
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="getInterfaceInformation">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdInterface" encodingStyle
       ="http://www.w3.org/2002/12/soap-encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdInterface" encodingStyle
       ="http://www.w3.org/2002/12/soap-encoding"/>
  </output>
 </operation>
 <operation name="setInterfaceInformation">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdInterface" encodingStyle
       ="http://www.w3.org/2002/12/soap-encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdInterface" encodingStyle
       ="http://www.w3.org/2002/12/soap-encoding"/>
  </output>
 </operation>
 <operation name="getInterfaceStats">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdInterface" encodingStyle
       ="http://www.w3.org/2002/12/soap-encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdInterface" encodingStyle
       ="http://www.w3.org/2002/12/soap-encoding"/>
  </output>
 </operation>
</binding>

<service name="Service">
 <documentation>gSOAP 2.3 rev 2 generated service definition</documentation>
 <port name="Service" binding="tns:ServiceBinding">
  <SOAP:address location="http://localhost"/>
 </port>
</service>
```

```
</definitions>
```

Listing B.3: nsif.wsdl

B.2.3. nsrt.wsdl

Diese WSDL-Definition enthält die Definitionen des Namensbereiches `WS_mgmt_stdRoute`.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Service"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 targetNamespace="urn:WS_mgmt_stdRoute"
 xmlns:tns="urn:WS_mgmt_stdRoute"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent">

<types>
 <schema targetNamespace="urn:WS_mgmt_stdDefs"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:WS_mgmt_stdDefs"
  xmlns:nssys="urn:WS_mgmt_stdSystem"
  xmlns:nsif="urn:WS_mgmt_stdInterface"
  xmlns:nsrt="urn:WS_mgmt_stdRoute"
  xmlns:nsevt="urn:WS_mgmt_stdEvent"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <element name="addrType" type="ns:addrType"/>
  <simpleType name="addrType">
   <restriction base="xsd:string">
```

```
    <enumeration value="IP"/>
    <enumeration value="ETH"/>
   </restriction>
  </simpleType>

  <element name="address" type="ns:address"/>
  <complexType name="address">
   <sequence>
    <element name="addr" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
        ="true"/>
   </sequence>
   <attribute name="type" type="ns:addrType" use="required"/>
  </complexType>

 </schema>
 <schema targetNamespace="urn:WS_mgmt_stdSystem"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:WS_mgmt_stdDefs"
  xmlns:nssys="urn:WS_mgmt_stdSystem"
  xmlns:nsif="urn:WS_mgmt_stdInterface"
  xmlns:nsrt="urn:WS_mgmt_stdRoute"
  xmlns:nsevt="urn:WS_mgmt_stdEvent"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <element name="sysInf" type="nssys:sysInf"/>
  <complexType name="sysInf">
   <sequence>
    <element name="description" type="xsd:string" minOccurs="1" maxOccurs="1"
        nillable="true"/>
    <element name="uptime" type="xsd:duration" minOccurs="0" maxOccurs="1"
        nillable="true"/>
    <element name="idletime" type="xsd:duration" minOccurs="0" maxOccurs="1"
        nillable="true"/>
   </sequence>
  </complexType>

 </schema>
 <schema targetNamespace="urn:WS_mgmt_stdInterface"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
```

```
xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="urn:WS_mgmt_stdDefs"
xmlns:nssys="urn:WS_mgmt_stdSystem"
xmlns:nsif="urn:WS_mgmt_stdInterface"
xmlns:nsrt="urn:WS_mgmt_stdRoute"
xmlns:nsevt="urn:WS_mgmt_stdEvent"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">

<element name="ifType" type="nsif:ifType"/>
<simpleType name="ifType">
 <restriction base="xsd:string">
  <enumeration value="IF-NONE"/>
  <enumeration value="IF-OTHER"/>
  <enumeration value="IF-ETHERNET"/>
  <enumeration value="IF-LOOPBACK"/>
  <enumeration value="IF-VIRTUAL"/>
 </restriction>
</simpleType>

<element name="interfaceDescription" type="nsif:interfaceDescription"/>
<complexType name="interfaceDescription">
 <sequence>
  <element name="type" type="nsif:ifType" minOccurs="1" maxOccurs="1"/>
  <element name="name" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
      ="true"/>
  <element name="mtu" type="xsd:int" minOccurs="1" maxOccurs="1"/>
  <element name="flag" type="xsd:string" minOccurs="1" maxOccurs="unbounded"
      nillable="true"/>
  <element name="address" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="netmask" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="broadcast" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="dst-address" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="link-address" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="alias" type="ns:address" minOccurs="1" maxOccurs="unbounded"
      nillable="true"/>
```

```
  </sequence>
 </complexType>

 <element name="interfaceStats" type="nsif:interfaceStats"/>
 <complexType name="interfaceStats">
  <sequence>
   <element name="ifname" type="xsd:string" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="rx-bytes" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-packets" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-errors" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-drop" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-fifo" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-frame" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-compressed" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-multicast" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-bytes" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-packets" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-errors" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-drop" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-fifo" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-colls" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-carrier" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-compressed" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  </sequence>
 </complexType>

</schema>
<schema targetNamespace="urn:WS_mgmt_stdRoute"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <complexType name="ArrayOfinterfaceDescription">
  <complexContent>
```

```
    <restriction base="SOAP-ENC:Array">
     <sequence>
      <element name="item" type="nsif:interfaceDescription" minOccurs="0"
          maxOccurs="unbounded"/>
     </sequence>
     <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsif:
         interfaceDescription[]"/>
    </restriction>
   </complexContent>
  </complexType>

  <complexType name="ArrayOfinterfaceStats">
   <complexContent>
    <restriction base="SOAP-ENC:Array">
     <sequence>
      <element name="item" type="nsif:interfaceStats" minOccurs="0" maxOccurs="
          unbounded"/>
     </sequence>
     <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsif:interfaceStats
         []"/>
    </restriction>
   </complexContent>
  </complexType>

  <element name="routingInformation" type="nsrt:routingInformation"/>
  <complexType name="routingInformation">
   <sequence>
    <element name="iface" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
        ="true"/>
    <element name="dst-addr" type="ns:address" minOccurs="1" maxOccurs="1"
        nillable="true"/>
    <element name="gw-addr" type="ns:address" minOccurs="1" maxOccurs="1"
        nillable="true"/>
    <element name="nm-addr" type="ns:address" minOccurs="1" maxOccurs="1"
        nillable="true"/>
    <element name="flag" type="xsd:string" minOccurs="1" maxOccurs="unbounded"
        nillable="true"/>
    <element name="mtu" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="metric" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="refcnt" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="use" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="window" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="irtt" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   </sequence>
```

```
</complexType>

<complexType name="ArrayOfroutingInformation">
 <complexContent>
  <restriction base="SOAP-ENC:Array">
   <sequence>
    <element name="item" type="nsrt:routingInformation" minOccurs="0"
        maxOccurs="unbounded"/>
   </sequence>
   <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsrt:
       routingInformation[]"/>
  </restriction>
 </complexContent>
</complexType>

</schema>
<schema targetNamespace="urn:WS_mgmt_stdEvent"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <element name="type" type="nsevt:type"/>
 <simpleType name="type">
  <restriction base="xsd:string">
   <enumeration value="EVT-TP-SUCCESS"/>
   <enumeration value="EVT-TP-IF"/>
   <enumeration value="EVT-TP-ROUTE"/>
   <enumeration value="EVT-TP-SRVCLOSE"/>
   <enumeration value="EVT-TP-DEFAULT"/>
   <enumeration value="EVT-TP-FAULT"/>
  </restriction>
 </simpleType>

 <element name="regType" type="nsevt:regType"/>
 <simpleType name="regType">
```

```
   <restriction base="xsd:string">
    <enumeration value="EVT-REGTYPE-NONE"/>
    <enumeration value="EVT-REGTYPE-IF"/>
    <enumeration value="EVT-REGTYPE-RT"/>
    <enumeration value="EVT-REGTYPE-ALL"/>
    <enumeration value="EVT-REGTYPE-FW"/>
   </restriction>
  </simpleType>

  <element name="regAction" type="nsevt:regAction"/>
  <simpleType name="regAction">
   <list>
    <restriction base="xsd:string">
     <enumeration value="EVT-REGISTER"/>
     <enumeration value="EVT-UNREGISTER"/>
    </restriction>
   </list>
  </simpleType>

 </schema>
</types>

<message name="getRoutingEntrysRequest">
 <part name="in" type="nsrt:routingInformation"/>
</message>

<message name="getRoutingEntrysResponse">
 <part name="result" type="nsrt:ArrayOfroutingInformation"/>
</message>

<message name="ServiceHeader">
 <part name="connection-ID" type="xsd:int"/>
</message>

<portType name="ServicePortType">
 <operation name="getRoutingEntrys">
  <documentation>Service definition of function nsrt__getRoutingEntrys</documentation>
  <input message="tns:getRoutingEntrysRequest"/>
  <output message="tns:getRoutingEntrysResponse"/>
 </operation>
</portType>

<binding name="ServiceBinding" type="tns:ServicePortType">
```

```
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="getRoutingEntrys">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdRoute" encodingStyle="
       http://www.w3.org/2002/12/soap-encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdRoute" encodingStyle="
       http://www.w3.org/2002/12/soap-encoding"/>
  </output>
 </operation>
</binding>

<service name="Service">
 <documentation>gSOAP 2.3 rev 2 generated service definition</documentation>
 <port name="Service" binding="tns:ServiceBinding">
  <SOAP:address location="http://localhost"/>
 </port>
</service>

</definitions>
```

Listing B.4: nsrt.wsdl

B.2.4. nssys.wsdl

Diese WSDL-Definition enthält die Definitionen des Namensbereiches `WS_mgmt_stdSystem`.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Service"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 targetNamespace="urn:WS_mgmt_stdSystem"
 xmlns:tns="urn:WS_mgmt_stdSystem"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
```

```
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent">

<types>
 <schema targetNamespace="urn:WS_mgmt_stdDefs"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:WS_mgmt_stdDefs"
  xmlns:nssys="urn:WS_mgmt_stdSystem"
  xmlns:nsif="urn:WS_mgmt_stdInterface"
  xmlns:nsrt="urn:WS_mgmt_stdRoute"
  xmlns:nsevt="urn:WS_mgmt_stdEvent"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <element name="addrType" type="ns:addrType"/>
  <simpleType name="addrType">
   <restriction base="xsd:string">
    <enumeration value="IP"/>
    <enumeration value="ETH"/>
   </restriction>
  </simpleType>

  <element name="address" type="ns:address"/>
  <complexType name="address">
   <sequence>
    <element name="addr" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
        ="true"/>
   </sequence>
   <attribute name="type" type="ns:addrType" use="required"/>
  </complexType>

 </schema>
 <schema targetNamespace="urn:WS_mgmt_stdSystem"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:WS_mgmt_stdDefs"
  xmlns:nssys="urn:WS_mgmt_stdSystem"
```

```
xmlns:nsif="urn:WS_mgmt_stdInterface"
xmlns:nsrt="urn:WS_mgmt_stdRoute"
xmlns:nsevt="urn:WS_mgmt_stdEvent"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">

<complexType name="ArrayOfinterfaceDescription">
 <complexContent>
  <restriction base="SOAP-ENC:Array">
   <sequence>
    <element name="item" type="nsif:interfaceDescription" minOccurs="0"
        maxOccurs="unbounded"/>
   </sequence>
   <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsif:
       interfaceDescription[]"/>
  </restriction>
 </complexContent>
</complexType>

<complexType name="ArrayOfinterfaceStats">
 <complexContent>
  <restriction base="SOAP-ENC:Array">
   <sequence>
    <element name="item" type="nsif:interfaceStats" minOccurs="0" maxOccurs="
        unbounded"/>
   </sequence>
   <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsif:interfaceStats
       []"/>
  </restriction>
 </complexContent>
</complexType>

<complexType name="ArrayOfroutingInformation">
 <complexContent>
  <restriction base="SOAP-ENC:Array">
   <sequence>
    <element name="item" type="nsrt:routingInformation" minOccurs="0"
        maxOccurs="unbounded"/>
   </sequence>
   <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsrt:
       routingInformation[]"/>
  </restriction>
 </complexContent>
```

```
  </complexType>

  <element name="sysInf" type="nssys:sysInf"/>
  <complexType name="sysInf">
   <sequence>
    <element name="description" type="xsd:string" minOccurs="1" maxOccurs="1"
        nillable="true"/>
    <element name="uptime" type="xsd:duration" minOccurs="0" maxOccurs="1"
        nillable="true"/>
    <element name="idletime" type="xsd:duration" minOccurs="0" maxOccurs="1"
        nillable="true"/>
   </sequence>
  </complexType>

 </schema>
 <schema targetNamespace="urn:WS_mgmt_stdInterface"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:WS_mgmt_stdDefs"
  xmlns:nssys="urn:WS_mgmt_stdSystem"
  xmlns:nsif="urn:WS_mgmt_stdInterface"
  xmlns:nsrt="urn:WS_mgmt_stdRoute"
  xmlns:nsevt="urn:WS_mgmt_stdEvent"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <element name="ifType" type="nsif:ifType"/>
  <simpleType name="ifType">
   <restriction base="xsd:string">
    <enumeration value="IF-NONE"/>
    <enumeration value="IF-OTHER"/>
    <enumeration value="IF-ETHERNET"/>
    <enumeration value="IF-LOOPBACK"/>
    <enumeration value="IF-VIRTUAL"/>
   </restriction>
  </simpleType>

  <element name="interfaceDescription" type="nsif:interfaceDescription"/>
  <complexType name="interfaceDescription">
   <sequence>
    <element name="type" type="nsif:ifType" minOccurs="1" maxOccurs="1"/>
```

```
   <element name="name" type="xsd:string" minOccurs="1" maxOccurs="1" nillable="true"/>
   <element name="mtu" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="flag" type="xsd:string" minOccurs="1" maxOccurs="unbounded" nillable="true"/>
   <element name="address" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
   <element name="netmask" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
   <element name="broadcast" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
   <element name="dst-address" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
   <element name="link-address" type="ns:address" minOccurs="1" maxOccurs="1" nillable="true"/>
   <element name="alias" type="ns:address" minOccurs="1" maxOccurs="unbounded" nillable="true"/>
  </sequence>
 </complexType>

 <element name="interfaceStats" type="nsif:interfaceStats"/>
 <complexType name="interfaceStats">
  <sequence>
   <element name="ifname" type="xsd:string" minOccurs="1" maxOccurs="1" nillable="true"/>
   <element name="rx-bytes" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-packets" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-errors" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-drop" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-fifo" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-frame" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-compressed" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="rx-multicast" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-bytes" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-packets" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-errors" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-drop" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-fifo" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-colls" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-carrier" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   <element name="tx-compressed" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  </sequence>
 </complexType>
```

```
</schema>
<schema targetNamespace="urn:WS_mgmt_stdRoute"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <element name="routingInformation" type="nsrt:routingInformation"/>
 <complexType name="routingInformation">
  <sequence>
   <element name="iface" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
       ="true"/>
   <element name="dst-addr" type="ns:address" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="gw-addr" type="ns:address" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="nm-addr" type="ns:address" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="flag" type="xsd:string" minOccurs="1" maxOccurs="unbounded"
       nillable="true"/>
   <element name="mtu" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="metric" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="refcnt" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="use" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="window" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   <element name="irtt" type="xsd:int" minOccurs="1" maxOccurs="1"/>
  </sequence>
 </complexType>

</schema>
<schema targetNamespace="urn:WS_mgmt_stdEvent"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
```

```
xmlns:nssys="urn:WS_mgmt_stdSystem"
xmlns:nsif="urn:WS_mgmt_stdInterface"
xmlns:nsrt="urn:WS_mgmt_stdRoute"
xmlns:nsevt="urn:WS_mgmt_stdEvent"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">

<element name="type" type="nsevt:type"/>
<simpleType name="type">
 <restriction base="xsd:string">
  <enumeration value="EVT-TP-SUCCESS"/>
  <enumeration value="EVT-TP-IF"/>
  <enumeration value="EVT-TP-ROUTE"/>
  <enumeration value="EVT-TP-SRVCLOSE"/>
  <enumeration value="EVT-TP-DEFAULT"/>
  <enumeration value="EVT-TP-FAULT"/>
 </restriction>
</simpleType>

<element name="regType" type="nsevt:regType"/>
<simpleType name="regType">
 <restriction base="xsd:string">
  <enumeration value="EVT-REGTYPE-NONE"/>
  <enumeration value="EVT-REGTYPE-IF"/>
  <enumeration value="EVT-REGTYPE-RT"/>
  <enumeration value="EVT-REGTYPE-ALL"/>
  <enumeration value="EVT-REGTYPE-FW"/>
 </restriction>
</simpleType>

<element name="regAction" type="nsevt:regAction"/>
<simpleType name="regAction">
 <list>
  <restriction base="xsd:string">
   <enumeration value="EVT-REGISTER"/>
   <enumeration value="EVT-UNREGISTER"/>
  </restriction>
 </list>
</simpleType>

</schema>
</types>
```

```
<message name="uptimeRequest">
</message>

<message name="uptimeResponse">
 <part name="result" type="xsd:string"/>
</message>

<message name="getSystemInformationRequest">
</message>

<message name="sysInf">
 <part name="description" type="xsd:string"/>
 <part name="uptime" type="xsd:duration"/>
 <part name="idletime" type="xsd:duration"/>
</message>

<message name="ServiceHeader">
 <part name="connection-ID" type="xsd:int"/>
</message>

<portType name="ServicePortType">
 <operation name="uptime">
  <documentation>Service definition of function nssys__uptime</documentation>
  <input message="tns:uptimeRequest"/>
  <output message="tns:uptimeResponse"/>
 </operation>
 <operation name="getSystemInformation">
  <documentation>Service definition of function nssys__getSystemInformation</documentation>
  <input message="tns:getSystemInformationRequest"/>
  <output message="tns:sysInf"/>
 </operation>
</portType>

<binding name="ServiceBinding" type="tns:ServicePortType">
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="uptime">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdSystem" encodingStyle="http://www.w3.org/2002/12/soap-encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdSystem" encodingStyle="
```

```
        http://www.w3.org/2002/12/soap−encoding"/>
  </output>
 </operation>
 <operation name="getSystemInformation">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdSystem" encodingStyle="
        http://www.w3.org/2002/12/soap−encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdSystem" encodingStyle="
        http://www.w3.org/2002/12/soap−encoding"/>
  </output>
 </operation>
</binding>

<service name="Service">
 <documentation>gSOAP 2.3 rev 2 generated service definition</documentation>
 <port name="Service" binding="tns:ServiceBinding">
  <SOAP:address location="http://localhost"/>
 </port>
</service>

</definitions>
```

Listing B.5: nssys.wsdl

B.2.5. nsevt.wsdl

Diese WSDL-Definition enthält die Definitionen des Namensbereiches `WS_mgmt_stdEvent`.

```
<?xml version="1.0" encoding="UTF−8"?>
<definitions name="Service"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 targetNamespace="urn:WS_mgmt_stdEvent"
 xmlns:tns="urn:WS_mgmt_stdEvent"
 xmlns:SOAP−ENV="http://www.w3.org/2002/12/soap−envelope"
 xmlns:SOAP−ENC="http://www.w3.org/2002/12/soap−encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
```

```
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent">

<types>
 <schema targetNamespace="urn:WS_mgmt_stdDefs"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:WS_mgmt_stdDefs"
  xmlns:nssys="urn:WS_mgmt_stdSystem"
  xmlns:nsif="urn:WS_mgmt_stdInterface"
  xmlns:nsrt="urn:WS_mgmt_stdRoute"
  xmlns:nsevt="urn:WS_mgmt_stdEvent"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <element name="addrType" type="ns:addrType"/>
  <simpleType name="addrType">
   <restriction base="xsd:string">
    <enumeration value="IP"/>
    <enumeration value="ETH"/>
   </restriction>
  </simpleType>

  <element name="address" type="ns:address"/>
  <complexType name="address">
   <sequence>
    <element name="addr" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
        ="true"/>
   </sequence>
   <attribute name="type" type="ns:addrType" use="required"/>
  </complexType>

 </schema>
 <schema targetNamespace="urn:WS_mgmt_stdSystem"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:WS_mgmt_stdDefs"
```

```
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <element name="sysInf" type="nssys:sysInf"/>
 <complexType name="sysInf">
  <sequence>
   <element name="description" type="xsd:string" minOccurs="1" maxOccurs="1"
       nillable="true"/>
   <element name="uptime" type="xsd:duration" minOccurs="0" maxOccurs="1"
       nillable="true"/>
   <element name="idletime" type="xsd:duration" minOccurs="0" maxOccurs="1"
       nillable="true"/>
  </sequence>
 </complexType>

</schema>
<schema targetNamespace="urn:WS_mgmt_stdInterface"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <element name="ifType" type="nsif:ifType"/>
 <simpleType name="ifType">
  <restriction base="xsd:string">
   <enumeration value="IF-NONE"/>
   <enumeration value="IF-OTHER"/>
   <enumeration value="IF-ETHERNET"/>
   <enumeration value="IF-LOOPBACK"/>
   <enumeration value="IF-VIRTUAL"/>
  </restriction>
```

```
</simpleType>

<element name="interfaceDescription" type="nsif:interfaceDescription"/>
<complexType name="interfaceDescription">
 <sequence>
  <element name="type" type="nsif:ifType" minOccurs="1" maxOccurs="1"/>
  <element name="name" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
      ="true"/>
  <element name="mtu" type="xsd:int" minOccurs="1" maxOccurs="1"/>
  <element name="flag" type="xsd:string" minOccurs="1" maxOccurs="unbounded"
      nillable="true"/>
  <element name="address" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="netmask" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="broadcast" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="dst-address" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="link-address" type="ns:address" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="alias" type="ns:address" minOccurs="1" maxOccurs="unbounded"
      nillable="true"/>
 </sequence>
</complexType>

<element name="interfaceStats" type="nsif:interfaceStats"/>
<complexType name="interfaceStats">
 <sequence>
  <element name="ifname" type="xsd:string" minOccurs="1" maxOccurs="1"
      nillable="true"/>
  <element name="rx-bytes" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="rx-packets" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="rx-errors" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="rx-drop" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="rx-fifo" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="rx-frame" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="rx-compressed" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="rx-multicast" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="tx-bytes" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="tx-packets" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="tx-errors" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="tx-drop" type="xsd:int" minOccurs="0" maxOccurs="1"/>
  <element name="tx-fifo" type="xsd:int" minOccurs="0" maxOccurs="1"/>
```

```
    <element name="tx-colls" type="xsd:int" minOccurs="0" maxOccurs="1"/>
    <element name="tx-carrier" type="xsd:int" minOccurs="0" maxOccurs="1"/>
    <element name="tx-compressed" type="xsd:int" minOccurs="0" maxOccurs="1"/>
   </sequence>
  </complexType>

 </schema>
 <schema targetNamespace="urn:WS_mgmt_stdRoute"
  xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
  xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:WS_mgmt_stdDefs"
  xmlns:nssys="urn:WS_mgmt_stdSystem"
  xmlns:nsif="urn:WS_mgmt_stdInterface"
  xmlns:nsrt="urn:WS_mgmt_stdRoute"
  xmlns:nsevt="urn:WS_mgmt_stdEvent"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">

  <element name="routingInformation" type="nsrt:routingInformation"/>
  <complexType name="routingInformation">
   <sequence>
    <element name="iface" type="xsd:string" minOccurs="1" maxOccurs="1" nillable
        ="true"/>
    <element name="dst-addr" type="ns:address" minOccurs="1" maxOccurs="1"
        nillable="true"/>
    <element name="gw-addr" type="ns:address" minOccurs="1" maxOccurs="1"
        nillable="true"/>
    <element name="nm-addr" type="ns:address" minOccurs="1" maxOccurs="1"
        nillable="true"/>
    <element name="flag" type="xsd:string" minOccurs="1" maxOccurs="unbounded"
        nillable="true"/>
    <element name="mtu" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="metric" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="refcnt" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="use" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="window" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="irtt" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   </sequence>
  </complexType>

 </schema>
```

```
<schema targetNamespace="urn:WS_mgmt_stdEvent"
 xmlns:SOAP-ENV="http://www.w3.org/2002/12/soap-envelope"
 xmlns:SOAP-ENC="http://www.w3.org/2002/12/soap-encoding"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ns="urn:WS_mgmt_stdDefs"
 xmlns:nssys="urn:WS_mgmt_stdSystem"
 xmlns:nsif="urn:WS_mgmt_stdInterface"
 xmlns:nsrt="urn:WS_mgmt_stdRoute"
 xmlns:nsevt="urn:WS_mgmt_stdEvent"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

 <element name="type" type="nsevt:type"/>
 <simpleType name="type">
  <restriction base="xsd:string">
   <enumeration value="EVT-TP-SUCCESS"/>
   <enumeration value="EVT-TP-IF"/>
   <enumeration value="EVT-TP-ROUTE"/>
   <enumeration value="EVT-TP-SRVCLOSE"/>
   <enumeration value="EVT-TP-DEFAULT"/>
   <enumeration value="EVT-TP-FAULT"/>
  </restriction>
 </simpleType>

 <element name="regType" type="nsevt:regType"/>
 <simpleType name="regType">
  <restriction base="xsd:string">
   <enumeration value="EVT-REGTYPE-NONE"/>
   <enumeration value="EVT-REGTYPE-IF"/>
   <enumeration value="EVT-REGTYPE-RT"/>
   <enumeration value="EVT-REGTYPE-ALL"/>
   <enumeration value="EVT-REGTYPE-FW"/>
  </restriction>
 </simpleType>

 <element name="regAction" type="nsevt:regAction"/>
 <simpleType name="regAction">
  <list>
   <restriction base="xsd:string">
    <enumeration value="EVT-REGISTER"/>
    <enumeration value="EVT-UNREGISTER"/>
   </restriction>
```

```
   </list>
  </simpleType>

  <complexType name="ArrayOfinterfaceDescription">
   <complexContent>
    <restriction base="SOAP-ENC:Array">
     <sequence>
      <element name="item" type="nsif:interfaceDescription" minOccurs="0"
          maxOccurs="unbounded"/>
     </sequence>
     <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsif:
         interfaceDescription[]"/>
    </restriction>
   </complexContent>
  </complexType>

  <complexType name="ArrayOfinterfaceStats">
   <complexContent>
    <restriction base="SOAP-ENC:Array">
     <sequence>
      <element name="item" type="nsif:interfaceStats" minOccurs="0" maxOccurs="
          unbounded"/>
     </sequence>
     <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsif:interfaceStats
         []"/>
    </restriction>
   </complexContent>
  </complexType>

  <complexType name="ArrayOfroutingInformation">
   <complexContent>
    <restriction base="SOAP-ENC:Array">
     <sequence>
      <element name="item" type="nsrt:routingInformation" minOccurs="0"
          maxOccurs="unbounded"/>
     </sequence>
     <attribute ref="SOAP-ENC:arrayType" WSDL:arrayType="nsrt:
         routingInformation[]"/>
    </restriction>
   </complexContent>
  </complexType>

 </schema>
</types>
```

```
<message name="registerEventHandlerRequest">
 <part name="portnumber" type="xsd:int"/>
 <part name="hostname" type="xsd:string"/>
 <part name="regact" type="nsevt:regAction"/>
 <part name="tp" type="nsevt:regType"/>
</message>

<message name="registerEventHandlerResponse">
 <part name="result" type="xsd:int"/>
</message>

<message name="signalEventRequest">
 <part name="error" type="xsd:string"/>
 <part name="msgtype" type="nsevt:type"/>
</message>

<message name="signalEventResponse">
 <part name="deliverySuccess" type="nsevt:type"/>
</message>

<message name="ServiceHeader">
 <part name="connection-ID" type="xsd:int"/>
</message>

<portType name="ServicePortType">
 <operation name="registerEventHandler">
  <documentation>Service definition of function nsevt__registerEventHandler</documentation>
  <input message="tns:registerEventHandlerRequest"/>
  <output message="tns:registerEventHandlerResponse"/>
 </operation>
 <operation name="signalEvent">
  <documentation>Service definition of function nsevt__signalEvent</documentation>
  <input message="tns:signalEventRequest"/>
  <output message="tns:signalEventResponse"/>
 </operation>
</portType>

<binding name="ServiceBinding" type="tns:ServicePortType">
 <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="registerEventHandler">
  <SOAP:operation soapAction=""/>
```

```
  <input>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdEvent" encodingStyle="
       http://www.w3.org/2002/12/soap-encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdEvent" encodingStyle="
       http://www.w3.org/2002/12/soap-encoding"/>
  </output>
 </operation>
 <operation name="signalEvent">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdEvent" encodingStyle="
       http://www.w3.org/2002/12/soap-encoding"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:WS_mgmt_stdEvent" encodingStyle="
       http://www.w3.org/2002/12/soap-encoding"/>
  </output>
 </operation>
</binding>

<service name="Service">
 <documentation>gSOAP 2.3 rev 2 generated service definition</documentation>
 <port name="Service" binding="tns:ServiceBinding">
  <SOAP:address location="http://localhost"/>
 </port>
</service>

</definitions>
```

Listing B.6: nsevt.wsdl

C. Defininition der SMIv2

```
RFC1065-SMI DEFINITIONS ::= BEGIN

EXPORTS -- EVERYTHING
        internet, directory, mgmt,
        experimental, private, enterprises,
        OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
        ApplicationSyntax, NetworkAddress, IpAddress,
        Counter, Gauge, TimeTicks, Opaque;

 -- the path to the root

 internet     OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }

 directory    OBJECT IDENTIFIER ::= { internet 1 }

 mgmt         OBJECT IDENTIFIER ::= { internet 2 }

 experimental OBJECT IDENTIFIER ::= { internet 3 }

 private      OBJECT IDENTIFIER ::= { internet 4 }
 enterprises  OBJECT IDENTIFIER ::= { private 1 }

 -- definition of object types

 OBJECT-TYPE MACRO ::=
 BEGIN
     TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                       "ACCESS" Access
                       "STATUS" Status
     VALUE NOTATION ::= value (VALUE ObjectName)

     Access ::= "read-only"
                  | "read-write"
```

```
                  | "write-only"
                  | "not-accessible"
    Status ::= "mandatory"
                  | "optional"
                  | "obsolete"
END

   -- names of objects in the MIB

   ObjectName ::=
       OBJECT IDENTIFIER

   -- syntax of objects in the MIB

   ObjectSyntax ::=
       CHOICE {
           simple
               SimpleSyntax,

   -- note that simple SEQUENCEs are not directly
   -- mentioned here to keep things simple (i.e.,
   -- prevent mis-use). However, application-wide
   -- types which are IMPLICITly encoded simple
   -- SEQUENCEs may appear in the following CHOICE

             application-wide
                 ApplicationSyntax
         }

     SimpleSyntax ::=
         CHOICE {
             number
                 INTEGER,

             string
                 OCTET STRING,

             object
                 OBJECT IDENTIFIER,

             empty
                 NULL
         }
```

```
ApplicationSyntax ::=
    CHOICE {
        address
            NetworkAddress,

        counter
            Counter,

        gauge
            Gauge,

        ticks
            TimeTicks,

        arbitrary
            Opaque

-- other application-wide types, as they are
-- defined, will be added here
    }

-- application-wide types

NetworkAddress ::=
    CHOICE {
        internet
            IpAddress
    }

IpAddress ::=
    [APPLICATION 0]         -- in network-byte order
        IMPLICIT OCTET STRING (SIZE (4))

Counter ::=
    [APPLICATION 1]
        IMPLICIT INTEGER (0..4294967295)

Gauge ::=
    [APPLICATION 2]
        IMPLICIT INTEGER (0..4294967295)

TimeTicks ::=
    [APPLICATION 3]
        IMPLICIT INTEGER
```

```
    Opaque ::=
        [APPLICATION 4]        -- arbitrary ASN.1 value,
            IMPLICIT OCTET STRING -- "double-wrapped"

    END
```

Listing C.1: SMI Definition

Abkürzungsverzeichnis

ACL - Access Control List

ASN.1 - Abstract Syntax Notation 1

BER - Basic Encoding Rules

CBC - Cipher Block Chaining

CMIP - Common Management Information Protocol

DES - Data Encryption Standard

DoS - Denial of Service

DTD - Document Type Definition

HEMS - High-Level Entity Management System

IAB - Intenet Activities Board

ICMP - Internet Control Message Protocol

ISO - International Organization for Standardisation

MIB - Management Information Base

OID - Object Identifier

OMG - Object Management Group

OSI - Open Systems Interconnect

Ping - Paket Internet Grouper

SGMP - Simple Gateway Monitoring Protocol

SMI - Structure of Management Information

SNMP - Simple Network Management Protocol

SOAP - Simple Object Access Protocol

Tcl - Tool Command Language

TTL - Time To Live

UDP - Universal Datagram Protcol

USM - User Security Model, SNMPv3

VACM - View Based Access Control Model

WSDL - Webservice Description Language