INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Untersuchung der Portierbarkeit von Managementanwendungen für PC-Management-Plattformen

Bernhard Foltin

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Stephen Heilbronner
	Dr. Bernhard Neumair
	Manfred Randelzofer
Abgabedatum:	15. Februar 1996

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Einführung

Nach [Heg93] umfaßt das Netz- und Systemmanagement die Gesamtheit der Vorkehrungen und Aktivitäten zur Sicherstellung des effektiven und effizienten Einsatzes eines Rechnernetzes bzw. eines verteilten Systems.
Die zunehmende Vernetzung von Rechnern, das Entstehen neuer und größerer Kommunikationsnetze, Netzverbunde und verteilter Systeme, die aufgrund der immer leistungsfähigeren Kommunikationstechniken immer höheren Anforderungen an Rechnernetze haben auch die Anforderungen an ein adäquates Management erhöht [Heg93]:

- Mit der Größe insbesondere der Netzverbunde wächst i.a. auch die Heterogenität der Netze. Das bedeutet, daß immer mehr Netzkomponenten und Systeme der unterschiedlichsten Hersteller gleichzeitig verwaltet werden müssen.

- Immer größer werdende Netze erfordern eine *zentrale* Verwaltung, da die Lokalisierung und Behebung von Fehlern jeweils vor Ort mit einem immer höheren (Personal-)Aufwand verbunden ist [Zen93].

- Neu hinzukommende Aufgabenbereiche sowie neue oder zusätzliche Aufgaben innerhalb bestehender Bereiche verändern auch die entsprechenden Managementaufgaben. Beispiel Anwendungsmanagement: umfaßte Netzmanagement anfangs ausschließlich (Netz-)Komponentenmanagement, so wurde später deutlich, daß auch Anwendungen wie z.B. E-Mail Managementaufgaben beinhalten [Heg93]. Ein Beispiel dafür, wie sich auch die Anforderungen in einem bestehenden Aufgabenbereich ändern können, ist die Datenhaltung. So wollen Benutzer heute über ein verteiltes Dateisystem von überall auf ihre Daten zugreifen können [Heg93].

Eine Antwort auf diese veränderten und sich verändernden Managementanforderungen ist das *integrierte Management*. Integriertes Management bedeutet nach

[Heg93] u.a.:

- Die integrierten Managementwerkzeuge bieten eine ausreichende Funktionalität d.h., es gibt keine Einschränkung auf bestimmte Aufgabenbereiche oder Produkte bestimmter Hersteller.

- Die Managementfunktionen selbst verwenden einheitliche, herstellerunabhängige Programmierschnittstellen und Bedienoberflächen. Dies bedeutet auch, daß die von einem heterogenen Netz- und Systemumfeld gelieferten Informationen herstellerunabhängig interpretiert werden können müssen. (Eine Architektur für integriertes Management muß dementsprechend ein Informations- und ein Kommunikationsmodell bereitstellen.)

Ein Teilproblem des integrierten Management ist also insbesondere, eine ausreichende Managementfunktionalität bereitzustellen. Die Managementfunktionalität konventioneller Managementwerkzeuge ist dabei i.a. nur unzureichend, da meistens nur ganz bestimmte Anwendungsbereiche und/oder nur Produkte bestimmter Hersteller unterstützt werden. Darüberhinaus ist eine Erweiterung der Managementfunktionalität entweder gar nicht oder nur über herstellerspezifische Schnittstellen und damit auch nur eingeschränkt möglich [Heg93].
Das Konzept der Managementplattformen schafft nun im Bereich der Managementfunktionalität gezielt die Voraussetzungen für integriertes Management in heterogenen Netz- und Systemumgebungen. Die Managementplattformen selbst fungieren dabei gewissermaßen als eine Art Trägersystem des integrierten Managementsystems. Auf den Plattformen ist jeweils nur eine beschränkte Palette von Basisanwendungen implementiert. Im Gegensatz zu den konventionellen Werkzeugen stellen die Plattformen jedoch (im Idealfall) standardisierte Schnittstellen zur Verfügung. Anwendungsprogrammierer haben so die Möglichkeit, das Basismanagement einer Plattform entsprechend den Bedürfnissen des Netzbetreibers zu erweitern. Die Erweiterungen sind dabei nicht auf das Management von Ressourcen bestimmter Hersteller oder auf bestimmte Funktionsbereiche beschränkt. Darüberhinaus enthält eine Plattform Werkzeuge für die Integration dieser zusätzlichen Anwendungen in das Managementsystem. [Heg93]

1.2 Aufgabenstellung

Eine solche plattformbasierte Managementanwendung ist beispielsweise der SNI Server Manager, der auf der PC-Managementplattform Network Management System (NMS) von Novell aufsetzt.
Die ursprüngliche Motivation der Themenstellung lag in der Absicht von SNI, den Server Manager evtl. auf zwei andere PC-Management-Plattformen - HP Open-View for Windows (HPOV) und Systems Management Server (SMS) von Microsoft - zu portieren. SNI stand also vor dem Problem, zwei weitere, funktionell

identische, aber auf diese beiden Plattformen zugeschnittene Versionen des SNI Server Managers entwickeln zu müssen. Vor einem ähnlichen Portierungsproblem stehen auch andere Hersteller von Anwendungen für Management-Plattformen.
Im Rahmen der Diplomarbeit werden die drei schon erwähnten Management-Plattformen betrachtet: NetWare Management System (NMS) Version 2.0 von Novell, HP OpenView for Windows (HPOV) Version 7.2 und Systems Management Server (SMS) Version 1.0 von Microsoft. Ziel ist die Entwicklung eines Software-Design-Konzepts, dessen Umsetzung die Portierbarkeit beliebiger Managementanwendungen auf diesen Plattformen gewährleistet. Im folgenden wird dieses Software-Design-Konzept jeweils kurz als *Portabilitätskonzept* bezeichnet.

1.3 Überblick

Kapitel 2 beschreibt zunächst allgemein die Architektur von Management-Plattformen und geht dann auf der Grundlage dieser Architektur näher auf die zu untersuchenden, konkreten Plattformen ein.
Das folgende Kapitel geht etwas näher auf die Entwicklungsumgebungen der Plattformen ein; insbesondere werden die Werkzeuge zur Integration der Anwendungen vorgestellt.
Das vierte Kapitel stellt den SNI Server Manager exemplarisch als eine typische plattformbasierte Managementanwendung vor. Neben der Architektur werden dabei auch die Schnittstellen zur Plattform (NMS) und die Integration der Anwendung in NMS beschrieben.
Ausgangspunkt für die Entwicklung des Portabilitätskonzepts ist die im fünften Kapitel vorgestellte allgemeine Lösungskonzeption; darüberhinaus werden bereits auf der Basis dieser allgemeinen Konzeption erste, allgemeine Lösungsschritte zu deren Umsetzung abgeleitet.
Die zwei folgenden Kapitel 6 und 7 beschäftigen sich mit grundsätzlichen Problemstellungen, die im Rahmen der Umsetzung der Lösungskonzeption auftauchen, z.B. dem Problem der Datenbankanbindung für portierbare Anwendungen. Dabei wird versucht, die Lösung dieser Probleme jeweils systematisch anzugehen und dementsprechend auch alternative Lösungswege aufzuzeigen; soweit möglich werden dabei auch grundsätzliche Designentscheidungen getroffen und jeweils begründet.
In den folgenden Kapiteln (Kapitel 8, 9, 10, 11, 12, 13) werden die allgemeine Konzeption und die allgemeinen Designkonzepte bei der Entwicklung von konkreten (Teil-)Portabilitätskonzepten für die verschiedenen funktionalen Bereiche der Plattformen und der dort angebotenen Schnittstellen umgesetzt. Diese Kapitel behandeln im einzelnen die Schnittstellen für die SNMP-Kommunikation, zu den Bereichen Alarm-, Topologie-, Konfigurationsmanagement, der grafischen Benutzeroberfläche und der Schnittstelle zum Betriebssystem.
Kapitel 14 gibt eine kurze Zusammenfassung der wichtigsten Ergebnisse und ver-

sucht Möglichkeiten aufzuzeigen, wie die Portabilität von Managementanwendungen auf PC-Management-Plattformen verbessert werden kann.

Kapitel 2

Aufbau der Management-Plattformen

Unter Architektur versteht man im Bereich der Datenverarbeitung das funktionelle Erscheinungsbild eines Systems für den Anwender. [BRO87]
In diesem Kapitel soll zunächst die *allgemeine* Architektur von Management-Plattformen kurz vorgestellt werden; die Darstellung nimmt dabei Bezug auf den logisch-funktionalen Aufbau wie er in [Heg93] beschrieben wird.
Danach werden unter Bezugnahme auf die *allgemeinen* Architektur die *konkreten* Implementierungen der einzelnen PC-Management-Plattformen vorgestellt.
In diesem Zusammenhang muß darauf hingewiesen werden, daß die allgemeine Architektur *keine* standardisierte Architektur ist. Sie ergibt sich zum einen aus allgemeinen Anforderungen an Managementarchitekturen wie sie z.B. in der OSF DME (Open Software Foundation, Distributed Management Environment) festgelegt werden, zum anderen durch Abstraktion von bereits implementierter Plattformarchitekturen (z.B. welche Basisanwendungen sind i.a. implementiert, vgl. 2.1.2). Die allgemeinen Architekturmerkmale haben aus diesem Grund oft keine exakte Entsprechung auf den Plattformen.

2.1 Architektur von Management-Plattformen

Der funktionale Aufbau von Plattformen wird durch folgende Architektur beschrieben:

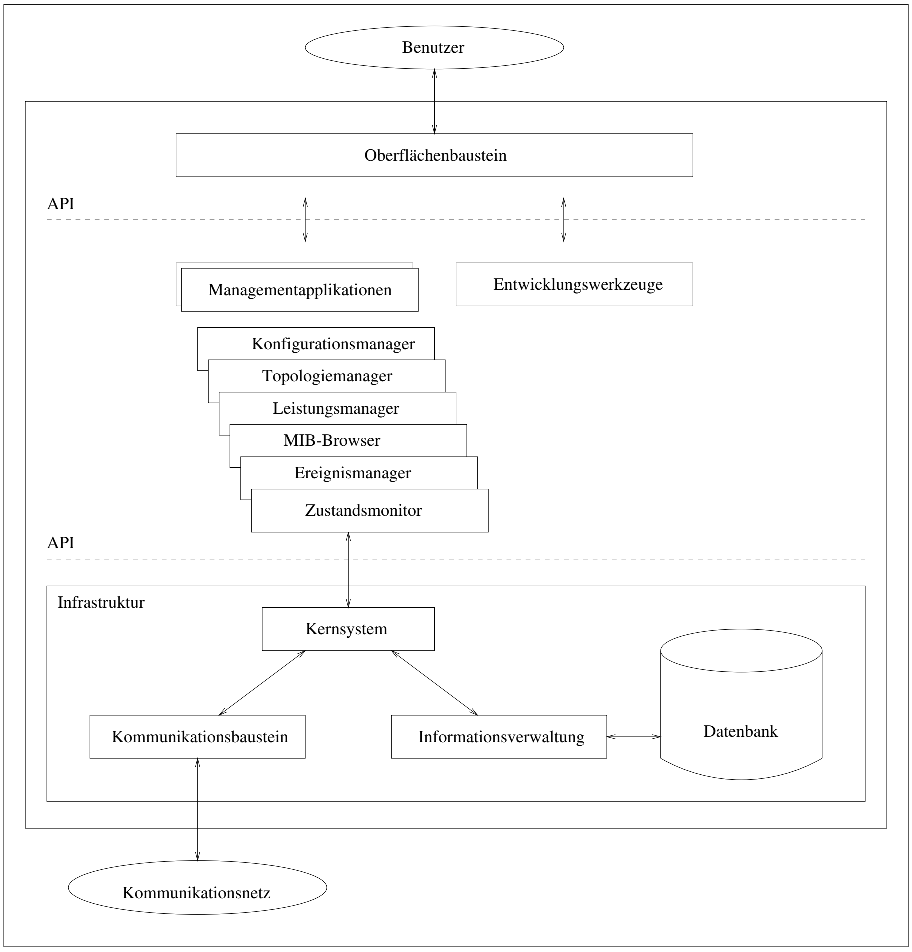

Abbildung 2.1: Funktionaler Aufbau einer Plattform nach [Heg93]

Die einzelnen Bausteine der Architektur - die sich ihrerseits wiederum in einen Funktionsteil und einen Konfigurationsteil unterteilen lassen - können dabei folgendermaßen gruppiert werden:

- Die *Infrastruktur* stellt verschiedene Grunddienste und Funktionen bereit, die von allen Managementanwendungen benötigt werden.

- Die Gruppe der *Basisanwendungen* enthält solche Managementanwendungen, die größtenteils auf den meisten Plattformen implementiert sind.

- Über den *Oberflächenbaustein* erhält der Benutzer Zugang zu den Managementapplikationen sowie eine graphische Darstellung des Netzes.

- *Entwicklungswerkzeuge* erlauben die Erweiterung der bestehenden Plattform; ein Beispiel für ein solches Entwicklungswerkzeug ist ein MIB-Compiler mit dem sich zusätzliche Managementinformationen integrieren lassen.

2.1.1 Die Infrastruktur einer Management-Plattform

Die Infrastruktur einer Plattform besteht aus dem Kernsystem, dem Kommunikationsbaustein und dem Informationsbaustein mit daran angeschlossener Datenbank.
Das *Kernsystem* koordiniert die einzelnen Bausteine und sorgt für die Kommunikation der Bausteine untereinander.
Der *Kommunikationsbaustein* stellt die Dienste bereit, mit deren Hilfe die Plattform mit Fremdsystemen, also Ressourcen (Managed Objects im Sinne des zugrundeliegenden Informationsmodells) und anderen Managementstationen kommunizieren kann. Der Kommunikationsbaustein wiederum kann als in einzelne Bausteine unterteilt gedacht werden; jeder dieser Bausteine implementiert ein Managementprotokoll, das auf einem dazu passenden Protokollstack aufsetzt. Idealerweise stellt der Kommunikationsbaustein seine Dienste den Anwendungen über eine protokollunabhängige Schnittstelle zur Verfügung. Das benötigte Managementprotokoll wählt dann ein *Kommunikationsmanager* auf der Basis einer Konfigurationstabelle oder eines entsprechenden Objektattributs aus.
Der *Informationsbaustein* implementiert das objektorientierte Informationsmodell (i.e. alle Konzepte, die zur Beschreibung der relevanten Managementinformationen verwendet werden) und stellt Dienste für das Kreieren und Verwalten von Managementinformationen bereit.

2.1.2 Die Basisanwendungen einer Management-Plattform

Zu den Basisanwendungen, die meistens auf den Plattformen implementiert sind, gehören der Konfigurationsmanager, der Topologiemanager, der Leistungsmonitor, der MIB-Browser, der Ereignismanager und der Zustandsmonitor.

Der Ereignismanager

Der Ereignismanager oder - entsprechend der NMS-/HPOV-Terminologie - der Alarmmanager gehört mit zu den wichtigsten Basisanwendungen einer Managementplattform. Ereignisse (oder Alarme) sind spontane Meldungen, die von Ressourcen an die Plattform gesendet werden; dabei wird zwischen *internen* Ereignissen, die von anderen Anwendungen der Plattform erzeugt werden, und *externen* Ereignissen, die von Ressourcen oder anderen Managementstationen erzeugt werden, unterschiedden. Aufgabe des Ereignismanagements ist es, diese Ereignisse zu empfangen und zu verarbeiten. Folgende Teilfunktionen können dabei implementiert sein:

- Abspeichern der Ereignisse in Logfiles, die vom Benutzer ausgewertet werden können.

- Änderung der Komponentenzustände in Abhängigkeit von den Ereignissen: die Beschreibung der Komponenten innerhalb des verwendeten (objektorientierten) Informationsmodell enthält Attribute, die den Zustand dieser Komponente - und Zustand meint hier die Funktionsfähigkeit - beschreiben. Der Wert dieses Zustandattributs wird entsprechend dem zugrundegelegten und evtl. frei konfigurierbaren Zustandsmodell geändert; das Zustandsmodell beschreibt die möglichen Zustände und die zustandsändernden Ereignisse.

- Melden von Ereignissen an den Benutzer über optische oder akustische Signale. Eine Rolle spielt in diesem Zusammenhang auch das Alarmforwarding, also wie Alarme einer niedriger eingeordneten Submap an eine darüberliegenden Submaps weitergeleitet werden, die der Betreiber möglicherweise gerade göffnet hat (vgl 2.1.3).

- Verteilen von Ereignissen an andere Managementanwendungen wie z.B. den Zustandsmonitor.

- Korrelierung von Ereignissen verschiedener Systeme unter Verwendung von Topologieinformationen: Netzwerkfehler ziehen im allgemeinen eine Flut von Ereignismeldungen an den Benutzer nach sich, da sich ein Fehler im Netz ausbreitet. Fällt beispielsweise in einem sternförmig aufgebauten Netz eine Komponente aus, so sind auch alle hinter dieser Komponente liegenden Systeme nicht mehr erreichbar und entsprechend viele Ereignisse werden gemeldet. Effizientes Ereignismanagement korreliert derartige Ereignisse auf der Basis vorhandener Topologieinformationen und zeigt in diesem Fall nur die der Managementstation am nächsten liegende Komponente als fehlerhaft an. Voraussetzung dafür ist allerdings die Unterstützung eines *netzorientierten* Informationsmodells, das das Netz nicht als Menge isolierter Systeme nachbildet (systemorientiertes Informationsmodell), sondern Beziehungen zwischen den Informationen über die verschiedenen Komponenten herstellt.

- Starten von Programmen: als Reaktion auf bestimmte Ereignisse können bei entsprechender Konfiguration beispielsweise Pager aufgerufen, Mails gesendet oder spezifische Diagnoseprogramme gestartet werden.

Die Ausführung der verschiedenen Teilfunktionen ist abhängig von der individuellen Konfiguration eines Ereignisses.

Der Topologiemanager

Topologiemanager lassen sich entsprechend der Funktionalität, die sie anbieten, in zwei Kategorien einteilen:

- *Discovery-Funktionen*:
  Discovery-Funktionen sammeln unter Verwendung der vorhandenen Managementprotokolle Informationen über die Konfiguration des Netzes und seiner Ressourcen und speichern diese in der Datenbank der Plattform ab.

- *Autotopology-Funktionen*:
  Eine Discovery-Funktion, die aus den gewonnenen Informationen automatisch zusätzlich auch die Netztopologie des überwachten Netzes aufbaut, wird als Autotopology-Funktion bezeichnet.

Beide Funktionen können entweder permanent als Hintergrundprozeß laufen oder jeweils auf Benutzeranforderungen hin gestartet werden.

Der Zustandsmonitor

Über den Zustandsmonitor der Plattform erhält der Benutzer möglichst aktuelle Informationen über Zustand und Dienstqualität der Ressourcen. Der Zustandsmonitor selbst sammelt diese Daten durch Polling der einzelnen Systeme. Zustandsinformationen werden dabei unter Verwendung von einfachen Echo- und Testprotokollen (z.B. ICMP) sowie verbindunglosen bzw. verbindungsorientierten Managementprotokollen (z.B. SNMP bzw. CMIP,CMOT) gewonnen. Die Zuverlässigkeit der Zustandsinformationen, die u.a. aufgrund verlorener oder verspäteter Antworten fehlerhaft sein können, kann dabei durch eine entsprechende Konfigurierung des Überwachungsvorganges erhöht werden: Timeout-Intervalle legen dabei fest, wie lange die Managementstation auf die Antwort eines Systems wartet, und Retry-Zähler bestimmen, wie oft dieselbe Anfrage an ein System geschickt wird, bevor die Verfügbarkeit dieses Systems bewertet wird.
Neben der Verfügbarkeit eines Systems überwacht der Zustandsmonitor auch dessen aktuelle Dienstqualität. Dazu wird dem Betreiber die Möglichkeit gegeben, Schwellwerte (Thresholds) zu definieren, die dann in festgelegten zeitlichen Abständen mit den entsprechenden aktuellen Attributwerten verglichen werden. Die Überwachung dieser Schwellwerte erfolgt in der Managementstation oder auf den überwachten Objekten. Bei Unter- bzw. Überschreitung eines Schwellwertes wird ein entsprechendes Ereignis an das Ereignismanagement geschickt.

Der MIB-Browser

Der MIB-Browser ermöglicht dem Benutzer alle von den Netzkomponenten und Endsystemen bereitgestellten Managementinformationen aktuell abzufragen.

Der Leistungsmonitor

Der Leistungsmonitor dient der langfristigen Sammlung von Leistungsdaten, sowie deren Auswertung und Anzeige. Die für eine Messung notwendigen Parameter wie Meßpunkt (Attribute eines Systems), Meßintervall und Meßdauer werden dabei vom Benutzer festgelegt. Für die Auswertung und Anzeige stehen meist einfache Hilfswerkzeuge zur Darstellung in graphische Kurven, Tabellen etc. zur Verfügung.

Der Konfigurationsmanager

Der Konfigurationsmanager verwaltet die Konfigurationsdaten der Ressourcen und erlaubt neben dem lesenden auch den schreibenden Zugriff auf diese Daten.

2.1.3 Der Oberflächenbaustein einer Management-Plattform

Eine der wichtigsten Aufgaben des Oberflächenbausteins ist die graphische Darstellung der Netztopologie: die Darstellung geschieht dabei mit Hilfe von graphischen Symbolen, die den darzustellenden Objekten (des Informationsmodells) innerhalb des Bausteins zugeordnet werden. Eine *Map* repräsentiert dem Benutzer das gesamte Netz als Hierarchie miteinander verknüpfter *Submaps*. Die Submaps ihrerseits repräsentieren mit Symbolen jeweils einen bestimmten Teil der Netztopologie.
Zusätzlich erlaubt die Plattform dem Benutzer die graphische Darstellung der Netztopologie realitätsnah zu gestalten: Submaps können mit entsprechenden Bildinformationen (Landkarten, Gebäudegrundrisse) unterlegt werden, ebenso Systeme (Gehäuse, Einschübe) oder Kommunikationsverbindungen (Leitungsverlauf, Kabeltyp).
Jede Management-Plattform stellt verschiedene Funktionen für den Umgang mit Maps und Submaps zur Verfügung:

- Mapfunktionen, die die Verwaltung (Erzeugen, Löschen, Ändern der Zugriffsrechte) von Maps und der dazugehörigen Objekte unterstützen. Dies beinhaltet auch die Möglichkeit verschiedener Views eines Netzes darzustellen beispielsweise nur dessen IP-basierte Netze.

- Editierfunktionen, die die Manipulation von Mapelementen erlauben wie z.B. das Löschen, Erzeugen, Einhängen von Submaps und Symbolen. Darüberhinaus können Zugriffsrechte und Attribute von Submaps und Symbolen geändert werden.

- Suchfunktionen helfen dem Benutzer bei der Suche nach Informationen. Das Ergebnis einer Anfrage wird entweder als Liste oder durch Markierung der entsprechenden Elemente in der Map angezeigt.

- Navigationsfunktionen erleichtern dem Benutzer die Bewegung innerhalb der Submap-Hierarchie. Dazu wird zur aktuell angezeigten Submap auch deren Stellung innerhalb der Submap-Hierarchie angezeigt.

2.2 Novell NetWare Management System

Die Darstellungen dieses Kapitels bauen inhaltlich auf [Nov93d] und [Nov93e] auf.
Das NetWare Management System 2.0 von Novell ist als verteilte Management-Plattform konzipiert: die Discovery-Funktion ist nicht auf der eigentlichen Plattform - der *NMS Konsole* - implementiert, sondern separat auf einem bestimmten NetWare Server, dem sogenannten *NMS Server*. Die Topologieinformationen werden dementsprechend vom NMS Server gesammelt und erst anschließend an die NMS Konsole zur Verarbeitung weitergeleitet. Die NMS Konsole selbst besteht aus mehreren MS Windows-Anwendungen, die die Funktionalität der Plattform implementieren.

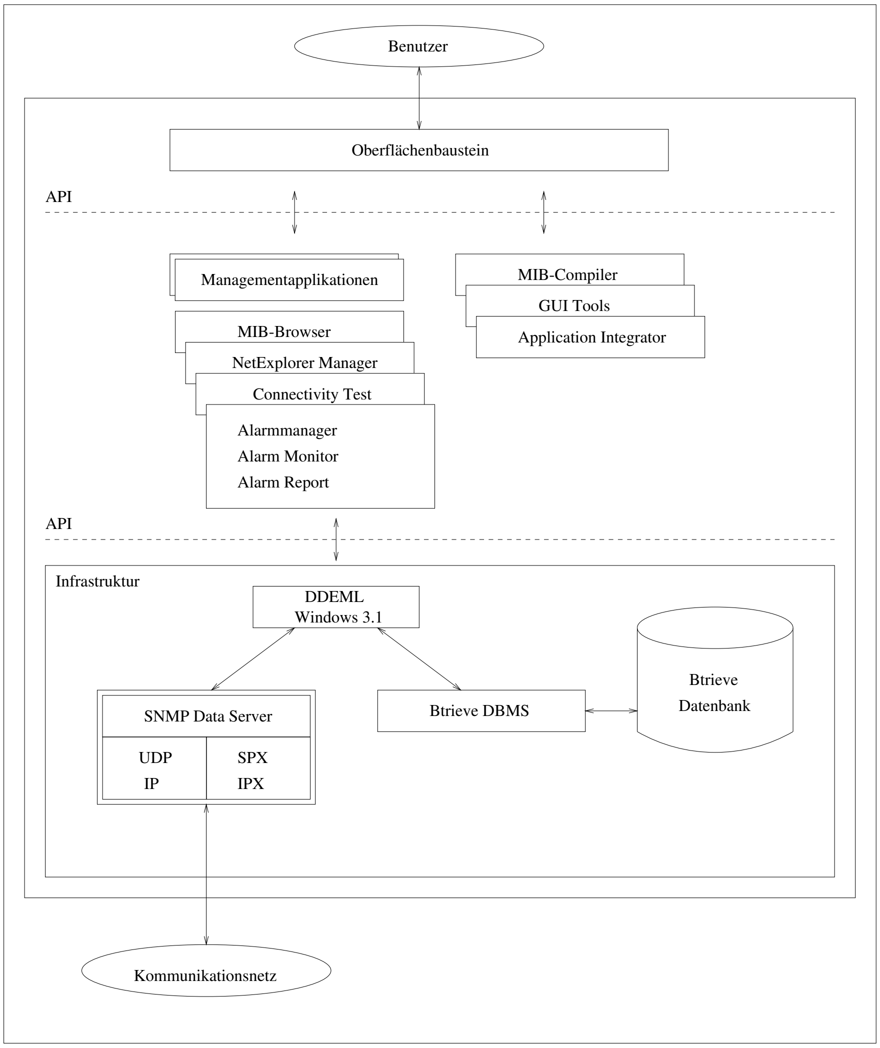

Abbildung 2.2: Funktionaler Aufbau von NMS

2.2.1 Infrastruktur

Die Windows-Anwendungen von NMS kommunizieren untereinander über DDE (Dynamic Data Exchange); dementsprechend übernimmt die DDEML (DDE Management Library) von Windows (streng genommen Windows selbst) die Aufgaben des Kernsystems. Der Kommunikationsbaustein von NMS besteht aus einem IPX/SPX- und einem UDP/IP-Protokollstack. Als Managementprotokoll unterstützt NMS SNMP; dabei liegt der Hauptanteil für die Durchführung der SNMP-Kommunikation bei den Anwendungen. Die Anwendungen übergeben dem *SNMP Data Server* von NMS die - bereits nach den BER (Basic Encoding Rules, vgl. Abschnitt 8.1) kodierten - SNMP-PDUs und den SNMP-Header (community, version). Der Data Server setzt diese Teile lediglich zu einer SNMP-Nachricht zusammen und gibt sie an die Transportschicht weiter.(vgl. 8.2) Der Data Server benutzt dabei sowohl den UDP/IP- als auch den IPX/SPX-Protokollstack (vgl. auch [RFC93b]).
Die Managementinformationen werden in einer Btrieve-Datenbank abgespeichert, deren Tabellen ein objektorientierte Informationsmodell unterstützen (vgl. [Nov93c]).

2.2.2 Basisanwendungen

NMS implementiert vier der sechs oben besprochenen Basisanwendungen:

- die *Connectivity Test* Anwendung fungiert als Zustandsmonitor: die Erreichbarkeit von Systemen wird in regelmäßigen Abständen (konfigurierbar) mit Echo-Requests überprüft; für Echo-Requests werden ICMP bei IP-Komponenten und IPX Echo Pakete für IPX-Komponenten eingesetzt.

- Der *SNMP MIB Browser* wird für den direkten Zugriff auf Managementinformationen verwendet. Die Ergebnisse von SNMP-Anfragen werden tabellarisch oder graphisch (bei gepollten SNMP-Informationen) angezeigt.

- Das Topologiemanagement unter NMS ist als Autotopologieanwendung implementiert: der *NetExplorer Manager* sammelt im Zusammenspiel mit dem *NetExplorer* des NMS Servers die Topologiedaten, während eine graphische Anwendung daraus die Netztopologie aufbaut. Der NetExplorer Manager kann dabei ständig als Hintergrundprozess laufen oder periodisch gestartet werden (konfigurierbar).

- Das Ereignismanagement umfaßt drei Anwendungen: den *Alarmmanager*, den *Alarm Monitor* und den *Alarm Report*. Der Alarmmanager empfängt externe Alarme vom SNMP Data Server, sowie interne Alarme vom NetExplorer Manager und der Connectivity Test Anwendung. Jeder dieser Alarme

ist konfigurierbar und je nach Konfiguration wird der Status des betroffenen Objekts geändert und entsprechend farblich angezeigt, ein akkustisches (Beep) oder optisches (Ticker Tape, bell icon) Signal generiert, ein Programm gestartet und/oder der Alarm in der Btrieve-Datenbank abgespeichert. Darüberhinaus werden Alarme bei entsprechend konfigurierter Dringlichkeitsstufe an den Oberflächenbaustein weitergeleitet und dem Benutzer über eigene Alarmicons an den betroffenen Objekten angezeigt.
Der Alarmmanager leitet außerdem alle Alarme an den Alarm Monitor weiter, der auf Anforderung dem Benutzer die letzten 400 Alarme anzeigt. Die vom Alarm Monitor gespeicherten Alarme werden bei Sitzungsende gelöscht.
Die vom Alarmmanager in der Btrieve-Datenbank abgespeicherten Alarme bleiben bis zur expliziten Löschung durch den Betreiber gespeichert und werden über den Alarm Report angezeigt.

2.2.3 Oberflächenbaustein

Der Oberflächenbaustein stellt dem Benutzer automatisch die Netztopologie über eine sogenannte *Internet Map* und die hierarchisch darunterliegenden *Segment Maps* dar. Außerdem wird dem Verwalter die Möglichkeit geboten, realitätsnahe Darstellungen des Netzes - sogenannte *locational maps* - selbst zu gestalten.

2.2.4 Entwicklungswerkzeuge

Neben einem MIB-Compiler enthält die Plattform einen sogenannten *Application Integrator* zur Integration von zusätzlichen Managementanwendungen in die Plattform. Darüberhinaus kann der Anwendungsprogrammierer verschiedene graphische Tools zur Informationsdarstellung verwenden.

2.3 HP OpenView for Windows

Ebenso wie NMS ist HP OpenView for Windows Workgroup Node Manager, Version 7.2, eine Management-Plattform, deren Funktionalität ebenfalls über Windowsanwendungen implementiert wird. Die folgenden Beschreibungen basieren inhaltlich auf den Darstellungen in [HP95b], [HP95a] und [HP94].

2.3.1 Infrastruktur

Das Kernsystem wird unter HPOV vom Hauptprogramm ovwin.exe implementiert, das Nachrichten von den Plattform-Anwendungen empfängt und an den gewünschten Adressaten innerhalb der Plattform weitergibt. HPOV unterstützt IPX, Banyan VINES/IP und UDP/IP, mitgeliefert wird jedoch nur ein UDP/IP

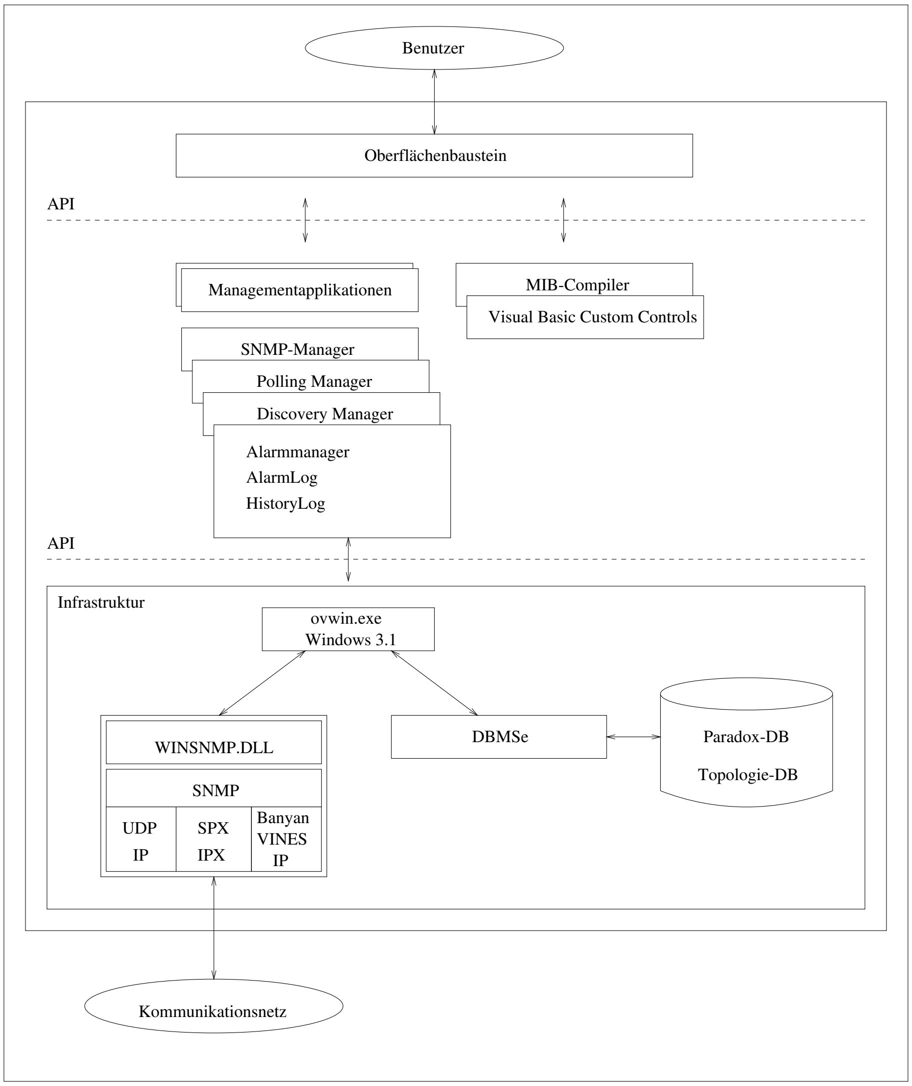

Abbildung 2.3: Funktionaler Aufbau von HPOV

Protokollstack der Firma FTP Software. Als Managementprotokoll verwendet HPOV ebenso wie NMS SNMP; HPOV bietet dabei allen Applikationen unter Umgehung des Kernsystems eine direkte, standardisierte SNMP-Schnittstelle zum Netz an.
Die Datenbanken und ihre Verwaltungssysteme, die das SNMP-Informationsmodell implementieren, benutzen teilweise proprietäres Format (vgl. Discovery Database 2.3.2); der Zugriff auf die Datenbank erfolgt dann über spezielle APIs.

2.3.2 Basisanwendungen

HPOV implementiert die funktionell die gleichen Basisanwendungen wie NMS:

- der *Polling Manager* überprüft mit ICMP (für IP-Komponenten) bzw. IPX Echo-Paketen in regelmäßigen, vom Benutzer konfigurierbaren Abständen die Verbindungen zu den einzelnen Systemen.

- Der *SNMP-Manager* fungiert als MIB-Browser, der die Ergebnisse von SNMP-Anfragen tabellarisch bzw. graphisch anzeigt.

- Das Topologiemanagement unter HPOV implementiert eine Autotopology-Anwendung: die Autodiscovery-Applikation sammelt als Hintergrundprozeß Topologieinformationen und legt diese in der Discovery Database - einem proprietären Datenbanksystem ab. Auf der Basis dieser Informationen baut die Layout-Funktion dann die Netztopologie des überwachten Netzes auf.

- Der Alarmmanager ist Teil des Hauptprogramms ovwin.exe: alle Alarme werden über eine einheitliche Schnittstelle für alle Anwendungsprogramme an ovwin.exe gemeldet. Der Alarmmanager empfängt dabei sowohl externe SNMP-Alarme via *Trap Manager* als auch interne Alarme vom Polling Manager oder anderen Anwendungen. Zusammen mit den Alarmen erhält der Alarmmanager die zugehörigen Konfigurationsinformationen für Statusanpassung, Abspeichern in den AlarmLog, akustische (Bell) und optische (Map update) Alarmanzeige sowie den Start von Programmen. Dabei ist jedoch zu bemerken, das nur ein Programm für jeden Objektstatus gestartet werden kann.
  Alarme, die im *AlarmLog* gespeichert sind, werden nach der Bestätigung durch den Benutzer in einen *HistoryLog* übertragen. Dabei handelt es sich in beiden Fällen um ein und diesselbe Paradox-Datenbank; bei der „Übertragung “vom AlarmLog in den HistoryLog wird intern nur das Feld für den Alarmstatus eines Alarms von „Open “auf „Cleared “geändert.

2.3.3 Oberflächenbaustein

Die Submap-Hierarchie kann unter HPOV entweder automatisch vom Discovery Manager oder manuell vom Netzwerkverwalter erzeugt werden, um verschiedene Views des Netzes zu realisieren. Die vom Discovery Manager generierte Submap-Hierarchie ordnet die Submaps nach *Internet View*, *Network Views* und *Segment Views*. Um die Darstellung des Netzes realitätsnah zu gestalten, können in jede Submap Bilder als Hintergrund mit eingebaut werden.

2.3.4 Entwicklungswerkzeuge

Neben einem MIB-Compiler enthält HPOV auch Visual Basic Custom Controls, Steuerelemente, die die Programmierung von Managementanwendungen in Visual Basic erleichtern, aber auch zum Teil in Visual C++ Anwendungen integriert werden können (bis VC++ 1.5).

2.4 MS Systems Management Server

Vor der Beschreibung von SMS muß auf zwei wesentliche Unterschiede zu den beiden vorangegangenen Managementsystemen hingewiesen werden:

1. **Die Zielsetzung**: während die beiden Plattformen von HP und Novell für das Netz- und Systemmanagement mit Einsatzschwerpunkt in LANs konzipiert sind, zielt Microsoft mit dem Systems Management Server auf das Systemmanagement von PCs und Servern in unternehmensweiten Netzen (Corporate Networks); der Begriff Systemmanagement für PCs bezieht sich dabei auf Aufgaben, die ein ganzheitliches Management mehrerer PCs erfordern, wie z.B. ein unternehmensweites Update von PC-Software. (vgl. [SMS95b])
   Dies bedeutet auch, daß für das Management nicht nur technische Faktoren eine Rolle spielen, sondern auch organisatorische und betriebswirtschaftliche Faktoren [Heg93].
   Da für die Unternehmen u.a. die Installation und das Update von Software auf ihren Rechnern sowie die Aufrechterhaltung der Funktionsfähigkeit ihrer Rechner wesentliche Kostenfaktoren sind, unterstützt SMS vor allem auch Anwendungen für Software Distribution und Remote Troubleshouting and Control.

2. **Die Management-Architektur**: SMS implementiert keine Plattformarchitektur im üblichen Sinn, die darauf abzielt, Anwendungsprogrammierern entsprechende APIs zur Erweiterung und Ergänzung der bestehenden Basisanwendungen zur Verfügung zu stellen. Die Hauptabsicht ist vielmehr, bestehende Lösungen für das Netzmanagement durch die Integration von SMS zu ergänzen (vgl. [SMS95b]). SMS stellt daher mehr eine Art Mischung aus rudimentärer Plattformarchitektur und Manager-of-Manager-Architektur dar (vgl. hierzu [Heg93]); die sich in der physischen Netzstruktur wiederspiegelnde Unternehmenshierarchie Zentrale-Zweigstellen wird managementtechnisch auf eine *Site*-Hierarchie abgebildet (Abbildung 2.4): Die einzelnen Sites werden dabei von den sogenannten *Site Servern* aus verwaltet, NT Rechnern, auf denen die SMS-Komponenten installiert sind. Je nachdem, ob die betreffende Site über eine eigene SQL Server Datenbank für die Daten der inventarisierten Systemkomponenten (vgl. 2.4.2) verfügt oder

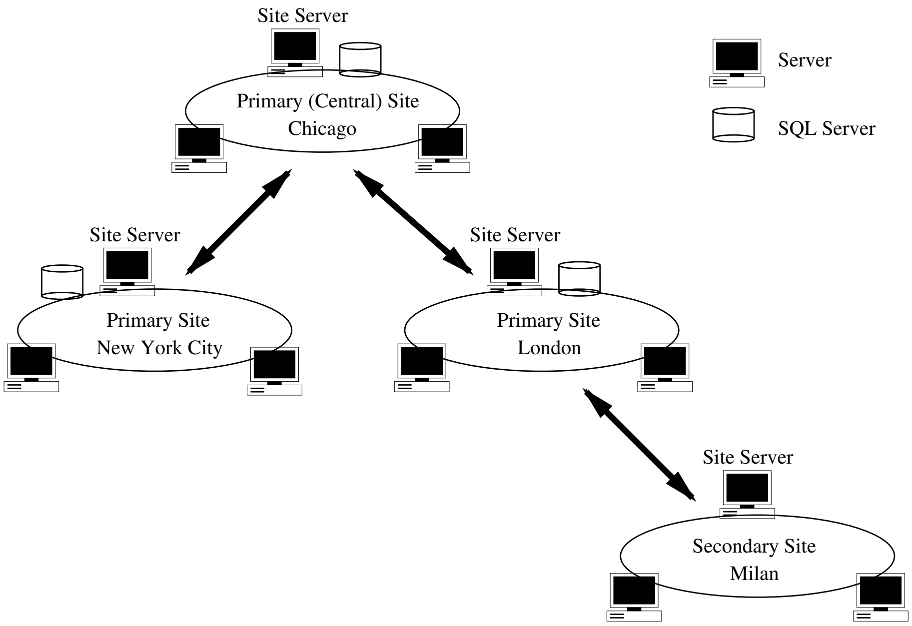

Abbildung 2.4: Site-Hierarchie unter SMS

nicht, wird von einer *Primary Site* oder einer *Secondary Site* gesprochen; die Primary Site der obersten Managementebene wird auch als *Central Site* bezeichnet. Jede dieser Sites kann lokal verwaltet werden, gibt ihre Managementinformationen immer aber auch an die ihr übergeordnete Site bis hin zur sogenannten Central Site weiter.

Dementsprechend läßt sich das Plattform-Architekturmodell nur dann auf SMS übertragen wenn das verwaltete Netz aus nur einer Site besteht.

Die folgenden Darstellungen halten sich inhaltlich - soweit nicht anders angegeben - an [SMS95c] und [SMS95a].

2.4.1 Die Infrastruktur

Der Informationsbaustein und die daran angeschlossene Microsoft SQL Server Datenbank implementieren das von der DMTF (Desktop Management Task Force) zugrunde gelegte Informationsmodell: Managed Objects werden durch sogenannte MIFs (Management Information Format) beschrieben. Neben standardisierten MIFs für Server und PCs kommen dabei unter SMS auch kundenspezifische MIFs zum Einsatz.(vgl. [DMT95a], [DMT95b], [DMT94]) Der Kommunitkationsbaustein von SMS unterstützt die Kommunikationsschnittstelle NetBIOS über die Kommunikationsprotokolle NetBEUI, TCP/IP und IPX; daneben Microsoft SNA Server LU 6.2 sowie RAS (Remote Access Service) über Telefon, X.25 und ISDN.(vgl. auch Anhang D)

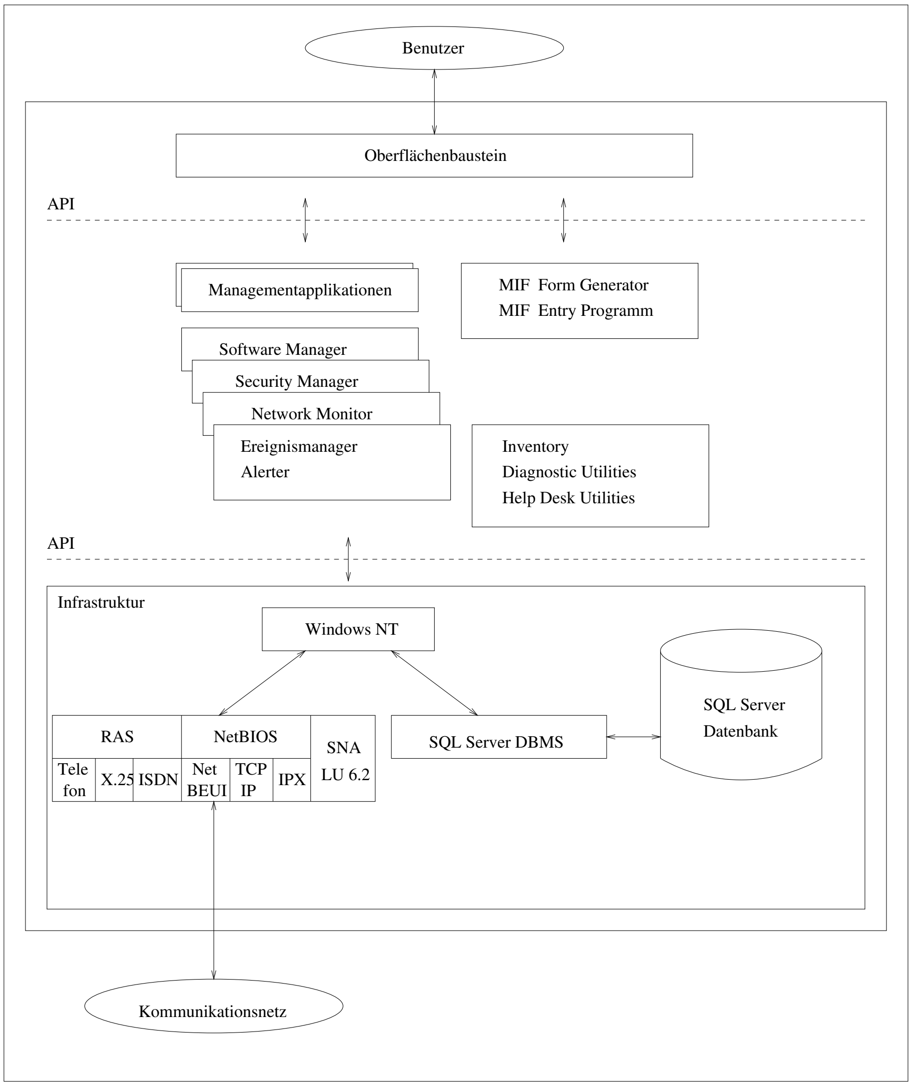

Abbildung 2.5: Funktionaler Aufbau von SMS

Er implementiert jedoch kein einheitliches Managementprotokoll; Managementinformationen, die während des Inventory-Prozesses (im allgemeinen Architekturmodell der Konfigurationsmanager) gesammelt werden, werden in sogenannten MIF-Dateien von den Agenten auf den Clients an den zuständigen Site Server weitergegeben und von diesem in der Site Database (SQL Server) abgelegt.

2.4.2 Basisanwendungen

Neben einigen Basisanwendungen des Architekturmodells implementiert SMS zusätzlich Anwendungen für das Software- und Sicherheitsmanagement:

- Das Konfigurationsmanagement unter SMS besteht aus Anwendungen für die Inventarisierung - der Inventory-Anwendung - und Utilities für die Fehlerbehebung in entfernten Systemen (Remote Troubleshooting).
  Die *Inventory-Anwendung* besteht ihrerseits aus mehreren Anwendungeni: dem sogenannten *Maintanence Manager*, der alle von den Agenten generierten Files sammelt und an den *Inventory Processor* weitergibt. Der Inventory Processor vergleicht die in den Files enthaltenen mit den derzeitig vorhandenen Managementinformationen und erzeugt daraus ein Delta-File, das alle geänderten Informationen enthält. Dieses Delta-File benutzt wiederum der *Inventory Data Loader*, um die Datenbank zu aktualisieren.
  Die *Werkzeuge für die Fehlerbehebung* enthalten Utilities für die Diagnose (Diagnostic Utilities) und den direkten Zugriff auf entfernte Systeme (Help Desk Utilities). Die Diagnose-Werkzeuge erlauben dem Verwalter, die aktuelle Konfiguration des Systems zu ermitteln sowie dessen Erreichbarkeit über ICMP zu testen. Über den direkten Zugriff auf das System können Fehler dann behoben werden.

- Ebenso wie für die Managementinformationen benutzen die Komponenten des SMS Systems MIFs, um Ereignisse an SMS weiterzugeben. Ereignisse werden dabei - im Gegensatz zu NMS und HPOV - nicht asynchron gemeldet, sondern in bestimmten, vom Benutzer einstellbaren Zeitintervallen abgefragt (lt. Auskunft eines SNI-Entwicklers nicht unter 15 Minuten). Das Ereignismanagement von SMS speichert diese Ereignisse in der SQL Server Datenbank ab und zeigt sie dem Benutzer in einem separaten Fenster an. Zum Ereignismanagement unter SMS gehört auch der *Alerter*, der das Eintreten bestimmter, vom Benutzer definierbarer Bedingungen innerhalb der Datenbank überwacht. Ist eine dieser Bedingungen erfüllt, kann der Alerter je nach Konfiguration ein Ereignis in der Datenbank abspeichern, eine Kommandozeile ausführen oder eine Nachricht versenden.

- Der *Network Monitor* implementiert einen Leistungsmonitor, der dem Betreiber einen Überblick über die Auslastung von Teilen bzw. des gesamten Netzes verschafft (Anzeigen: Netzauslastung, Rahmen/Byte pro Sekunde etc.), und so die Möglichkeit gibt Schwachstellen zu erkennen.

- Das Software Management ermöglicht dem Betreiber die Installation von Software auf Clients und Servern sowie von Netzwerkanwendungen auf Servern.

- Mit Hilfe des *Security Managers* können Zugriffsrechte für bestimmte Teile des Managements festgelegt werden.

2.4.3 Oberflächenbaustein

Die Toolbar des SMS Administrator Windows ermöglicht dem Benutzer den Zugang zu allen Managementanwendungen wie auch zur Darstellung der Netztopologie.

2.4.4 Entwicklungswerkzeuge

SMS bietet zwei Entwicklungswerkzeuge an, die es dem Verwalter erlauben, die Inventarisierung über die Standard MIFs (Server MIF und PC MIF) hinaus den eigenen Bedürfnissen entsprechend zu erweitern. Beispiele für mögliche Erweiterungen sind Angaben zum Standort eines PC wie Zimmernummer oder zu dessen Benutzer wie Name, Angestelltennummer etc. Der sogenannte *MIF Form Generator* hilft dabei, eine Art Formular zu generieren, das dann beispielsweise vom Benutzer eines PCs ausgefüllt werden muß. Der PC-Benutzer ruft dazu das MIF-Entry Programm auf, welches das ausgefüllte Formular anschließend in ein syntaktisch korrektes MIF-File übersetzt und in einem speziellen Verzeichnis ablegt, daß jeweils vom Inventory-Agenten auf derartige MIFs hin abgefragt wird.

Kapitel 3

Entwicklungsumgebungen

PC-Management-Plattformen sind als offene Systeme konzipiert. Das jeweils von Herstellerseite implementierte Basismanagement kann durch zusätzliche Managementanwendungen erweitert werden und die Managementfunktionalität insgesamt so auf die Bedürfnisse des jeweiligen Netzbetreibers zugeschnitten werden (vgl. Abschnitt 1.1).
In diesem Kapitel wird beschrieben, wie auf den Plattformen Managementanwendungen implementiert und wie sie in die Plattformen integriert werden können.

3.1 Implementierung

Zusätzliche Managementanwendungen können sowohl unter NMS als auch unter HPOV als Windows- oder DOS-Programme implementiert werden.
Novell empfiehlt für die Erstellung in [Nov93e] den Microsoft C Compiler ab Version 6.0a und den Microsoft C/C++ Compiler ab Version 7.0, während Hewlett Packard den Einsatz des Microsoft C Compilers erst ab Version 7.0 oder des Borland C Compilers ab Version 3.1 bevorzugt [HP95a]. Demgegenüber macht Microsoft keinerlei Angaben zum Einsatz eines bestimmten C Compilers für die Entwicklung von Managementanwendungen unter SMS.
Tatsächlich dürften aber auch andere C und C++ Compiler verwendet werden können, sofern sie zumindest den folgenden Randbedingungen genügen:

- Es müssen C/C++ Compiler für Intel-Prozessoren eingesetzt werden, da auch die Plattform-Module jeweils nur als Intel-basierte Versionen existieren.

- Die C/C++ Compiler müssen die Windows-APIs unterstützen, da die meisten Managementanwendungen als Windows-Programme implementiert werden.

3.2 Integration

3.2.1 Integration unter NMS

Unter NMS wird die Integration von Managementanwendungen durch den Application Integrator, einem Paket aus mehreren Integrationswerkzeugen, unterstützt (vgl. [Nov93b]).

Methoden

Der Application Integrator unterstützt momentan zwei Integrationsmethoden (eine dritte Integrationsmethode ist für zukünftige Versionen vorgesehen), die sich hinsichtlich der Startbedingungen (für den Benutzer) und dem Grad der Integration unterscheiden: Quick Launch und Smart Launch.

Quick Launch Bei dieser Methode wird lediglich ein zusätzlicher Menüpunkt in eines der NMS-Menüs eingefügt, von dem aus die Anwendung dann gestartet werden kann.

Smart Launch Die Smart Launch Methode erreicht gegenüber Quick Launch eine bei weitem höhere Stufe der Integration. Die verbesserte Integration beruht darauf, daß die Anwendung zu einem integrierten Teil des in der NMS Datenbank implementierten, objektorientierten Informationsmodells wird:
Jeder Knoten - also jeder Netz- und jeder Systemkomponente - wird innerhalb der DB durch eine Instanz des NMS-Objekttyps BOX repräsentiert. Jede BOX wiederum stellt eine Art Container u.a. für die Funktionen des betreffenden Knotens dar. Knotenfunktionen wie z.B. NetWare Server Service werden ebenso wie die Knoten als Objekte aufgefaßt und über Instanzen der entsprechenden (Funktionen-)Objektklassen dargestellt; über eine sogenannte Owner-Relationship wird die Verbindung zur BOX des Knotens hergestellt.
Managementanwendungen nehmen häufig Bezug auf bestimmte Aspekte bestimmter Netz- oder Systemkomponenten (Beispiel Servermanagement: wann ist ein Knoten ein Server?). Daher fordern sie von ihren Zielknoten eine gewisse Funktionalität und umgekehrt muß ein Zielknoten diese Funktionalität erbringen. Eine Managementanwendung kann auf diese Weise auch als Funktion der Zielknoten aufgefaßt werden.
Die Smart Launch Integrationsmethode basiert im wesentlichen darauf, daß die Managementanwendung als neuer (Funktions-)Objekttyp innerhalb der NMS Datenbank definiert wird.
Dadurch ergeben sich u.a. folgende, zusätzliche Möglichkeiten der Integration:

- Eine Anwendung kann sowohl unspezifisch wie für die Quick Launch Integration als auch auf einem bestimmten Mapobjekt (i.e. ein möglicher Zielknoten oder ein Segment) gestartet werden: das Objekt wird ausgewählt und über Menü gestartet.

- Der NetExplorer (vgl. 2.2.2) kann so konfiguriert werden, daß er die neue Funktion auf den einzelnen Knoten sucht.

- NMS stellt sowohl globale (Name des NetExplorer Servers, Hintergrundfarbe der Map etc.) als auch objektspezifische Informationen über die Dialogseiten sogenannter Dialog-Bücher zur Verfügung. Neue Dialogseiten, deren Inhalt mit der neuen Funktion im Zusammenhang steht, können mit dem NMS Dialog Book GUI Tool erstellt und in die bestehenden Dialog-Bücher eingefügt werden.

Die Integrationsmethode und die Integrationsparameter werden in einem sogenannten *Object Class Definition* (.OLF) File festgelegt; als Werkzeug für die Erstellung einer .OLF-Datei stellt NMS den *OLF Editor* zur Verfügung.

Ablauf

Der Integrationsprozeß umfaßt zwei Stufen: die Entwicklungsstufe und die Installationsstufe.
In der *Entwicklungsstufe* werden alle für die Integration benötigten Ressourcen erstellt: die Anwendung selbst, die .OLF Datei, die Icon-Datei für die neue Funktionsklasse sowie anwendungsspezifische .INI- und Help-Dateien. In der *Installationsstufe* werden alle Ressourcen in bestimmte Verzeichnisse kopiert und entsprechende Einträge in der NMS .INI Datei vorgenommen. Die eigentliche Integration wird dann durch den *OLF Introducer* vorgenommen.

3.2.2 Integration unter HPOV

Anwendungen werden unter HPOV über eine Initialisierungsprozedur innerhalb des WinMain-Programms (i.e. das Hauptprogramm jedes Windowsprogramms analog zu main in C-Programmen). der Anwendung integriert. Die Initialisierungsprozedur legt über den Aufruf entsprechender Funktionen folgendes fest:

- welche Menüpunkte von der Anwendung eingefügt werden (Aufruf von *OVMenuAddExt()* und *OVMenuAddCommandExt()*),

- welche Benutzer die Anwendung starten können (HPOV Sicherheitsmechanismus),

- welche Alarme die Anwendung melden kann (*OVAlarmRegisterSet()* und *OVAlarmRegisterType()*, vgl. auch 9.2),

- ob die Anwendung über SNMP kommunizieren will (durch Aufruf der WinSNMP-Funktion *SnmpRegister()*, vgl. auch 8),

- ob die Anwendung (auch) als Discovery-Funktion agiert (*OVADLRegisterForAutodiscovery*, vgl. auch 10.2),

Damit die Anwendung überhaupt gestartet und damit die Initialisierungsprozedur aufgerufen wird, muß sie in die [OpenViewApps] Section der OVWIN.INI Datei eingetragen werden.

3.2.3 Integration unter SMS

Da SMS keine Plattform im eigentlichen Sinn ist (vgl 2.4), ist auch keine Integration von Anwendungen wie unter NMS oder HPOV vorgesehen, sondern nur die Installation als eigenständige Anwendung. SMS-Anwendungen steht nur das SMS Data API als Programmierschnittstelle zu einer Site-Datenbank zur Verfügung (vgl 2.4); die Applikationen müssen daher auf einem NT-Rechner installiert werden, auf dem eine Site-Datenbank eingerichtet ist. Darüberhinaus muß lediglich sichergestellt werden, daß die Anwendung Zugriff auf alle notwendigen Laufzeitbibliotheken hat (SMSAPI.DLL und OBJECTTY.DLL).

Kapitel 4

Der SNI Server Manager

Einer der Hauptgründe für die Implementierung von Management-Plattformen ist, dem Netzwerkbetreiber durch die Bereitstellung definierter Schnittstellen die Möglichkeit zu geben, die Funktionalität der Plattform seinen Bedürfnissen entsprechend zu erweitern. Um eine Vorstellung davon zu geben, wie eine solche Erweiterung typischerweise aussehen kann, soll in diesem Kapitel der auf NMS (vgl. 2.2) implementierte Server Manager von SNI vorgestellt werden. Dabei wird zunächst die Software-Architektur beschrieben, bevor dann auf die Einbindung und die Schnittstellen des Server Managers - insbesondere diejenigen zur Plattform - eingegangen wird. Die Beschreibungen beschränken sich dabei auf das Wesentliche, so daß kein Anspruch auf Vollständigkeit besteht. Inhaltlich basieren die Darstellungen dabei auf der Lösungsstudie [SNI95] für den Server Manager.

4.1 SW-Architektur

Der Server Manager wurde sowohl als *Stand-alone-* als auch als *Snap-in-*Anwendung implementiert: Stand-alone bedeutet im Gegensatz zu Snap-in, daß keine Einbindung der Applikation in die Plattform gegeben ist. Die Stand-alone Anwendung nutzt lediglich den vorhandenen Kommunikationsbaustein ohne Plattform zur SNMP-Kommunikation aus.
Alle Server, die überwacht werden sollen, können (NetWare Server, IPX) oder müssen (IP Server) vom Benutzer über ein Dialogfenster der Applikation eingegeben werden; IPX Server können aber auch automatisch - über Abfrage einer (Default-) Bindery eines NetWare Servers - ermittelt werden.
Die Überwachung der Server erfolgt über:

- das Alarmmanagement:
  für jedes Ereignis senden die Agenten auf den Servern einen SNMP-Trap an die Managementstation. Die Art der Ereignisse, die gemeldet werden, wird im allgemeinen von den Trap-Definitionen in den betreffenden MIBs

bestimmt. Der Server Manager bietet darüberhinaus die Möglichkeit, bei einigen (proprietären) Ereignissen (Threshold-Überwachung) serverspezifisch festzulegen, ob sie als Traps gesendet werden sollen oder nicht.
In der Alarmkonfiguration ist auch festgelegt, wie das Alarmmanagement auf einen eingegangenen Trap reagieren soll. Dabei kann in der Alarmkonfiguration des Server Managements zusätzlich zu den schon von der Plattform angebotenen Möglichkeiten - Alarm abspeichern, akustisches Signal erzeugen, Starten eines externen Programms - festgelegt werden, ob der Alarm an einen Pager weitergeleitet werden soll oder eine Nachricht an eine bestimmte Station (z.B. Administrationsplatz) gesendet werden soll.
Dem Benutzer werden die Alarme entweder im Alarm Monitor (jeweils die letzten Alarme) oder im Alarm Manager Window (alle geloggten Alarme, auf Wunsch gefiltert) angezeigt.

- Threshold-Überwachung:
  der Verwalter kann serverspezifische Schwellwerte (Thresholds) in Bezug auf verschiedene MIB-Variablen des Typs Counter oder Gauge definieren. Die Überwachung dieser Schwellwerte übernimmt die Threshold-Funktion im Server; wird ein Schwellwert über- oder unterschritten generiert die Threshold-Funktionen ein entsprechenden Ereignis, das als Trap an den Manager gesendet wird.

- Reports:
  ein Report ist die regelmäßige Abfrage vom Betreiber festgelegter, serverspezifischer MIB-Variablen über SNMP. Das Ergebnis zeigt den zeitlichen Verlauf der Variablenwerte an und wird entweder als Tabelle oder graphisch präsentiert.

- Abfrage der aktuellen Serverkonfigurationen:
  der Server Manager zeigt dem Verwalter auf Wunsch die aktuelle Konfiguration eines bestimmten Servers an. Die Daten werden dabei nach logischen Bereichen geordnet angezeigt:
  Mass Storage: enthält eine Übersicht und Informationen zu den Massenspeichern. Dazu gehören Daten über die Controller (Bus-Typ, Slot-Nummer etc.), das Gerät (u.a. Kapazität, Sektoren, Zylinder), die Partitionen sowie das Filesystem.
  System Board: enthält die Einstellungsdaten des Mainboards. Beispiele: Anzahl, Typ und Status der vorhandenen Prozessoren, Typ, Name, Slot der installierte Adapter.
  Power Supply: informiert über die Art und Zustand der Stromversorgung des betreffenden Servers sowie evtl. vorhandener HD-Erweiterungsschränke. Die graphische Darstellung gibt also beispielsweise darüber Aufschluß, ob eine Battery Backup Unit und/oder eine ununterbrechbare Stromversorgung vorhanden ist oder nicht.

Network Interfaces: enthält Informationen zu den Netzwerkanschlüssen und und Paketstatistiken. Beispiele: Typ (CSMA/CD, Tokenring), Hersteller, Adresse einer Schnittstelle.
Environment: enthält im wesentlichen spezifische Angaben zum SNI System Control Board (SCB) wie z.B. Lüfterfunktion oder Temperatur.
Operating System: zeigt Betriebssysteminformationen von NetWare- und Windows NT-Servern an. Beispiele: Name, Version, aktuelle Auslastung (Windows NT) etc.
Recovery: zeigt den Inhalt des Fehlerspeichers des SCB an z.B. Parity Fehler, CPU-Fehler und Fehler in der Stromversorgung.
Alle Informationen zur Serverkonfigurationen sind in Standard-MIBs und proprietäre MIBs enthalten und werden jeweils aktuell über SNMP abgefragt.

4.2 Einbindung und Schnittstellen des Server Manager

Der SNI Server Manager wird über ein spezielles Startprogramm in NMS eingebunden, das selbst wiederum über ein NMS-Entwicklungswerkzeug, den sogenannten OLF-Compiler (vgl. 3.2.1), in NMS integriert wurde. Die dafür notwendige OLF-Datei wurde ebenfalls mit NMS-Tools erstellt.
Das vom Startprogramm gestartete Hauptprogramm des Server Manager überwacht verschiedene, von NMS beschriebene DOS-Umgebungsvariablen, über die anderen Anwendungen mitgeteilt wird, welche Daten angefordert werden. Wird also unter NMS das SNI-Server-Icon angeklickt, wird von NMS eine entsprechende Umgebungsvariable beschrieben, die wiederum vom Hauptprogramm gelesen wird. Daraufhin bringt sich der Server Manager selbst in den Vordergrund und zeigt die geforderten Informationen an.
Die Implementierung des Server Manager - hier soll in diesem Zusammenhang nur die Snap-in Lösung betrachtet werden - stützt sich auf mehrere Schnittstellen ab, um die geforderte Funktionalität zu realisieren:

- Schnittstelle zum NMS SNMP Data Server: wie bereits erwähnt werden alle Managementinformationen jeweils aktuell über SNMP von den einzelnen Servern angefordert.

- Schnittstelle zum NetWare Client SDK von Novell: jeder NetWare Server verwaltet eine kleine Datenbank, die sogenannte Bindery, die unter anderem die Namen und IPX-Adressen aller anderen am gleichen Netz angeschlossenen NetWare Server enthält. Die im Server Manager enthaltene Discovery-Funktion, die diese Daten automatisch sammelt, greift periodisch über die NetWare Client-Schnittstelle auf die (Default-) Bindery zu.

- Mit Hilfe der Microsoft Foundation Classes wird die Benutzerschnittstelle implementiert.
- Für die Online-Dokumentation wird die Microsoft Windows Help verwendet.
- Für graphische Darstellungen werden die Microsoft Custom Controls herangezogen.

Kapitel 5

Allgemeines Lösungskonzept

In diesem Kapitel wird zunächst die Problemstellung anhand eines einfachen Schichtenmodells verdeutlicht und davon ausgehend die Konzeption einer allgemeinen Lösung entwickelt. Der damit umrissene allgemeine Lösungsraum wird in einem zweiten Schritt durch die Spezifizierung von zusätzlichen Randbedingungen eingeschränkt. Anschließend wird versucht, daraus konkrete Lösungsschritte abzuleiten.

5.1 Allgemeine Konzeption

Im Rahmen der Diplomarbeit soll ein Portabilitätskonzept entwickelt werden, dessen Umsetzung die Portierbarkeit allgemeiner Managementanwendungen so weit wie möglich erreichen bzw. erleichtern soll. Der Bereich der Zielplattformen wird dabei auf die genannten drei Plattformen eingeschränkt: NetWare Management System (NMS) von Novell, HP OpenView for Windows (HPOV) und Systems Management Server (SMS) von Microsoft.
Entscheidend für die Entwicklung eines solchen Konzepts ist der Begriff der Portierbarkeit; dieser Begriff soll daher zunächst kurz an einem verallgemeinerten, sehr einfachen Schichtenmodell für Managementplattformen und -anwendungen erläutert werden:

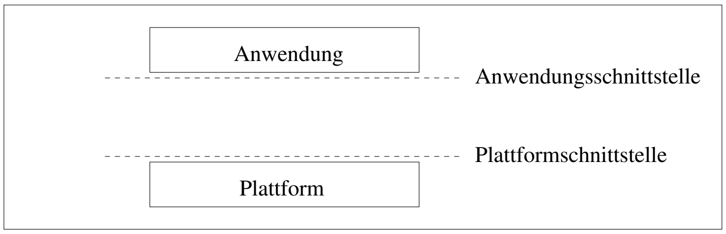

Abbildung 5.1: Ausgangssituation

Die Anwendungen nehmen über eine Schnittstelle zur Plattform (Anwendungsschnittstelle) die Dienste der jeweiligen Plattform in Anspruch, während die Plattformen ihrerseits den Managementanwendungen über eine Schnittstelle (Plattformschnittstelle) ihre Dienste zur Verfügung stellen. Wird eine Anwendung für eine Plattform implementiert, ergibt sich in diesem Modell folgendes Bild :

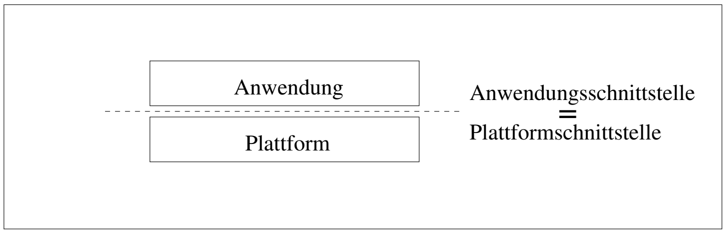

Abbildung 5.2: Plattformspezifische Implementierung

Anwendungs- und Plattformschnittstelle stimmen überein.
Auf portierbare Anwendungen läßt sich dieses Modell ebenfalls anwenden: auch eine portierbare Anwendung besitzt eine Anwendungsschnittstelle; da sie jedoch auf allen (drei) Plattformen laufen muß, kann hier die Identität von Anwendungs- und Plattformschnittstelle i.a. nicht gelten. Für jede Plattform muß dementsprechend Software entwickelt werden, die die Abbildung der Anwendungsschnittstelle auf die jeweilige Plattformschnittstelle implementiert. Im Schichtenmodell kann dies auch so interpretiert werden, daß zwischen Anwendungen und Plattformen eine Zwischenschicht eingezogen wird, die die Abbildungen übernimmt; die Anwendungen rufen damit Dienste der Zwischenschicht auf.

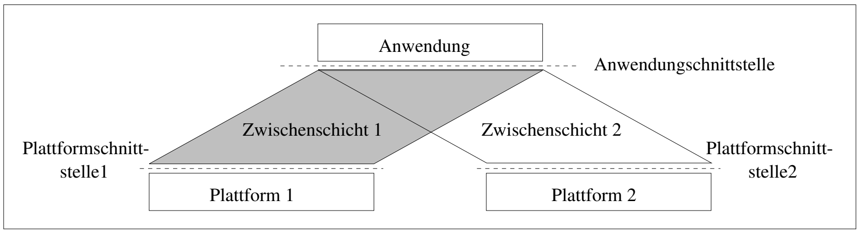

Abbildung 5.3: Portierbare Implementierung

Das Portabilitätskonzept muß daher folgendes beinhalten:

- Die Spezifikation einer Anwendungsschnittstelle für die Anwendungen: Funktionsaufrufe, Datentypen und -strukturen sowie die Semantik der Funktionsaufrufe.
- Ein Konzept für die Implementierung der Abbildung dieser Anwendungsschnittstelle auf die Plattformschnittstellen der drei Plattformen.

5.2 Randbedingungen

Eine Lösung sollte jedoch zusätzlich noch mehreren Randbedingungen genügen:

- Der Aufwand für die Implementierung der Abbildungen auf die Plattformschnittstellen soll möglichst gering gehalten werden.
- Bei der Spezifizierung der Anwendungsschnittstelle sollen Schnittstellenstandards wie z.B. das Standard-API WinSNMP für die Kommunikation über SNMP (vgl. Kapitel 8 und Anhang A) nach Möglichkeit berücksichtigt werden. Die Verwendung von Standards hat dabei mehrere Vorteile, u.a.:
  - Es kann davon ausgegangen werden, daß die Funkionalität der Implementierung einer Standardschnittstelle in ihrem Anwendungsbereich ausreichend ist; darüber, welche Funktionalität ausreichend ist haben sich bereits die Entwickler der Schnittstelle Gedanken gemacht.
  - Eine Anwendung, die einen Schnittstellen-Standard unterstützt, ist besser portierbar, da sich auch die Plattformentwickler an Standards orientieren; darüberhinaus ist damit zu rechnen, daß über kurz oder lang alle oder zumindest die meisten Plattformen den Standard unterstützen.
  - Der Lernaufwand für einen Anwendungsprogrammierer wird so geringer gehalten.
- Die Funktionalität der Anwendungsschnittstelle darf diejenige der Plattformschnittstellen nicht unnötig verkleinern. „Unnötig “ist dabei im Bezug auf die *potentielle* Funktionalität der virtuellen Managementplattform zu sehen.
- Funktionale Beschränkungen der Plattformschnittstellen aufgrund der Plattformimplementierungen oder darunterliegender Schnittstellen müssen bei der Spezifizierung der Anwendungsschnittstelle berücksichtigt werden, sofern sie nicht mit vertretbarem Aufwand umgangen werden können. Beispiele: die Maximalgröße der versendbaren SNMP-PDUs, Möglichkeiten der Datenbankanbindung (7.1).

5.3 Lösungsschritte

Ausgehend von der allgemeinen Lösungskonzeption und den Randbedingungen können einzelne Lösungsschritte näher spezifiziert werden.

Zerlegung in Teilprobleme

Ein in der Softwareentwicklung allgemein übliches Verfahren bei der Lösung komplexer Probleme ist die Modularisierung in mehrere, kleinere Teilprobleme [Mar94]. Da sich die einzelnen APIs der Plattformen verschiedenen funktionalen Bereichen zuordnen lassen, bietet sich eine dementsprechende Unterteilung in einzelne Problembereiche von selbst an. Portabilitätskonzepte müssen danach für folgende Anwendungs-Schnittstellen entwickelt werden:

- die Schnittstelle für die SNMP-Kommunikation,
- die Schnittstelle zum Alarmmanagement,
- die Schnittstelle zum Topologiemanagement,
- die Schnittstelle zum Konfigurationsmanagement,
- die Schnittstelle zur graphischen Benutzeroberfläche,
- die Schnittstelle zum Betriebssystem der Plattform.

Spezifizierung der Anwendungsschnittstellen

Hier ist zunächst jeweils zu untersuchen, ob für die betrachtete Anwendungs(teil)schnittstelle anerkannte Standards existieren. Evtl. vorhandene Standards müssen dann bei der Spezifizierung berücksichtigt werden, nicht zuletzt auch im Hinblick auf ein mögliche Erweiterung der Portabilität auch auf andere Plattformen.
Existieren (noch) keine Standards, muß ein Kompromiß gefunden werden zwischen der benötigten Funktionalität seitens der Anwendungsschnittstelle und dem dafür insgesamt erforderlichen, geschätzten Implementierungsaufwand für die Abbildung auf die Plattformschnittstellen.

Kapitel 6

Einbindung der Zwischenschichten

Eine Problemstellung im Zusammenhang mit der Entwicklung des Portabilitätskonzepts, die unabhängig von Managementbereichen und konkreten APIs behandelt werden kann, ist die Einbindung der Zwischenschichten in die Anwendungen. Dabei stehen grundsätzlich zwei Methoden zur Auswahl: die statische und die dynamische Bindung.
Im ersten Abschnitt dieses Kapitels werden zunächst die grundsätzlichen Vorteile der dynamischen Bindung gegenüber der statischen Bindung erläutert und die Entscheidung für die Verwendung der dynamischen Bindung so begründet.
Daß diese Entscheidung nicht nur formaler Natur ist, sondern durchaus Auswirkungen auf das Design des Portabilitätskonzepts hat, zeigt dann der folgende Abschnitt: die dynamische Bindung verursacht mitunter auch Probleme. Eines dieser Probleme, das auch im Rahmen dieser Arbeit auftaucht (vgl. 8.3.2) und ursächlich mit der Speicherverwaltung dynamisch eingebundener Routinen unter Windows zusammenhängt, ist die Lebensdauer von Variablen, die diese Routinen definieren.

6.1 Einbindung der Zwischenschichten

Alleine die logische Unterteilung in Anwendungsprogramm und Zwischenschicht schreibt per se noch keine konkrete Methode der Einbindung vor. Zur Auswahl stehen also zunächst sowohl die Methode der statischen als auch die der dynamischen Bindung.
Die Methode der dynamischen Bindung - unter Windows über die sogenannten Dynamic Link Libraries (DLLs) realisiert - bietet jedoch u.a. folgende allgemeinen Vorteile (vgl. auch Anhang C):

- Speicherplatzersparnis sowohl im Hauptspeicher als auch auf der Festplatte; der Code dynamisch einbindbarer Routinen muß jeweils nur einmal vorhanden sein, gleichgültig wieviele Anwendungen darauf zugreifen.

- Die Änderung einer DLL-Routine erfordert keinen erneuten Linker-Aufruf für das betreffende Programm, solange die Schnittstelle zwischen dem „Hauptprogramm “und der DLL unverändert bleibt.

Im Hinblick speziell auf die Portabilität von Managementanwendungen auf Plattformen bedeutet dies, daß eine Erweiterung auf weitere Plattformen ohne Änderung der Hauptanwendung nur durch Austausch der entsprechenden DLLs möglich ist.
Nähere Informationen über DLLs finden sich im Anhang C.
Ein Nachteil, den die Verwendung von DLLs u.U. mit sich bringt, ist die aufwendigere Verwaltung globaler Variablen (einer Anwendung) durch eine DLL; Einzelheiten der Problemstellung und ihre Lösung werden im folgenden Abschnitt geschildert.

6.2 Lebensdauer von DLL-Variablen

Problemstellung

DLL-Routinen verwenden für die dynamische Speicherbelegung ihre eigenen Datensegmente, benutzen also nicht die Datensegmente der Anwendungen, durch die sie aufgerufen werden. Nach der Beendigung einer Routine und der Rückkehr ins aufrufende Programm wird dieses Datensegment gelöscht bzw. ist nicht mehr zugänglich.
Manchmal ist es jedoch erforderlich, daß die von einer DLL definierte und belegte Variablen länger erhalten bleiben; die Variablen sollen beispielsweise bis zur Beendigung der aufrufenden Anwendung gültig sein (vgl. Local Database Functions in 8.3.2).

Lösung

Eine Lösung bieten hier die Verwaltungsroutinen für den globalen Heap - i.e. der gesamte zur Verfügung stehende, freie Arbeitsspeicher. Die DLL-Routinen belegen mit *GlobalAlloc* Segmente auf dem globalen Heap, wobei diese Segmente über entsprechende Flags z.B. auch als verschiebbar, aber nicht verwerfbar belegt werden können. Der von *GlobalAlloc* zurückgelieferte Handle auf das allokierte Segment muß dann an die aufrufende Anwendung, die beim Aufruf der DLL-Routine einen entsprechenden Rückgabeparameter mitliefern muß, zurückgegeben. Von da an ist die Verwaltung des Datensegments - insbesondere die Freigabe mit *GlobalFree* - Aufgabe der Anwendung.
Bei Aufrufen von DLL-Routinen, die dieses Datensegment verwenden, muß dann jeweils der entsprechende Handle mitgegeben werden.

Die Verwaltungsroutinen für den globalen Heap sind auch unter Windows NT verfügbar (vgl. [Ric94]).

Kapitel 7

Datenbankanbindung von Managementanwendungen

Ein weiterer Problembereich, der zu einem Großteil unabhängig von konkreten Implementierungen auf den Plattformen behandelt werden kann, ist der Bereich der Datenbankanbindung in den einzelnen Managementbereichen:
In einigen Managementbereichen der Plattformen werden Datenbanken verwendet. Diese Datenbanken werden zum Teil von den Basisanwendungen generiert und/oder genutzt, können aber auch von anderen, zusätzlichen Managementanwendungen eingerichtet worden sein. Voraussetzung dafür ist allerdings, daß auf den Plattformen ein erweiterbares Datenbanksystem installiert ist; dies trifft jedoch für alle drei der untersuchten Plattformen zu und daher können Anwendungen *prinzipiell* auf allen drei Plattformen Datenbanken einrichten. Die eingesetzten Datenbanksysteme sind jedoch proprietär; dementsprechend verwenden Anwendungen auf jeder Plattform andere Datenbankschnittstellen. Eine Aufgabe des Portabilitätskonzepts ist es deshalb auch, diese verschiedenartigen Datenbankschnittstellen zu vereinheitlichen. In diesem Kapitel werden die Möglichkeiten einer einheitlichen Datenbankanbindung - zumindest innerhalb der einzelnen Managementbereiche - untersucht.
Im Mittelpunkt stehen dabei jeweils die Fragen:

- Existiert eine Standardschnittstelle zur einheitlichen Anbindung verschiedener Datenbanksysteme?

- Wenn ja: unter welchen Bedingungen kann diese Schnittstelle von Managementanwendungen als Zwischenschicht-API verwendet werden?

- Wenn nein: wann muß eine spezielle, an die bestehenden Datenbanken angepaßte Schnittstelle entwickelt werden?

7.1 Allgemeine Problemstellung

7.1.1 Vorbemerkung

Im folgenden werden immer wieder die Begriffe Datenbank (DB), Datenbanksystem (DBS) und auch Datenbankmanagementsystem (DBMS) verwendet; diese Begriffe werden hier jedoch in einer anderen, allgemeineren Bedeutung als der üblichen verwendet.
Im allgemeinen versteht man unter einem DBS eine Kombination aus einem DBMS und einer oder mehreren DBen. Dabei ist eine DB eine Sammlung von Daten, die von einem DBMS verwaltet wird. Das DBMS fungiert dabei als Schnittstelle zwischen Benutzern und Anwendungsprogrammierern auf der einen Seite und der oder den DB(en) auf der anderen Seite: es ermöglicht und kontrolliert den Zugriff auf die Daten [Vos94].
Ein DBMS erfüllt üblicherweise mehrere Bedingungen wie z.B. einheitliche Verwaltung der Daten (Datenintegration), Datensicherung, Zugriff auf die DB-Beschreibung (Data Dictionary) etc. (vgl.[Heu95]).
Die Datenhaltung auf den Plattformen beschränkt sich jedoch nicht auf DBSe (in obigem Sinn), sondern schließt auch einfache, proprietäre Dateisysteme ein wie z.B. die HPOV-Topologie-„Datenbank“(vgl. 2.3.2) Im Rahmen dieser Arbeit wird deshalb abweichend von der allgemein gültigen Definition *jede* Datensammlung und ihre dazugehörige Verwaltung, die auf den Plattformen implementiert sind, als DBS bezeichnet - auch dann, wenn die Verwaltung nur einfache Dateizugriffsoperationen zuläßt. Die Verwaltung selbst wird in der gleichen Allgemeinheit als DBMS bezeichnet.

7.1.2 Allgemeine Problembeschreibung

Wie oben schon angedeutet wurde ergeben sich für Managementanwendungen eine Reihe von Möglichkeiten zur Datenbankanbindung. Die DB-APIs lassen sich dabei grob in drei Kategorien einteilen:

1. Veränderung (Löschen, Einfügen, Aktualisieren) und Abfrage bestehender DBen d.h., solcher DBen, die von den Basisanwendungen der Plattformen angelegt wurden und von ihnen benutzt werden.

2. Veränderung und Abfrage von DBen, die von anderen Anwendungen als den Basisanwendungen generiert wurden.

3. Generierung von DBen durch die Anwendungen selbst und Verändern sowie Abfragen dieser DBen.

Die Zielsetzung im Rahmen der Entwicklung des Portabilitätskonzepts ist, die DB-Anweisungen innerhalb der einzelnen Managementbereiche zu vereinheitlichen. Um die damit verbundenen Probleme und Lösungen adäquat beschreiben

zu können, müssen zunächst einige zusätzliche Begriffe der DB-Terminologie eingeführt werden:
Die von einem DBS bzw. DBMS der „Außenwelt "und damit auch den Anwendungsprogrammierern zur Verfügung gestellte Schnittstelle wird in der DB-Terminologie auch als DB-Sprache bzw. DB-Sprachschnittstelle bezeichnet [Vos94]. DB-Sprachen lassen sich funktional im wesentlichen in drei (Teil-)Sprachen zerlegen:

- Die *Daten-Definitionssprache* (Data Definition Language, DDL), ein Werkzeug für die Umsetzung eines DB-Schemas - i.e. die konzeptionelle syntaktische und semantische Beschreibung einer DB anhand eines DB-Modells wie z.B. des relationalen DB-Modells - in ein konkretes DBS (vgl. [Heu95]). Beispiele für DDL-Befehle in der relationalen DB-Sprache SQL sind der *create table*- und der *drop table*-Befehl.

- Die *Daten-Manipulationssprache* (Data Manipulation Language, DML), die die Anweisungen für Anfragen an eine DB und für Änderungen der DB-Daten enthält ([Vos94]). In SQL gehört beispielsweise der select-Befehl zu dieser Sprache.

- Die *Datenverwaltungssprache* (Data Administration Language, DAL), die z.B. Anweisungen für die Vergabe von Zugriffsrechten in einer DB enthält.

Gemäß dieser Einteilung verwenden Managementanwendungen damit sowohl DML-Anweisungen (APIs der 1., 2. und 3.Kategorie) als auch DDL-Anweisungen (APIs der 3.Kategorie).
Über die Unterscheidung in Teilsprachen hinaus lassen sich die DB-Anweisungen dieser beiden Sprachen noch weiter unterteilen:
Jede DML-Anweisung besteht aus einem Operationsteil, der die auszuführende Operation bestimmt, und einem Spezifikationsteil, der die Daten bestimmt, auf die die Operation angewendet werden soll; Abbildung 7.1 zeigt dies am Beispiel eines SQL select-Befehls. Dieser Befehl gibt die Attribute AUTOR und TITEL aller

Operationsteil	Spezifikationsteil
select from where	autor, titel buecher leihfrist > 0;

Abbildung 7.1: Operations- und Spezifikationsteil am Beispiel eines SQL-Befehls

Einträge aus der Tabelle BUECHER zurück, für die das Attribut LEIHFRIST einen Wert größer 0 aufweist; der Spezifikationsteil legt dabei die Namen der Tabellen

und der Attribute fest.
Analog bestehen DDL-Operationen aus einem Operationsteil und einem Spezifikationsteil; der Spezifikationsteil beschreibt hier jedoch den von der Operation betroffenen Teil des implementierten DB-Schemas, also beispielsweise in einer relationalen DDL eine Tabelle und die dazugehörigen Attribute.
Um die DB-Schnittstellen zu vereinheitlichen müssen also einheitliche Operationen *und* einheitliche Spezifikationen entwickelt werden. Für DML-Anweisungen, über die auf bestehende DBen zugegriffen wird, ergibt sich daraus ein weiteres, grundsätzliches Problem:
im DML-Bereich bedeutet Spezifikation die Spezifikation konzeptueller Daten; die Anwendung einer einheitlichen Datenspezifikation setzt jedoch voraus, daß die Daten, auf die damit zugegriffen werden soll, auch vorhanden sind. Bevor also die Datenspezifikationen festgelegt werden, muß festgelegt sein, welche Daten überhaupt in der DB vorhanden sein sollen. Evtl. fehlende Daten müssen dann durch zusätzliche Anwendungen gesammelt und abgespeichert werden.
Grundsätzlich stehen für die Vereinheitlichung der DB-Schnittstellen zwei Methoden zur Auswahl:

1. Die Verwendung einer standardisierten Schnittstelle jeweils an Stelle der ursprünglichen, proprietären Schnittstellen.

2. Die Entwicklung einer gemeinsamen, an die Schnittstellen der DBSe in den einzelnen Managementbereichen angepaßten (niedrigerer Implementierungsaufwand für die Zwischenschicht! Vgl. 5.2), speziellen Schnittstelle.

Dabei ist die Wahl einer standardisierten Schnittstelle - falls vorhanden - der einer speziellen Schnittstelle aus Gründen der Portierbarkeit, Vollständigkeit in Bezug auf die geforderte Schnittstellenfunktionalität etc. vorzuziehen (vgl. 5.2).

7.2 Allgemeiner Lösungsansatz mit ODBC

Im Bereich der DDL- und DML-Anweisungen hat sich für die Anbindung relationaler und nicht-relationaler DBSe unter Windows ODBC (Open Database Connectivity) als Standard-API etabliert [Gei95].

Vorteile

Die Verwendung von ODBC hat gegenüber einer speziellen Schnittstelle neben den Vorteilen, die standardisierte Schnittstellen ganz allgemein besitzen (vgl. 5.2), noch weitere Vorteile:

- ODBC umfaßt gleichzeitig beide Sprachbereiche: DDL und DML.

- Die ODBC-APIs müssen nicht von den Anwendungsprogrammierern durch die Implementierung einer Zwischenschicht auf die entsprechenden DB-APIs der Plattformen abgebildet werden; die Abbildung wird von den bereits implementierten ODBC-Treibern übernommen (vgl. Anhang B).

Voraussetzungen

Um jedoch überhaupt ODBC im Hinblick auf die Portabilität von Managementanwendungen im Bereich DB-Anbindung einsetzen zu können, müssen auf allen Plattformen DBSe installiert sein, für die ODBC-Treiber existieren. Diese Voraussetzung ist für die drei gegebenen Plattformen erfüllt: es existieren ODBC-Treiber für Btrieve (allgemein ab Version 5.1), MS SQL Server (ab Version 4.21) und Paradox (ab Version 3.0) [Mor94].
Damit ist die Frage nach der Existenz eines Standards und seiner *potentiellen* Verwendung auf den Plattformen geklärt. Offen ist jedoch weiterhin, unter welchen Bedingungen ODBC *tatsächlich* eingesetzt werden kann.
Im folgenden wird für jede der drei Kategorien von DB-Anweisungen untersucht, ob und wenn ja unter welchen Voraussetzungen ODBC verwendet werden kann.

7.2.1 Zugriff auf Daten der Basisanwendungen

Auf jeder der drei Plattformen sind Basisanwendungen implementiert, die Daten sammeln und abspeichern - wie z.B. die Autodiscovery-Anwendung des HPOV Topologiemanagements oder die Inventory-Anwendung unter SMS - und andere, die auf die abgespeicherten Daten zugreifen wie beispielsweise die Layout-Anwendung des HPOV Topologiemanagers.
Aus diesem Grund sind auf jeder der Plattformen verschiedene DBSe implementiert: so unterstützt NMS das Dateiverwaltungssystem Btrieve und SMS das relationale DBS MS SQL Server. Unter HPOV sind sogar unterschiedliche DBSe für die Daten der verschiedenen Managementbereiche implementiert, so z.B. Paradox im Bereich Alarmmanagement oder ein spezielles proprietäres DBS im Topologiemanagementbereich.
Infolgedessen können im Hinblick auf die in *einem* Managementbereich eingesetzten DBSe der dazugehörigen Basisanwendungen zwei Szenarien auftreten (vgl. 7.2):

1. Für jedes der entsprechenden DBSe auf den Plattformen existiert ein ODBC-Treiber.

2. Für mindestens ein DBS des betrachteten Managementbereichs existiert kein ODBC-Treiber.

Designkonzept für das erste Szenario

Auf den DBSen wird ODBC eingesetzt: die Basisanwendungen greifen über die gleichen DB-Schnittstelle wie vorher auf die DB zu, während alle anderen (zusätzlichen) Anwendungen ODBC verwenden.
Auch wenn es auf den ersten Blick so aussieht, als ob der Einsatz von ODBC ohne weiteres möglich wäre - der entscheidende Schwachpunkt liegt hier im Spezifikationsteil der DB-Befehle:
ODBC verwendet SQL, SQL als relationale DB-Sprache spezifiziert Daten über Tabellennamen und Attributnamen des in einer DB implementierten DB-Schemas. Da die drei DBSe auf der Ebene von ODBC - auf der alle DBSe relational erscheinen - i.a. unterschiedliche Tabellen, Tabellennamen, Attribute und Attributnamen verwenden, ist auch keine einheitliche Datenspezifikation möglich. Eine Vereinheitlichung der Spezifikation wäre möglich mit Hilfe eines Precompilers, der oberhalb von ODBC eine entsprechende Namenanpassung vornimmt.

Designkonzept für das zweite Szenario

Hier bestünde die Möglichkeit, eine zweite Datenbank auf dem vorhandenen, ODBC unterstützenden DBS zu implementieren; in dieser DB würden dann die gleichen Daten gespeichert wie in der ursprünglichen DB, die ODBC nicht unterstützt. Auf diese zweite DB könnte dann mit ODBC zugegriffen werden:
Neben den Nachteilen, die auch im ersten Szenario auftreten, hat dieses Konzept weitere gravierende Nachteile:

- Die ursprüngliche DB kann nicht einfach gelöscht werden und die gleichen Daten müßten daher zweimal gespeichert werden. Der Grund dafür, daß die Daten auch in der ursprünglichen DB weiter erhalten bleiben müssen, ist einfach:
  Die Basisanwendungen der Plattformen ergänzen sich oft komplementär: so schreibt beispielsweise die HPOV Autodiscovery-Anwendung Daten in die DB, die wiederum von der Layout-Anwendung als Grundlage für die grafische Darstellung der Netztopologie verwendet werden (vgl. 2.3.2). Die abfragenden Basisanwendungen - wie die Layout-Funktion - bleiben jedoch weiterhin auf die ursprüngliche DB-Schnittstelle und damit DB angewiesen.

- Es muß ein Mechanismus zur wechselseitigen Datenübertragung zwischen den zwei DBen implementiert werden, der die Konsistenz und Aktualität der Daten gewährleistet.

7.2.2 Zugriff auf Daten anderer Managementanwendungen

Nicht nur Basisanwendungen, sondern auch andere Managementanwendungen können DBen generieren. So könnte beispielsweise eine Server-Manager-Anwendung - der SNI Server Manager legt beispielsweise in einem .INI-File als Konfigurationsinformationen die Servernamen, deren Adressen etc. ab - eine Datenbank für Konfigurationsinformationen einrichten und in ihr Daten über die einzelnen Server abspeichern.
Der Zugriff auf die Daten anderer Anwendungen ist hier plattformübergreifend zu verstehen d.h., auf allen Plattformen müssen Anwendungen mit derselben Zielrichtung installiert sein, also z.B. Server-Manager, die Konfigurationsinformationen abspeichern. Damit entsteht jedoch die gleiche Situation wie für den Zugriff auf die Daten der Basisanwendungen: die Anwendungen speichern ihre Daten in proprietäre DBen ab, auf die ein einheitlicher Zugriff nicht oder nur schwer möglich ist.
Eine Ausnahme bildet hier lediglich der Spezialfall, daß jede dieser Anwendungen ihre DB über ODBC *und* auf der Basis des gleichen DB-Schemas - was für eine portierbare ODBC-Anwendung zutrifft - aufgebaut hat. Dann ist auch der Zugriff über ODBC problemlos möglich.

7.2.3 Datenbankgenerierung durch Managementanwendungen

Der Bereich der DDL-Befehle wird auf den Plattformen eher stiefmütterlich behandelt:

- NMS:
  Wie schon erwähnt, stellt NMS ein DBS für alle Anwendungen zur Verfügung. Das implementierte DB-Schema reflektiert dabei das Netz über eine Sammlung von Tabellen, die Netzwerke, Segmente, Netz- und Systemkomponenten sowie deren Beziehungen untereinander beschreiben (vgl. [Nov93c]).
  Die bereitgestellten DB-APIs sind Teil der DML der DB (Get-, Set-, Enum-Befehle). Anwendungen, die diese APIs verwenden, können nur solche Daten abspeichern, für die Tabellen vorhanden sind. Eine Erweiterung des DB-Schemas ist nur direkt über die Btrieve DLL möglich; eine neuen Btrieve-Datei kann mit dem CREATE-Befehl erstellt werden [Sch92].

- HPOV:
  HPOV unterstützt zwar mehrere DBSe gleichzeitig, aber auch diese stellen jeweils nur DMLs zur Verfügung (vgl. [HP93]. Darüberhinaus ist auch keines dieser DBSe - im Gegensatz zu NMS - für die Speicherung von Konfigurationsinformationen über Netz- und Systemkomponenten vorgesehen.

Anwendungen, die mit diesen oder anderen Informationen arbeiten, müssen ihr eigenes DB-Schema mit Hilfe der Paradox-DLL implementieren.

- SMS:
  SMS unterstützt wie NMS nur ein DBS und auch hier kann ein neues DB-Schema nur mit Hilfe der SQL-Schnittstelle der SQL Server DB in eine DB umgesetzt werden; die SMS Data APIs lassen ebenso nur die Abfrage, Einfügen und Löschen zu (vgl. [SMS95c])

Der Einsatz von ODBC schafft im Bereich der DDL-Anweisungen problemlos eine einheitliche Schnittstelle; mit ODBC eingerichtete Datenbanken haben den zusätzlichen Vorteil einer einheitlichen DML-Schnittstelle.

7.3 Allgemeiner Lösungsansatz mit speziellen Datenbankschnittstellen

Abschnitt 7.2 hat gezeigt, daß der Zugriff von Managementanwendungen auf vorhandene Datenbanken anderer Anwendungen - seien es Basisanwendungen oder andere Managementanwendungen - über ODBC mit großen Schwierigkeiten verbunden ist.
Eine Alternative zum ODBC-Ansatz liegt in der Entwicklung spezieller, an die jeweiligen proprietären DB-Schnittstellen der Plattformen angepaßten, gemeinsamen Schnittstellen. Für die Entwicklung solcher Schnittstellen bieten sich drei Methoden an:

1. Die Bottom-up-Methode.

2. Die Top-down-Methode.

3. Die Verbindung von Bottom-up- und Top-down-Ansatz.

7.3.1 Bottom-up-Methode

Ansatz

Die Bottom-up-Methode stellt den heuristischsten Ansatz für die Entwicklung gemeinsamer Schnittstellen dar:
Die vorhandenen DB-Schnittstellen werden miteinander verglichen und es wird versucht, Funktionen mit jeweils identischer Funktionalität zu finden. Auf der Basis dieser Funktionen wird dann eine portierbare Schnittstelle spezifiziert.

Vorteil

Die Spezifikation der portierbaren Schnittstelle orientiert sich sehr stark an den vorhandenen Plattformschnittstellen. Aus diesem Grund ist der Aufwand für die Abbildung der einzelnen Funktionen auf die entsprechenden Plattformfunktionen gering.

Nachteile

Die Nachteile des Bottom-up-Verfahrens sind jedoch evident:

- Ein Vergleich von jeweils 60-80 Funktionen pro Plattform - eine durchaus übliche Größenordnung - kann alleine schon von der Anzahl der Funktionen überaus komplex und schwierig sein.

- I.a. implementieren die jeweiligen Schnittstellen unterschiedliche Funktionen, so daß es schwierig ist, eine portierbare Schnittstelle zu spezifizieren. Dieses Problem wird noch dadurch vergrößert, daß die DB-APIs einer Plattform oft Funktionen enthalten, die auch über andere Funktionen der gleichen Schnittstelle implementiert werden könnten. Für die Vereinheitlichung der Schnittstelle würde es jedoch genügen, einen Satz von Basisfunktionen zu spezifizieren, und nicht alle Funktionen, für die das möglich wäre, in die gemeinsame Schnittstelle zu übernehmen. Die gemeinsamen Basisfunktionen müßten dabei so gewählt werden, daß die Gesamtfunktionalität der Einzelschnittstellen dabei erhalten bleibt (vgl.5.2). Das Problem beim dabei ist, eine Menge von Basisfunktionen - im allgemeinen wird es mehrere Möglichkeiten geben - zu bestimmen.

7.3.2 Top-Down-Methode

Ansatz

Die Top-down-Methode simuliert die Vorgehensweise der Plattform-Entwickler: Ausgangspunkt ist jeweils die Frage, was die Schnittstelle leisten können soll. Übertragen auf eine DB-Schnittstelle heißt das:

1. Welches Informationsmodell soll die DB implementieren?
   Managementanwendungen einschließlich Basisanwendungen, die Daten sammeln und speichern, ermöglichen dadurch verschiedene Sichten auf das Netz: der Topologiemanager -sofern er eine Autotopologyfunktion implementiert - unterstützt z.B. eine globale Sicht auf das Netz, während der Konfigurationsmanager die Sicht auf einzelne Komponenten unterstützt.
   Voraussetzung ist jeweils, daß das zugrundeliegende Informationsmodell der Plattform diese verschiedenen Beschreibungsebenen auch unterstützt (dazu

auch [Heg93]). Ebenfalls nach [Heg93] umfaßt der Begriff Informationsmodell alle Konzepte zur Beschreibung managementrelevanter Objekte und Informationen; d.h in dem Informationsmodell (genauer Informationsteilmodell), das die DBen implementieren, müssen jeweils alle „sichtrelevanten "Managementinformationen beschrieben werden.

2. Welche Abfrage- und Änderungsoperationen sollen möglich sein?

Vorteil

Der Vorteil beim Top-down-Vorgehen ist, daß die portierbare Schnittstelle systematischer entwickelt werden kann:
ein Problem beim Bottom-up-Vorgehen bestand darin, eine Menge von Basisfunktionen zu finden (vgl. 7.3.1). Dieses Problem resultiert in erster Linie daraus, daß das von den Plattform-Entwicklern zugrundegelegte Informationsmodell einer DB hinter der jeweiligen DB-Schnittstelle „versteckt"ist. Ist dagegen das Informationsmodell bekannt, läßt sich relativ leicht eine Menge von Basisfunktionen bestimmen.

Nachteile

Das Top-down Verfahren hat jedoch auch zum Teil gravierende Nachteile:

- Hoher Aufwand für die Spezifizierung einer Schnittstelle (Entwicklung eines Informationsmodells, Ableiten einer Menge von Basisfunktionen).

- Die spezifizierte, portierbare Schnittstelle basiert auf einem Informationsmodell, daß u.U. stark von den Informationsmodellen, die die Plattform-DBen implementieren, abweicht.
  Infolgedessen kann die Abbildung der Schnittstellenfunktionen auf die Plattformfunktionen schwierig bis unmöglich sein.

7.3.3 Verbindung von Bottom-up- und Top-Down-Ansatz

Sowohl die Top-down- als auch die Bottom-up-Methode haben gravierende Nachteile; eine i.a. effizientere Methode bietet ein Vorgehen, das beide Methoden kombiniert:
In vielen Managementbereichen - u.a. im Bereich Topologiemanagement - ist das (den DBen) zugrundegelegte Informationsmodell bis zu einem gewissen Grad generisch; das bedeutet, daß Informationsmodelle für diese Bereiche weitgehend übereinstimmen (vgl. 10.3.1).
Mit Hilfe der Top-down-Methode läßt sich in diesen Fällen zunächst ein *grobes* Informationsmodell entwerfen, das die Managementinformationen soweit spezifiziert wie diese als generisch gelten können; die „Feinspezifikation"erfolgt anschließend über die Bottom-up-Methode.

7.4 Fazit

Der Einsatz von ODBC im Hinblick auf die Portierbarkeit von DB-Anbindungen beschränkt sich auf die Managementanwendungen, die zuvor auch ihr eigenes DB-Schema über ODBC implementiert haben (7.2).
Für alle Managementanwendungen, die auf „Fremd"-DBen zugreifen, muß jedoch eine spezielle, auf die vorhandenen Plattformschnittstellen abgestimmte Schnittstelle definiert werden. Die gravierenden Nachteile der reinen Bottom-up- bzw. Top-Down-Methode können unter bestimmten Voraussetzungen durch ein Vorgehen, das beide Methoden kombiniert, vermieden werden.

Kapitel 8

Die SNMP-Schnittstelle

Das Internet-Management bildet aktuell die Basis für die meisten herstellerübergreifenden Managementlösungen [Heg93]. Das Simple Network Management Protocol (SNMP) als Kern des Internet-Managements - das daher auch als SNMP-Management bezeichnet wird [Hei94] - hat sich damit als De-facto-Standard etabliert [Jan93]. Dementsprechend wird SNMP auch auf den aktuell untersuchten Plattformen von NMS und HPOV eingesetzt.
In diesem Kapitel werden zunächst die wesentlichen Spezifikationen für das SNMP-Management - das sogenannte Internet-standard Network Management Framework - kurz vorgestellt, bevor anschließend auf die SNMP-Schnittstellen - das sind die APIs für die SNMP-Kommunikation - von NMS und HPOV eingegangen wird.
Dabei zeigt sich, daß sich die SNMP-Schnittstellen der beiden Plattformen stark voneinander unterscheiden: HPOV unterstützt das Standard-API WinSNMP (vgl. [Nat94]), während NMS ein proprietäres SNMP-API implementiert (vgl. [Nov93f]). Anschließend wird auf der Basis von WinSNMP ein Portabilitätskonzept für die SNMP-Schnittstelle von Managementanwendungen entwickelt.
Dieses Portabilitätskonzept schließt jedoch nur die beiden Plattformen NMS und HPOV ein, da unter SMS keine SNMP-Schnittstelle implementiert ist. Momentan ist eine SNMP-Kommunikation unter Windows NT nur mit Hilfe der in Visual C++ (ab Version 2.0) enthaltenen Laufzeitbibliotheken snmp.dll und mgmtapi.dll möglich. Eine Abbildung der WinSNMP-Funktionen auf diese DLL-Funktionen erübrigt sich, da nach Aussage von Microsoft in einer der zukünftigen Windows NT-Versionen ebenfalls eine WinSNMP-Implementierung enthalten sein wird.

8.1 Internet-standard Network Management Framework

Grundlage jedes Netzmanagements ist einerseits die Beschreibung aller zu verwaltenden Managementobjekte (Managed Objects) aus Managementsicht innerhalb des sogenannten Informationsmodell (vgl. auch [Heg93]), andererseits die Spezifizierung eines Kommunikationsprotokolls, mit der diese Managementinformationen abgefragt werden können.

Das Informationsmodell

Unter dem Begriff Informationsmodell werden alle Konzepte zur Beschreibung von Managementinformationen und die konzeptuelle Beschreibung aller managementrelevanten Objekte und Informationen zusammengefaßt (vgl. [Heg93]). Das Internet-Informationsmodell wird durch die Requests for Comment RFC 1155, RFC 1157, RFC 1213 und RFC 1212 festgelegt (vgl. [RFC90b], [RFC90a], [RFC91a] und [RFC91c]). Im einzelnen werden dabei die folgenden Dinge festgelegt (vgl. [Jan93], [Heg93]):

- Die Beschreibungssprache:
  ASN.1 (Abstract Syntax Notation One) als standardisierte und systemübergreifend gültige Beschreibungssprache wird ASN.1 (Abstract Syntax Notation One) festgelegt (vgl. [RFC90b]); damit wird a priori keine Implementierung der beschriebenen Managementinformationen festlegt (vgl. [Heg93]).
- Die Beschreibungssyntax:
  die Beschreibungssyntex von Managementinformationen wird durch das OBJECT-TYPE MAKRO festgelegt (vgl. [Ros93],[RFC91a]).
- Die Identifikation von Managementinformationen:
  die Managementinformationen werden über die Objektidentifikatoren des Internet-Namenbaums weltweit eindeutig identifiziert. (RFCs 1155 und 1213)
- Die Managementobjekte und -informationen:
  Managementobjekte und -informationen werden in den Internet-MIBs, insbesondere der Standard MIB MIB-II (RFC1213) festgelegt.
- Die Kodierung der Managementinformationen bei der Übertragung durch das SNMP wird durch die Basic Encoding Rules (BER) von ASN.1 festgelegt.

Das Managementprotokoll

Als Managementprotokoll wird SNMP verwendet; die Protokollspezifikation für SNMP (RFC 1157) legt fest (vgl. [Jan93]):

- Die Protokollformate und die Regeln für ihren Austausch.
- Die Managementoperationen z.B. GetRequest, SetRequest.
- Regeln für den Zugriff auf Managementobjekte; darunter fallen die Authentikation über Community Name und ACCESS-Informationen.

Weitere Einzelheiten zu SNMP, ASN.1, BER etc. finden sich in [Hei94] und [Jan93].

8.2 Die SNMP-Schnittstellen auf den Plattformen

In diesem Abschnitt sollen die SNMP-Schnittstellen von HPOV und NMS kurz beschrieben werden; dabei sollen Unterschiede aufgezeigt und dadurch die Notwendigkeit für die Entwicklung eines Portabilitätskonzepts begründet werden.
HPOV stellt Anwendungen eine Implementierung der WinSNMP/Manager API zur Verfügung; die WinSNMP API spezifizieren eine Standard-Schnittstelle für die SNMP-Kommunikation (vgl. Anhang A, [Nat94]).
Demgegenüber implementiert NMS keine direkte Schnittstelle für die SNMP-Kommunikation; die SNMP-Anweisungen der Anwendungen werden statt dessen mittelbar über den SNMP Data Server von NMS durchgeführt.
Die Anwendung baut dazu eine entsprechende, sogenannte SNMP_REQUEST Struktur auf und übergibt diese über eine DDE-Verbindung (Dynamic Data Exchange, ein Windows-Kommunikationsprotokoll) an den Data Server. Analog werden Response-PDUs und Traps über SNMP_REQUEST Strukturen an die Anwendung zurückgegeben.
Eine SNMP_REQUEST Struktur enthält u.a. alle für den Aufbau einer SNMP-Nachricht notwendigen Bausteine (SNMP-PDU, Community String und Version) bereits BER-kodiert; die für die Kodierung notwendigen Funktionen stellt NMS zur Verfügung. Darüberhinaus enthält die SNMP_REQUEST Struktur auch alle notwendigen transportprotokollrelevanten Angaben z.B. Werte für die Anzahl der Retries und für die Länge des Timeout-Zeitintervalls.
Wesentliche Unterschiede hinsichtlich der implementierten Funktionalität zwischen den SNMP-Schnittstellen sind u.a.:

- Die Elemente einer SNMP-Nachricht müssen unter NMS von den Anwendungen selbst (unter Zuhilfenahme der entsprechenden NMS-Funktionen) nach den BER kodiert werden; unter HPOV wird die entsprechende Kodierung von der WinSNMP-Implementierung durchgeführt.

- Der Speicherplatz, der für den Aufbau einer SNMP-PDU bzw. SNMP-Nachricht benötigt wird, muß von einer NMS-Anwendung - im Gegensatz zu Anwendungen unter HPOV - selbst allokiert werden.

- Während die HPOV-Schnittstelle auch Default-Werte für Retries und Timeout unterstützt (vgl. 8.3.2 Local Database Functions), muß eine Anwendung unter NMS bei jeder SNMP-Anfrage entsprechende Werte mit übergeben.

8.3 Das Portabilitätskonzept

8.3.1 Spezifikation der SNMP-Schnittstelle

Als Spezifikation einer SNMP-Anwendungsschnittstelle bietet sich WinSNMP aus zwei Gründen an:

1. WinSNMP ist ein Industriestandard; es ist steht daher zu erwarten, daß auch andere Plattform-Hersteller in Zukunft ihre SNMP-Schnittstellen entsprechend implementieren.

2. WinSNMP ist auf HPOV bereits implementiert, so daß hier kein Aufwand für die Abbildung auf die SNMP-Schnittstelle von HPOV getrieben werden muß.

8.3.2 Abbildung auf NMS

WinSNMP unterscheidet insgesamt sechs Funktionengruppen:

1. Local Database Functions

2. Communication Functions

3. Entity/Context Functions

4. PDU Functions

5. Variable Binding Functions

6. Utility Functions

Jede dieser Funktionengruppen wird im folgenden einzeln behandelt; dabei werden jeweils in einem ersten Schritt nur die Funktionen abgebildet, die für die SNMP-Kommunikation unabdingbar sind. In einem zweiten Schritt wird dann jeweils kurz auf die Abbildung der übrigen Funktionen eingegangen.
In den folgenden Abschnitten wird immer wieder und auch sehr detailliert Bezug auf die BER genommen; eine genaue Darstellung der BER würde jedoch über den Rahmen dieser Arbeit hinausgehen. Es sei deshalb nochmals auf entsprechende Literatur in [Hei94], [Jan93] und [Ros93] verwiesen.
Ebenso wird die Verwendung der Routinen für die globale Heap-Verwaltung (GlobalAlloc, GlobalFree, GlobalLock etc.) nicht weiter begründet; eine Erläuterung dazu findet sich in Abschnitt 7.1.

Variable Binding Functions

Hierunter faßt WinSNMP alle Funktionen zusammen, die unmittelbar mit den Variable Bindings einer SNMP-PDU zusammenhängen. WinSNMP spezifiziert sieben Variable Binding Functions, von denen jedoch nur fünf unbedingt erforderlich sind:
SnmpCreateVbl und SnmpSetVbl zum Aufbau einer Variable Binding List (VBL), SnmpCountVbl und SnmpGetVbl zur Analyse einer VBL sowie SnmpFreeVbl zur Freigabe aller im Zusammenhang mit einer VBL belegten Ressourcen.

SnmpCreateVbl

Syntax:

HSNMP_VBL	SnmpCreateVbl(
IN HSNMP_SESSION	session,
IN smiLPCOID	name,
IN smiLPCVALUE	value);

Parameter:	Beschreibung:
session	Session-Handle (vgl. Communication Functions)
name	NULL oder Zeiger auf OID-Struktur (Object Identifier)
value	NULL oder Zeiger auf value-Struktur

Rückgabewert:
Im Erfolgsfall wird ein Handle auf eine evtl. initialisierte VBL zurückgegeben, ansonsten die Fehlermeldung SNMPAPI_FAILURE.
Funktion:
SnmpCreateVbl allokiert Speicher für eine VBL, belegt diesen Speicherplatz evtl. (falls name ungleich NULL) mit der nach den BER kodierten, übergebenen Variable Binding (VB) und gibt einen Handle auf die so erzeugte VBL zurück.
Abbildung:
Die DLL für NMS muß nun diese Funktionalität mit Hilfe der vorhandenen NMS-Schnittstelle nachimplementieren:
1.Allokierung von Speicherplatz für eine VBL:
Mit GlobalAlloc werden 512 Byte auf dem globalen Heap für die VBL belegt; 512 Byte ist dabei die Maximalgröße, die eine VBL unter NMS besitzen darf (Konstante MAX_PDU_DATA_SIZE in n-snmp.h).
2.Falls der Parameter name ungleich NULL ist, muß die beim Aufruf übergebene VB (bestehend aus name und value) entsprechend den BER kodiert und in die

VBL eingetragen werden:
Dazu muß die NMS-Funktion build_varbind aufgerufen werden, die genau diese Funktionen implementiert. build_varbind benötigt drei Eingabeparameter:

- eine Adresse innerhalb des VBL-Speicherbereichs, ab der die kodierte VB eingetragen werden soll; für SnmpCreateVbl ist dies die mit GlobalLock gelieferte Anfangsadresse des VBL-Puffers.
- die Länge des freien VBL-Puffers in Byte, hier also 512.
- einen Zeiger auf eine NMS VB-Struktur:
  diese Struktur besteht aus fünf Komponenten: OID der Variable, Länge des OID, Syntaxtyp des Datums, Länge des Datums und dem Datum selbst. Diesen Komponenten können direkt die entsprechenden Werte aus den übergebenen Strukturen name (OID, Länge OID) und value (Syntaxtyp, Datumslänge,Datum) zugewiesen werden.

3.Rückgabewerte:
Falls die Funktion erfolgreich ausgeführt wurde, gibt sie den VBL-Handle aus GlobalAlloc zurück; falls nicht wird SNMPAPI_FAILURE(0) zurückgegeben. In diesem Fall kann über SnmpGetLastError detailliertere Information über den aufgetretenen Fehler erhalten werden.

SnmpSetVb()

Syntax:

```
SNMPAPI_STATUS          SnmpSetVb(
      IN HSNMP_VBL      vbl,
      IN smiUINT32      index,
      IN smiLPCOID      name,
      IN smiLPCVALUE    value);
```

Parameter:	Beschreibung:
vbl	Identifiziert die VBL
index	Bezeichnet die Stelle in der VBL an der die VB eingefügt werden soll (update) bzw. ist 0, falls VB an die VBL angehängt werden soll
name	Zeiger auf OID-Struktur
value	Zeiger auf value-Struktur

Rückgabewert:
Falls erfolgreich, gibt die Funktion die Position (index) zurück, an der die VB

eingefügt bzw. angehängt wurde.
Funktion:
Fügt die kodierte VB an der gewünschten Stelle in die spezifizierte VBL ein.
Abbildung:
Da dies für die übliche SNMP-Kommunikation ausreicht, wird vorerst nur das Anhängen einer VB an eine bestehende VBL betrachtet:

1. Länge der VBL berechnen:
   hierzu muß eine Funktion VblLength implementiert werden, die zu einer gegebenen VBL die Länge des belegten VBL-Puffers berechnet. Diese Funktion ist in Kenntnis der ASN.1 BER leicht zu implementieren:
   Ein Zeiger wird mit GlobalLock auf den Anfang des VBL-Puffers gesetzt;falls der Inhalt des referenzierten Byte ungleich 0 ist, ist die VBL nicht leer. Je nach Länge der kodierten VBL (ohne Header!) ist deren Länge im 2. bzw. 3. und 4. Byte des Headers kodiert. Bei der Dekodierung der Länge ist zu beachten, daß die Zahldarstellung gemäß den BER das Big Endian Format verwendet, so daß eine Konvertierung in das Little Endian Format nötig ist (Intel-Format). Die Gesamtlänge der VBL ergibt sich aus der Summe dieser Länge und des VBL-Headers (2 bzw.4 Byte). Der entsprechend hochgezählte Puffer-Zeiger zeigt dann auf den freien VBL-Puffer.

2. Prüfen, ob der freie Puffer für die kodiert VB ausreicht:
   globalen Puffer für VB allokieren, VB mit build_varbind kodiert in diesen Puffer schreiben, Länge der kodierten VB wie unter 1. angedeutet berechnen und mit der freien Pufferlänge aus 1. vergleichen.

3. Falls der freie Puffer ausreicht, wird die in 2. kodierte VB ab der aus 1. bekannten Anfangsadresse in den freien Puffer kopiert.

4. Die im VBL-Header kodierte VBL-Länge muß aktualisiert werden.

Rückgabewert:
im Erfolgsfall wird die Anzahl der im VBL-Puffer kodierten VBs zurückgegeben. Dazu muß eine Funktion VblCount implementiert werden: nach dem Header der VBL sind die einzelnen VBs kodiert; auch deren Länge ist jeweils im 2. bzw. 3. und 4. Byte der VB-Header gespeichert. Die sich so ergebenden Gesamtlängen für die VBs ermöglichen innerhalb der VBL von VB zu VB zu springen und diese zu zählen bis keine VB mehr folgt bzw. der Pufferbereich zu Ende ist.

SnmpCountVbl()

Syntax:

SNMPAPI_STATUS	SnmpCountVbl(
IN HSNMP_VBL	vbl);
Parameter:	Beschreibung:
vbl	Identifiziert die VBL

Rückgabewert:
Die Funktion gibt im Erfolgsfall die Anzahl der in der VBL enthaltenen VBs zurück.
Abbildung:
Aufruf der Funktion VblCount (vgl. SnmpSetVb).

SnmpGetVb()

Syntax:

SNMPAPI_STATUS	SnmpGetVb(
IN HSNMP_VBL	vbl,
IN smiUINT32	index,
OUT smiLPOID	name,
OUT smiLPVALUE	value);
Parameter:	Beschreibung:
vbl	Identifiziert die VBL
index	Identifiziert die Position der VB in der VBL
name	Zeiger auf OID-Struktur
value	Zeiger auf value-Struktur

Rückgabewert:
SNMPAPI_SUCCESS im Erfolgsfall, sonst SNMPAPI_FAILURE.
Funktion:
SnmpGetVb gibt den Namen (OID) und den Wert einer Objektvariablen (VB) über die bereitgestellten Strukturen name und value an den Aufrufer zurück.
Abbildung:

1. Abbildung der spezifizierten VB auf eine NMS VB-Struktur:
   Aufruf der NMS-Funktion parse_varbind; parse_varbind benötigt folgende Parameter:

   - einen Zeiger auf die betroffene VB innerhalb der spezifizierten VBL. Dieser Zeiger kann durch „Sprünge“ von VB zu VB und entsprechendes Hochzählen des Zeigers auf den VBL-Anfang bis hin zur betroffenen VB (index) gewonnen werden.
   - die Länge des belegten Pufferbereichs der VBL ab der durch den obigen Zeiger bestimmten Position. Mit der oben angedeuteten Methode kann auch gleichzeitig die Länge des belegten Puffers vor dem Zeiger bestimmt werden, die Gesamtlänge des belegten VBL-Bereichs erhält man mit VblLength (vgl. SnmpSetVb).
   - einen Zeiger auf eine NMS VB-Struktur.

2. Die Komponenten der NMS VB-Struktur können direkt den entsprechenden Komponenten der Ausgabestrukturen name und value zugewiesen werden (vgl. SnmpSetVb).

SnmpFreeVbl()

Funktion:
SnmpFreeVbl gibt alle in Verbindung mit einer bestimmten VBL belegten Ressourcen wieder frei.
Abbildung:
Aufruf von GlobalFree.

Erweiterung der Minimalimplementierung für die Variable Binding Functions

Neben den bisher vorgestellten Variable Binding Functions spezifiziert WinSNMP noch zwei weitere Funktionen:

- **SnmpDuplicateVbl()**: kopiert die übergebene VBL und liefert einen Handle auf die Kopie zurück.

- **SnmpDeleteVb()**: löscht die über einen Index bestimmte VB in einer spezifizierten VBL.

Sowohl diese beiden Funktionen als auch die Erweiterung von SnmpSetVb auf beliebige index-Werte (bisher nur 0) erfordern konzeptuell nichts neues, sondern lassen sich im wesentlichen durch das Kopieren von Pufferbereichen implementieren.

PDU Functions

WinSNMP spezifiziert fünf PDU Functions, von denen drei unbedingt erforderlich sind: SnmpCreatePdu zum Aufbau einer Request-PDU, SnmpGetPduData zur Analyse und SnmpFreePdu zur Freigabe der Ressourcen.

SnmpCreatePdu()

Syntax:

HSNMP_PDU	SnmpCreatePdu(
IN HSNMP_SESSION	session,
IN smiINT	PDU_type,
IN smiINT32	request_id,
IN smiINT	error_status,
IN smiINT	error_index,
IN HSNMP_VBL	vbl);
Parameter:	Beschreibung:
session	Session-Handle
PDU_type	bestimmt den PDU-Typ des Requests (z.B. GetRequest)
request_id	PDU-Identifier, anhand dessen die Anwendung die zu einer Request-PDU gehörige Response-PDU identifizieren kann
error_status	NULL für SNMPv1-Requests
error_index	NULL für SNMPv1-Requests
vbl	VBL-Handle

Rückgabewert:
SnmpCreatePdu gibt einen Handle auf die erzeugte PDU zurück.
Funktion:
SnmpCreatePdu baut aus den Eingabeparametern eine SNMP-PDU auf; je nach WinSNMP-Implementierung ist diese PDU bereits ASN.1 BER kodiert oder wird erst mit der Aufforderung zum Senden (vgl. SnmpSendMsg) kodiert.
Abbildung:

1. Allokierung von Speicher für eine NMS SNMP_REQUEST Struktur:
   Für die SNMP_REQUEST Struktur müssen mit GlobalAlloc 720 Byte Speicher allokiert werden. Notwendig wird diese Allokierung dadurch, daß es innerhalb der SNMS_REQUEST Struktur keine genaue Entsprechung für die SNMP-PDU gibt: die logisch unterscheidbare Datenstruktur, die hier der SNMP-PDU am nächsten kommt, ist die DATA_PDU Struktur. Diese

Struktur enthält jedoch neben den Komponenten für die request_id, error_status, error_index und VBL auch Verwaltungskomponenten für den NMS Data Server, die den VBL-Puffer beschreiben: Anzahl der enthaltenen VBs, Gesamtlänge des Puffers und die Länge des belegten Teils.
Dazu kommt, daß die request_id unter NMS nicht durch die Anwendungen zugewiesen wird, sondern durch den Data Server. Um die Requests und Responses in Relation zueinander setzen zu können, verwenden die Applikationen unter NMS das magic-Feld innerhalb der SNMP_REQUEST Struktur.
Zu bemerken ist hier, daß diese Allokierung bereits durch die Funktion SnmpCreateVbl vorgenommen werden kann. Bei allen VBL-Operationen muß dann der Offset von 208 Byte innerhalb der SNMP_REQUEST Struktur zur VBL berücksichtigt werden (vgl. SnmpCreateVbl).

2. Belegung der SNMP_REQUEST Struktur:
die Werte der übergebenen Parameter können dann den entsprechenden Feldern innerhalb der Request Struktur zugewiesen werden, wobei die übergebene request_id jedoch dem magic-Feld zugewiesen wird. Zusätzlich müssen jedoch noch die unter 1. erwähnten Verwaltungskomponenten belegt werden: die Gesamtlänge des Puffers ist immer 512, der belegte Anteil ergibt sich mit VblLength und die Anzahl der VBs mit SnmpCountVb.

SnmpGetPduData

Syntax:

```
SNMPAPI_STATUS          SnmpGetPduData(
      IN HSNMP_PDU          PDU,
      OUT smiLPINT          PDU_type,
      OUT smiLPINT32        request_id,
      OUT smiLPINT          error_status,
      OUT smiLPINT          error_index,
      OUT LPHSNMP_VBL       vbl);
```

Parameter:	Beschreibung:
PDU	PDU-Handle
PDU_type	bestimmt den PDU-Typ
request_id	PDU-Identifier
error_status	error-status der Response-PDU
error_index	error-indices der Response-PDU
vbl	Zeiger auf VBL-Handle

Funktion:
SnmpGetPduData gibt bestimmte Daten aus der spezifizierten PDU über die

Ausgabeparameter an den Aufrufer zurück.
Abbildung:
Die Funktion SnmpGetPduData unterstützt nicht nur die Analyse von Response-PDUs (wie man aus den Ausgabeparametern schließen könnte), sondern auch die von Traps: die WinSNMP-Spezifikation schreibt vor, daß eine WinSNMP-Implementierung alle Traps als SNMPv2 Traps an die Anwendung übergibt (vgl [Nat94]). Das bedeutet, daß die in einem SNMPv1 Trap vorhandenen Felder für time-stamp, enterprise und Trapinformationen von der VBL übernommen werden.
NMS liefert sowohl für Responses als auch für Traps eine SNMP_Request Struktur über DDE an die Anwendung zurück.
Die Abbildung muß daher nicht nur die entsprechenden Werte der Request Struktur den Ausgabeparametern zuweisen, sondern für Traps auch die VBL entsprechend den in RFC 1452 Abschnitt 3.1.2 festgelegten Migrationsvorschriften von SNMPv1 nach SNMPv2 ergänzen (vgl. [RFC93a]). Dabei ist zu beachten, daß die erweiterte VBL evtl. mehr als 512 Byte Speicherplatz benötigt.

SnmpFreePdu()

Funktion:
Gibt den von der spezifizierten PDU belegten Speicherplatz wieder frei.
Abbildung:
Aufruf von GlobalFree wie gehabt.

Erweiterung der Minimalimplementierung für die PDU-Funktions

Neben den behandelten PDU-Funktionen spezifiziert WinSNMP noch **SnmpSetPduData()** zur Aktualisierung einer bestimmten PDU und **SnmpDuplicatePdu()**, um eine PDU zu kopieren. Beide Funktionen erfordern im wesentlichen nichts als Neuzuweisungen an Komponenten einer Request Struktur und das Kopieren von Speicherbereichen.

Communication Functions

Alle der von WinSNMP spezifizierten Communication Functions sind für die SNMP-Kommunikation unabdingbar: SnmpStartup, daß die Implementierung auffordert, alle notwendigen Initialisierungen und Allokationen durchzuführen sowie als Gegenstück SnmpCleanup. SnmpOpen führt alle Initialisierungen durch, die zur Herstellung der Kommunikationsbereitschaft notwendig sind, SnmpClose, macht dies wieder rückgängig; außerdem die Funktionen SnmpSendMsg, um eine SNMP-Nachricht zu senden und SnmpRecvMsg, um eine empfangene SNMP-Nachricht von der WinSNMP-Implementierung zu übernehmen. Mit der Funktion

SnmpRegister kann sich die Applikation registrieren lassen, um Traps zugestellt zu bekommen.

SnmpStartup

Syntax:

```
HSNMPAPI_STATUS            SnmpStartup(
      OUT smiLPUINT32      nMajorVersion,
      OUT smiLPUINT32      nMinorVersion,
      OUT smiLPUINT32      nLevel,
      OUT smiLPUINT32      nTranslateMode,
      OUT smiLPUINT32      nRetransmitMode);
```

Parameter:	Beschreibung:
nMajorVersion	Enthalten die Versionsnummer der
nMinorVersion	implementierten WinSNMP APIs
nLevel	Variable für SNMP-Support der Implementierung
nTranslateMode	Variable für Translate Mode(Default)
nRetransmitMode	Variable für Retransmission Mode

Funktion:
SnmpStartup führt alle Initialisierungen und Allokationen durch die für die Benutzung der WinSNMP Funktionen vorausgesetzt werden. Die Funktion liefert zugleich Informationen über die WinSNMP-Implementierung selbst: Version der implementierten WinSNMP APIs, „Level of SNMP Support" und die Default-Werte für Translate und Retransmission Mode.
Abbildung:
die NMS-Version der Funktion muß hier alle Variablen und Tabellen, die die NMS DLL verwendet allokieren und initialisieren: Variablen für Retransmission und Translate Mode, für die WinSNMP API Version und Support-Level,die Tabellen für die SnmpStrToEntity und SnmpStrToContext Funktionen. Die Allokierung erfolgt dabei wie üblich durch GlobalAlloc.

SnmpCleanup()

SnmpCleanup sorgt für die Freigabe aller Ressourcen, die durch SnmpStartup belegt worden sind. Die NMS-Version der Funktion muß dementsprechend alle allokierten Speicherbereiche mit LocalFree wieder freigeben.

SnmpOpen()

Syntax:

HSNMP_SESSION	SnmpOpen(
IN HWND	hWnd,
IN UINT	wMsg);

Parameter:	Beschreibung:
hWnd	Fensterhandle der Windowsanwendung
wMsg	identifiziert die notification message für den Aufruf der Callback-Funktion

Rückgabewert:
SnmpOpen gibt einen Handle für die Session zurück.
Abbildung:
die Kommunikationsbereitschaft der Anwendung wird unter NMS durch die Einrichtung eines DDE-Kommunikationskanals zum SNMP Data Server hergestellt:

1. Anmeldung der Applikation bei der DDEML mit DdeInitiate.
2. Aufbau der Verbindung zum SNMP Data Server mit DdeConnect; der Rückgabewert von DdeConnect - ein sogenannter Konversationshandle - ist gleichzeitig auch der Rückgabewert für SnmpOpen.
3. Einrichten des Kommunikationskanals (hot link) mit DdeClientTransaction.

SnmpClose()

Abbildung:
die DDE-Verbindung zum Data Server wird mit DdeDisconnect geschlossen und mit DdeUninitialize meldet sich die Applikation bei der DDEML ab.

SnmpSendMsg()

Syntax:

SNMPAPI_STATUS	SnmpSendMsg(
IN HSNMP_SESSION	session,
IN HSNMP_ENTITY	srcEntity,
IN HSNMP_ENTITY	dstEntity,
IN HSNMP_CONTEXT	context,
IN HSNMP_PDU	pdu);

Parameter:	Beschreibung:
session	Session-Handle
srcEntity	von Bedeutung nur in SNMPv2-Umgebung, sonst no-op
dstEntity	Handle für Zieladresse
context	Handle für Community
pdu	Handle für Request-PDU

Funktion:
SnmpSendMsg sendet die spezifizierte PDU an die angegebene Zieladresse.
Abbildung:

1. Belegung der SNMP_REQUEST Struktur:
   zunächst müssen die nach dem Aufbau der SNMP-PDU (vgl. SnmpCreatePdu) noch fehlenden Felder der über den PDU-Handle bestimmten Request Struktur belegt werden; außer den Feldern für die Zieladresse und die Community - die nach den BER kodiert übergeben werden müssen - sind dies:

   - die Felder für localAddr und PollFlag. Beide werden auf Null gesetzt.
   - die Felder für retries und timeout. Beide können durch Default-Werte bestimmt werden, die entweder als globale Variablen der DLL implementiert sind oder innerhalb der von WinSNMP vorgeschlagenen LocalDatabase (vgl. Local Database Functions). In letzterem Fall werden für jede Zieladresse eigene Default-Werte angegeben.

2. Senden:
   mit DdeTransaction wird die SNMP_Request Struktur über einen Handle an den Data Server übergeben.

SnmpRecvMsg()

Syntax:

```
SNMPAPI_STATUS              SnmpRecvMsg(
       IN HSNMP_SESSION      session,
       OUT HSNMP_ENTITY      srcEntity,
       OUT HSNMP_ENTITY      dstEntity,
       OUT HSNMP_CONTEXT     context,
       OUT HSNMP_PDU         pdu);
```

Parameter:	Beschreibung:
session	Session-Handle
srcEntity	Handle für Agentenadresse
dstEntity	Handle für Zieladresse
context	Handle für Community
pdu	Handle für Request-PDU

Funktion:
eine Anwendung wird über die in SnmpOpen() angegebene notification message über den Empfang einer Response bzw. eines Traps durch die WinSNMP-Implementierung benachrichtigt. In Reaktion auf diese Nachricht muß die Applikation die Funktion SnmpRecvMsg aufrufen; diese übernimmt dann die empfangene Response bzw. den empfangenen Trap von der Implementierung.
Abbildung:
Unter NMS wird jede empfangene Response und jeder empfangene Trap vom Data Server in eine SNMP_Request Struktur umgewandelt. Die DDEML ruft dann die Callback-Funktion der Applikation auf und übergibt ihr gleichzeitig einen Handle auf die SNMP_Request Struktur.
Innerhalb der Callback-Funktion muß dann ebenso wie unter HPOV die NMS-Version der Funktion SnmpRecvMsg aufgerufen werden. Den Ausgabeparametern werden die Werte der entsprechenden Komponenten der Request Struktur zugewiesen; pdu wird ein Zeiger auf den Handle für die Request Struktur zugewiesen. (wie in SnmpCreatePdu gibt es unter NMS kein Gegenstück zur WinSNMP-PDU).

SnmpRegister()

Syntax:

```
SNMPAPI_STATUS          SnmpRegister(
      IN HSNMP_SESSION    session,
      IN HSNMP_ENTITY     srcEntity,
      IN HSNPM_ENTITY     dstEntity,
      IN HSNMP_CONTEXT    context,
      IN smiLPCOID        notification,
      IN smiUINT32        status);
```

Parameter:	Beschreibung:
session	Session-Handle
srcEntity	no-op in SNMPv2-Umgebung
dstEntity	Agentenadresse
context	nur für SNMPv2-Umgebung

notification	Trap-OID
status	anmelden/abmelden für Trap-Empfang

Funktion:
SnmpRegister() meldet eine Anwendung für den Empfang von Traps bei der WinSNMP-Implementierung an bzw. ab. Dabei können sowohl die Agenten (Trap-Sender) als auch die in Frage kommenden Traps festgelegt werden.
Auch hier ist zu beachten, daß die Spezifikation der Funktion auf eine SNMPv2-Umgebung ausgelegt ist; nur hier sind Trap-OIDs festgelegt.
Abbildung:
Für die SNMP-Kommunikation muß nicht die volle Funktionalitt dieser Funktion implementiert werden; es reicht aus, die Funktion so zu implementieren, daß sich die Anwendung für den Empfang aller Traps an- und abmelden kann. Eine Filterung der Traps nach Agenten und Traptyp kann auch die Anwendung durchführen. Im entsprechenden Aufruf von SnmpRegister werden dabei die Werte für srcEntity, dstEntity, context und notification auf NULL gesetzt.
Anmeldung:
die Anmeldung erfolgt wiederum über den Aufbau eines DDE hot links mit entsprechendem DDE-Topic und DDE-Item zum SNMP Data Server, also über Aufrufe von DdeConnect und DdeClientTransaction.
Abmeldung:
die Abmeldung erfolgt wiederum über DdeDisconnect-Aufruf.

Local Database Functions

Der Begriff „Local Database "im Zusammenhang mit der WinSNMP Spezifikation bedeutet nicht, daß eine WinSNMP-Implementierung ein „richtige" Datenbank verwaltet, sondern bezieht sich auf die SNMP-Verwaltungsinformationen, die der Implementierung zur Verfügung stehen müssen.
Diese Verwaltungsinformationen werden über die Local Database Functions manipuliert:

1. **SnmpGetTranslateMode()**, **SnmpSetTranslateMode()** verändern den Translate Mode der Anwendung; der Translate Mode legt fest, in welcher Form den Entity/Context Funktionen die entity- und context-Parameter übergeben werden müssen. Die Funktionen SnmpSendMsg und SnmpRecvMsg verlangen als Parameter Handles für Quell-/ Zielentities und context; diese Handles werden von den Entity/Context Funktionen erzeugt.
   In einer SNMPv1-Umgebung stehen entities für IP-Adressen und contexts für community-strings; für IP-Adressen gibt es dabei zwei Translate Modes: zum einen kann eine IP-Adresse als benutzerfreundlicher String (z.B. „Main_Hub"), zum anderen in der „dottet version" (z.B.„192.151.209.23") übergeben werden.

2. **SnmpGetRetransmitMode(), SnmpSetRetransmitMode()** zeigen den gegenwärtigen Retransmission Mode an bzw. legen ihn fest; der Retransmission Mode gibt an, ob die WinSNMP-Implementierung die Retransmission von SNMP-Requests übernimmt oder nicht.

3. SnmpGetTimeout(), SnmpSetTimeout(), SnmpGetRetry(), SnmpSetRetry() manipulieren und zeigen die Timeout- bzw. Retry-Werte an, die die Local Database für jede Zieladresse enthält.

Abbildung auf NMS:

zu 1. Für die SNMP-Kommunikation ist die Unterstützung verschiedener Translate Modes nicht notwendig; in einer SNMPv1 Umgebung kann auf die Verwendung benutzerfreundlicher Strings für Zieladressen verzichtet werden. Man kann also festlegen, daß für IP-Adressen Strings in der „dottet version" verwendet werden (community-strings werden für jeden Translate Mode als String übergeben).
Die Funktionen SnmpGetTranslateMode und SnmpSetTranslateMode müssen dann in der NMS DLL nicht implementiert werden; für HPOV-Applikationen muß jedoch der Translate Mode dann zu Beginn entsprechend festgelegt werden.

zu 2. Auch eine Implementierung dieser Funktionen für NMS ist nicht unbedingt notwendig; im Hinblick darauf, daß unter NMS der Data Server (mit den in der Request Struktur übergebenen timeout- und retries-Werten) die Retransmission von SNMP-Requests durchführt, kann der Retransmit Mode so festgelegt werden, daß die Implementierung die Retransmission übernimmt. HPOV-Applikationen müssen dementsprechend zu Beginn SnmpSetRetransmitMode aufrufen.

zu 3. Eine Möglichkeit die Implementierung dieser Funktionen für NMS zu vermeiden besteht dann, wenn für *alle* Requests dieselben (Default-)Werte für Timeout und Retry verwendet werden können: innerhalb der Headerdateien WinSNMP.h (HPOV) und n_snmp.h (NMS) können dann diese Default-Werte definiert werden. Diese Werte werden dann in der NMS-Version von SnmpSendMsg den entsprechenden Feldern der SNMP_Request Struktur zugewiesen werden; für HPOV-Applikationen müssen dann vor jedem SnmpSendMsg-Aufruf die Werte über SnmpSetTimeout und SnmpSetRetry gesetzt werden.

Erweiterung der Minimalimplementierung der Local Database Functions

zu 2. Die Implementierung der Funktionen für die Retransmission läßt sich für NMS folgendermaßen erreichen: innerhalb der DLL muß eine globale Variable für den Mode der Retransmission definiert werden, die dann über diese Funktionen gelesen und gesetzt wird. Zusätzlich muß dann in der NMS-Version der Funktion SnmpSendMsg zuerst der alkuelle Retransmission Mode geprüft werden; falls der Mode auf SNMPAPI_OFF (die Implementierung übernimmt die Retransmission nicht) eingestellt ist, muß in der SNMP_REQUEST Struktur der Wert für retries auf 0 gesetzt werden.

zu 1.,3. Um die Local Database Functions für Translate Mode, Retry und Timeout zu implementieren, kann durch die NMS-DLL eine Tabelle implementiert werden, die für jede mögliche Zieladresse einen Eintrag mit Feldern für die Zieladresse (dotted version), den Benutzerstring, Timeout-Wert, Retry-Wert sowie den Handle auf die interne Darstellung der Zieladresse enthält. Für die Translate Mode Funktionen muß zusätzlich eine globale Variable definiert werden, in der der aktuelle Translate Mode abgespeichert wird.

Entity/Context Functions

Eine Anwendung, die auf WinSNMP aufsetzt, verwendet NULL terminierte Strings zur Darstellung von IP-Adressen und Communities. Die WinSNMP Funktionen verwenden ihrerseits die internen Darstellungen durch die WinSNMP-Implementierung; die internen Implementierungsdarstellungen von IP-Adressen und Communities orientieren sich an Kriterien wie Zugriffsgeschwindigkeit, benötigter Speicher etc. Die Lücke zwischen beiden Darstellungen wird durch die Entity/Context Funktionen geschlossen; die Entity/Context Funktionen **SnmpStrToEntity()** und **SnmpStrToContext()** wandeln die Anwendungsdarstellung in die interne Darstellung um und liefern einen Handle auf die interne Darstellung zurück. Umgekehrt geben die Funktionen **SnmpEntityToStr()** und **SnmpContextToStr()** zu einem übergebenen Handle je nach Translate Mode den entsprechenden String zurück.
Abbildung: Die entsprechenden Funktionen lassen sich für NMS mit Hilfe der oben beschriebenen Tabelle sowie einer weiteren Tabelle für die Communities - für jede Community ein Eintrag bestehend aus Feldern für die Community sowie einen Handle - umsetzen. Die interne Darstellung entspricht hier den NMS-spezifischen Strukturen für IP-Adressen (struct GEN_ADDR) und Communities (128 Byte langer String). Dementsprechend müssen die Funktionen SnmpStrToEntity und SnmpStrToContext Speicher allokieren; der community-string der Anwendung kann dabei direkt in den allokierten Pufferbereich eingetragen werden, während für Belegung der GEN_ADDR Struktur die (über die Tabelle zugängliche) dotted version benötigt wird. Die beim Allokieren zurückgegebenen Handles werden in die IP-Adressen Tabelle und die Community-Tabelle eingetragen und bei der Rücktransformation von den Funktionen SnmpEntityToStr und SnmpContextToStr verwendet.
Die Freigabe des allokierten Speichers erfolgt wie üblich mit GlobalFree innerhalb der Funktionen **SnmpFreeEntity()** und **SnmpFreeContext()**.

Utility Functions

Hierunter sind alle die Funktionen zusammengefaßt, die keinem der bisherigen Bereiche zugeordnet werden können und die für die SNMP-Kommunikation auch nicht unbedingt benötigt werden.

- **SnmpGetLastError()**:
  SnmpGetLastError gibt die letzte Fehlermeldung einer WinSNMP Funktion an den Aufrufer zuück.
  Für die Implementierung in der NMS-DLL genügt es, innerhalb der DLL eine globale Variable für die Fehlermeldungen zu definieren, die dann über SnmpGetLastError abgefragt wird. Die Variable muß von jeder Funktion neu gesetzt werden, sobald ein Fehler aufgetreten ist.

- **SnmpStrToOid()**, **SnmpOidToStr()**: SnmpStrToOid wandelt die „dottet version“ eines Objektidentifiers in die intern von der Implementierung verwendete Darstellung um; zurückgegeben wird ein Zeiger auf die interne Darstellung. SnmpOidToStr führt die entgegengesetzte Transformation durch. NMS bildet einen Objektidentifier in einen Puffer mit maximal 512 Byte Länge ab, wobei je 4 aufeinanderfolgende Bytes einen Subidentifier darstellen.
  Die Implementierung dieser beiden Funktionen unter NMS besteht deswegen im wesentlichen darin, einen Speicherbereich in einen anderen zu Kopieren.

- **SnmpOidCopy()**:
  Kopiert einen Objektidentifier in eine übergebene Objektidentifiervariable. Für die Implementierung gilt dasselbe, wie für die beiden Funktionen oben.

- **SnmpOidCompare()**:
  SnmpOidCompare vergleicht zwei Objektidentifier lexikographisch miteinander. Die Implementierung besteht Vergleich der aufeinanderfolgenden Subidentifier.

- **SnmpEncodeMsg()**:
  SnmpEncodeMsg formt aus den selben Eingabeparametern, die auch die Funktion SnmpSendMsg erhält, eine ASN.1 BER kodierte SNMP-Nachricht und schreibt diese in einen dafür bereitgestellten Puffer. Eine Implementierung verlangt neben der Bereitstellung eines Puffers nur die strikte Anwendung der ASN.1 BER.

- **SnmpDecodeMsg()**:
  SnmpDecodeMsg dekodiert eine ASN.1 BER kodiert SNMP-Nachricht und belegt dabei die selben Ausgabeparameter wie SnmpRecvMsg. Die Implementierung erfordert ebenfals nur die Anwendung der ASN.1 BER.

- **SnmpFreeDescriptor()**:
  SnmpFreeDescriptor dient dazu Speicher von Objekten freizugeben, der im Zusammenhang mit anderen WinSNMP-Funktionen belegt worden ist und über Zeiger referenziert wird; Beispiele dafür sind der Speicher für die interne Darstellung eines Contexts (SnmpContextToStr), der Speicher für die kodierte SNMP-Nachricht in SnmpEncodeMsg etc.
  Innerhalb der NMS DLL sind diese Objekte auf dem globalen Heap über GlobalAlloc angelegt worden; der Speicher eines Objekts wird mit GlobalFree und dem zugehörigen Handle als Übergabeparameter freigegeben. Der Handle wiederum kann aus einer gegebenen Adresse mit GlobalHandle gewonnen werden.

Kapitel 9

Die Schnittstelle zum Alarmmanagement

In jedem Netzwerk treten Zustandsänderungen auf, die für die Funktionsfähigkeit des Netzwerks von Bedeutung sind. Eine effiziente Netzwerkverwaltung muß in der Lage sein, auf derartige Zustandsänderungen, die auch als Ereignisse bezeichnet werden, schnell zu reagieren.
Voraussetzung dafür ist allerdings, daß die Netzwerkverwaltung schnell über Ereignisse informiert wird. Dazu stehen zwei Methoden zur Verfügung:

1. Die betroffenen Komponenten senden spontane Nachrichten (Alarme) an die Netzwerkverwaltung (NMS, HPOV, vgl. 2.2.2 und 2.3.2).

2. Die Netzwerkverwaltung fragt die einzelnen Komponenten regelmäßig daraufhin ab, ob Ereignisse aufgetreten sind (SMS, vgl. 2.4.2).

Auf den Plattformen übernimmt das Ereignismanagement die Behandlung der Ereignisse. NMS und HPOV melden Ereignisse über SNMP-Traps, die dort auch als *Alarme* bezeichnet werden; Novell und HP verwenden dementsprechend auch den Begriff *Alarmmanagement* als Synonym für Ereignismanagement. Im Rahmen dieser Arbeit wird daher ebenfalls die Bezeichnung Alarmmanagement verwendet.
In diesem Kapitel werden zunächst die Alarmmanagementsysteme der Plattformen beschrieben.
Daran anschließend wird auf die bereitgestellten APIs eingegangen; diese lassen sich fünf verschiedenen funktionellen Bereichen zuordnen. Im Einzelnen gibt es Programmierschnittstellen für den Zugriff auf alarmspezifische Informationen, statische Trapinformationen, Alarmkonfigurationen und abgespeicherte Alarme; darüberhinaus können unter HPOV auch Anwendungen Alarme generieren.
Für jede dieser Schnittstellen wird jeweils separat ein Portabilitätskonzept entwickelt.

9.1 Das Alarmmanagement der Plattformen

Bevor im nächsten Abschnitt auf die APIs der einzelnen Plattformen zum Alarmmanagementsystem eingegangen wird, sollen die Alarmmanagementsysteme kurz vorgestellt werden.

NMS

Unter NMS können Alarme von SNMP-Agenten, vom NetExplorer Manager und von der Connectivity Test Anwendung erzeugt werden. Alle diese Alarme werden dann über DDE an den Alarmmanager weitergeleitet. Je nach Konfiguration des Alarms kann der Alarmmanager verschiedene Aktionen durchführen z.B.:

- den Alarm graphisch auf der Netzkarte (Map) anzeigen,
- den Alarm zusammen mit statischen Zusatzinformationen im Alarm Monitor (die 400 letzten Alarme, Lebensdauer bis Sitzungsende) bzw. Alarm Report (alle Alarme, Lebensdauer bis Löschung) speichern und anzeigen (vgl. 2.2.2).

HPOV

Im Gegensatz zu NMS sieht HPOV die Erzeugung von Alarmen durch Anwendungen von Haus aus vor (Novell weist auf die Implementierung einer DDE-Verbindung für die Kommunikation von der Anwendung zum Alarmmanager in der Zukunft hin). Daneben werden auch hier Alarme von SNMP-Agenten sowie vom Polling Manager erzeugt. Alle Alarme werden zusammen mit Konfigurationsinformationen an die HPOV-Hauptanwendung (ovwin.exe) weitergeleitet; ovwin.exe kann auf der Basis dieser Informationen verschiedene Aktionen durchführen z.B.:

- den Alarm graphisch darstellen (Map),
- den Alarm zusammen mit statischer Zusatzinformation im AlarmLog abspeichern (maximal 900 Einträge); nach der Bestätigung durch den Betreiber werden diese Alarme dann im HistoryLog abgespeichert.

SMS

Unter SMS werden Ereignisse (in SMS-Terminologie Events) nicht an die Plattform geschickt, sondern die Netzkomponenten werden nach Ereignissen abgefragt. Liegt ein Ereignis vor, dann wird dieses im Rahmen eines sogenannten Event MIF file (MIF = Management Information File) an den entsprechenden Site Server weitergegeben und von diesem wiederum in der Site Database abgespeichert. Die gespeicherten Ereignisse können - gegebenenfalls gefiltert - im sogenannten Events-Fenster von SMS angezeigt werden.

9.2 Die APIs zum Alarmmanagement der Plattformen

Die APIs der Plattformen zum Alarmmanagement können fünf verschiedenen funktionellen Bereichen zugeordnet werden:

APIs für alarmspezifische Informationen

Ein Alarm enthält i.a. spezifische Informationen wie Zeitstempel oder (bei Traps) Agentenadresse, die über diese APIs an die Anwendungen weitergegeben werden. Unter NMS baut eine Anwendung dazu eine DDE-Verbindung zum Alarmmanager auf, über die ihr dann die Alarme - hier kann je nach verwendetem DDE-Item (i.e. das „Gesprächsthema"der beiden DDE-Kommunikationspartner, vgl. z.B. auch [Hon92]) zwischen SNMP-Alarmen, NetExplorer Manager Alarmen oder allen Alarmen gewählt werden - als sogenannte EM_CLIENT_CANON-Strukturen übergeben werden. Demgegenüber kann sich eine Anwendung unter HPOV bei der WINSNMP.DLL nur als Empfänger für SNMP-Traps registrieren lassen; SMS bietet keine entsprechende Schnittstelle an.

APIs für den Zugriff auf statische Trapinformationen

Außer den Objekttypen werden in MIBs auch Traptypen definiert; Traptypen werden üblicherweise unter Verwendung des TRAP-TYPE Makros (vgl. [RFC91b]) definiert und enthalten dementsprechend:

- den Trapnamen,
- einen Objektidentifier, der die MIB im ISO-Objektidentifier-Baum identifiziert (Datentyp ENTERPRISE),
- eine geordnete Sequenz der MIB-Objekte, deren Werte mit dem Trap zusammen gesendet werden und das Ereignis näher beschreiben (Varbind-Liste),
- eine kurze textuelle Beschreibung des Traps sowie
- eine Nummer, die zusammen mit dem Objektidentifier für die MIB den Trap eindeutig identifiziert.

Beispiel:

```
dupIpxNetAddr                    TRAP-TYPE
      ENTERPRISE                 netware-GA-alert-mib
      VARIABLES                  {osName, osLoc, tiTrapTime,
                                 tiEventValue, tiEventSeverity,
                                 tiServer}
      DESCRIPTION                ,,Two servers use the same
                                 IPX internet address.''
      ::=8
```

Darüberhinaus können die Trap-Definitionen z.B. durch eingefügte Kommentare innerhalb der MIB-Definitionen (NMS, vgl. [Nov93d]) oder zusätzliche Dateien (HPOV: Trap Definition File, vgl. [HP95a]) erweitert werden. Alle in diesen Trap-Definitionen enthaltenen Informationen werden als *statische* Trapinformationen bezeichnet.
Unter NMS werden die statischen Trapinformationen über den MIB-Compiler in die Plattform integriert; der MIB-Compiler generiert dazu eine entsprechende Datei (TRAP.BTV) in der Btrieve-DB. (vgl. [Nov93d])
Unter HPOV müssen Entwickler zur Integration dieser Informationen ein separates, sogenanntes Trap Definition File schreiben; das Format dieser Datei folgt dabei dem Windows .INI-Format. Die Datei selbst muß dann in einem bestimmten Verzeichnis (/OV/TRAPMGR) abgelegt werden.(vgl. [HP95a])
SMS bietet keine entsprechende Schnittstelle an (kein SNMP!).

APIs für den Zugriff auf Informationen über die Konfiguration der Alarme

Die Behandlung der Alarme durch das Alarmmanagement kann zum großen Teil vom Benutzer konfiguriert werden. Die Konfigurationsinformationen sind unter NMS ebenfalls in der Btrieve-Datenbank TRAP.BTV eingetragen, unter HPOV im Trap Definition File.

APIs für den Zugriff auf abgespeicherte Alarme

Die Alarmmanagementsysteme aller drei Plattformen speichern empfangene Alarme ab:i
NMS in der Btrieve-Datenbank EVENT.BTV, HPOV in einer Paradox-Datenbank und SMS in einer SQL Server Datenbank. Die Informationen können über die entsprechenden DB-Requests (keine eigens implementierten APIs) abgerufen werden; dabei ist jedoch zu beachten, daß unter SMS alle Ereignisse abgespeichert werden, unter NMS und HPOV nur die entsprechend konfigurierten Alarme.

APIs für die Generierung von Alarmen durch die Anwendung

Unter HPOV ist zusätzlich ein API implementiert, über das Anwendungen eigene Alarme definieren, konfigurieren und generieren können. Die Anwendungen benutzen dabei das gleiche API, das auch die alarmgenerierenden Basisanwendungen dieser Plattform - Trap Manager und Polling Manager - verwenden.
Voraussetzung für die Alarmgenerierung ist, daß die betreffende Anwendung ihre *Alarm Sets* und ihre Alarme während der Startup-Phase von HPOV beim Alarmsystem registrieren läßt (vgl. 3.2.2). Alarm Sets sind Alarmgruppen, zu denen Alarme unter HPOV zusammengefaßt werden; i.a. werden die Alarme, die eine Anwendung generieren kann, zu einem Alarm Set zusammengefaßt. Die Registrierung erfolgt über die API-Aufrufe *OVAlarmRegisterSet()* und *OVAlarmRegisterType()*. Gleichzeitig mit der Aufforderung zur Registrierung werden ovwin.exe eine textuelle Kurzbeschreibung des Alarmtyps sowie die Adresse eines evtl. vorhandenen Help-Files zu dem Alarmtyp übergeben. Die Kurzbeschreibung wird dann jeweils im AlarmLog-Fenster angezeigt, über den Help-Button dieses Fensters wird der Inhalt des Help-Files abgerufen.
Die Alarmgenerierung selbst erfolgt über Aufruf von OVAlarmEvent; erst mit diesem Aufruf wird neben den aktuellen Alarmdaten - alarmauslösendes Objekt, Zeitstempel und der alarmspezifischen Beschreibung (über einen Formatstring) - auch die Konfiguration des Alarms über sogenannte *Action Flags* übergeben; die Alarmkonfigurationen werden also lokal von der Anwendung gehalten.
Auch die Bestätigung dieser Alarme kann außer durch den Benutzer (über den Oberflächenbaustein) auch programmatisch durch die Anwendung selbst erledigt werden: *OVAlarmEvent()* liefert dem Aufrufer einen Schlüssel für den Zugriff auf den erzeugten Alarm zurück. Dieser Schlüssel wird von *OVAlarmClear* als Parameter verwendet. Der Aufruf von OVAlarmClear() bewirkt außerdem, daß der Anwendung eine Nachricht von ovwin.exe zugeschickt wird. Diese Nachricht kann wiederum von der Anwendung dazu verwendet werden, erneut einen Alarm zu generieren, der jedoch nur die Statusanzeige der Map dem höchsten jetzt noch unbestätigten Alarm anpaßt.
Zu den anwendungsspezifischen Alarmen ist jedoch zu bemerken, daß die Konfigurationsinformationen zwar lokal von der Anwendung, aber nicht wie beim Trap Manager und beim Polling Manager in separaten Dateien gehalten werden. Der Benutzer hat daher hier auch nicht wie bei den Alarmen dieser Anwendungen die Möglichkeit, die Alarme über die Benutzeroberfläche zu konfigurieren, d.h. die anwendungsspezifischen Alarme sind nicht voll in das Alarmsystem integriert.

9.3 Das Portabilitätskonzept

Im folgenden wird für jede der oben aufgeführten Teilschnittstellen ein Konzept entwickelt. Dabei wird zunächst jeweils eine portierbare Schnittstelle festgelegt, die dann auf die vorhandenen Plattformschnittstellen abgebildet wird.

9.3.1 Alarmspezifische Informationen

Spezifizierung der Schnittstelle

Um einen Standard für diese Teilschnittstelle festzulegen, muß festgelegt werden:

1. welche Alarme über diese Schnittstelle an die Anwendungen weitergegeben werden sollen und

2. welche Informationen dabei übergeben werden sollen.

Die Festlegung auf SNMPv1-Traps und die damit verbundenen Informationen bietet sich aus mehreren Gründen an:

- die Alarmmanagementsysteme von NMS und HPOV bieten bereits eine derartige Schnittstelle an.

- Sowohl die Traps selbst - durch die entsprechenden MIB-Definitionen - als auch die in den SNMP-Traps enthaltenen Informationen sind standardisiert. Jeder Trap enthält Angaben zum Hersteller des Managed Objects sowie der Agentenadresse, spezifische Trapinformationen, einen Zeitstempel und eine Liste Trap-relevanter Variablen.

- Auf SMS kann eine solche Schnittstelle durch die Installation eines SNMP-Protokollstacks realisiert werden.

Abbildung auf die Plattformschnittstellen

Die alarmspezifischen Informationen können unter HPOV direkt aus dem empfangenen Trap mit Hilfe der WinSNMP-Funktionen gewonnen werden; unter NMS werden sie innerhalb der mit DDE übergebenen Datenstrukturen EM_CLIENT_CANON (vom Alarmmanager) oder SNMP_REQUEST (vom SNMP Data Server) an die Anwendungen weitergeleitet (vgl. [Nov93a]).

9.3.2 Statische Trapinformationen

Spezifikation einer Schnittstelle

Die Spezifikation dieser Schnittstelle kann wiederum in einem kombinierten Bottom-up- und Top-down-Verfahren durchgeführt werden (7.3.3).
Dabei müssen die statischen Trapinformationen, die einer Anwendung zur Verfügung gestellt werden sollen, und die Funktionsaufrufe zu deren Abfrage - die Informationen sollen nur gelesen werden können - spezifiziert werden.

Als Basis für diese Spezifikation bieten sich zunächst die im TRAP-TYPE Makro für die Definition von Traps in MIBs festgelegten und somit standardisierten Informationen an (Top-down).
Eine Erweiterung hängt davon ab, welche zusätzlichen Informationen auf den beiden Plattformen definiert werden (Bottom-up). NMS definiert Zusatzinformationen als Kommentare innerhalb der Standard-Trap-Definition nach dem TRAP-TYPE Makro:
Beispiel:

```
dupIpxNetAddr                    TRAP-TYPE
      ENTERPRISE                 netware-GA-alert-mib
      VARIABLES                  {osName, osLoc,
                                 tiTrapTime, tiEventValue,
                                 tiEventSeverity, tiServer}
      DESCRIPTION                ,,Two servers use the same
                                 IPX internet address. ''
      --NMS trap annotation

      --#TYPE                    ,,Duplicate IPX address ''
      --#SUMMARY                 ,,% s at % s and % s are using
                                 the same IPX address ''
      --#ARGUMENTS               { 0, 1, 5 }
      --#SEVERITY                CRITICAL
      --#TIMEINDEX               2
      --#HELP                    ,,MYHELP:HLP ''
      --#HELPTAG                 60004
      --#STATE                   DEGRADED
      ::=8
```

Dabei werden in ARGUMENTS die den Platzhaltern entsprechenden Objekte aus dem VARIABLES-Absatz und deren Reihenfolge im SUMMARY-String beschrieben, SEVERITY bezeichnet die dem Trap zugeordnete Dringlichkeitsstufe und STATE den aktuellen Zustand Managed Node, der den Trap gesendet hat. HELP und HELPTAG dienen dazu, dem Betreiber zusätzliche Hilfeinformation über einen Trap zu geben.
Unter HPOV können die folgenden Zusatzangaben zu Traps im Trap Definition File festgelegt werden:

- Der (Folge-)Status des betroffenen Knotens.

- Eine Trap-Beschreibung, in die die Variable Bindings des Trap über entsprechende Format-Parameter aufgenommen werden.

- Eine Zusatzbeschreibung, ebenfalls mit Format-Parametern, die dem Benutzer über das *More Info*-Dialogfenster zugänglich gemacht wird.

Für den Aufbau des Trap Definition File wird das .INI File Format von Windows benutzt: die Datei ist in Abschnitte unterteilt, wobei in jedem Abschnitt die Traps für einen Managed Node beschrieben werden. Innerhalb jedes Abschnitts existiert für jeden Trap ein Eintrag, der außer den statischen Trapinformationen auch noch Informationen über die Konfiguration der Traps enthält.
Damit bieten sich für die Spezifizierung dieser Schnittstelle neben den Basisinformationen aus dem TRAP-TYPE Makro als weitere Informationen an:

- die Beschreibung des Status des betroffenen Objekts; dazu müssen die Statusbeschreibungen der beiden Plattformen angepaßt werden. NMS verwendet zum einen Dringlichkeitsstufen (Severities) für die Bewertung von Alarmen, zum anderen Status zur Beschreibung des Objektzustands; HPOV verwendet dagegen lediglich 11 verschiedene Status. Da die Dringlichkeit eines Alarms abhängig ist vom Folgezustand des betroffenen Objekts und HPOV bei den Status mehr differenziert (11 Status gegen 4 unter NMS) kann eine Lösung dadurch erreicht werden, daß als Schnittstellenstandard die HPOV-Status verwendet werden und die NMS-Angaben entsprechend angepaßt werden.

- die Zusatzbeschreibung für Traps: Voraussetzung für eine Standardisierung dieser Schnittstelle ist natürlich, daß für gleiche Traptypen textuell - mit Ausnahme der Formatstrings - gleiche Beschreibungen mit gleicher Anzahl und gleicher Reihenfolge der Parameter vorliegen. Da die Formatstrings unter NMS eine Teilmenge der Formatstrings von HPOV sind, können diese als Standard verwendet werden.

- die Beschreibung der Traps in Helpfiles bzw. für den „More Info "-Dialog: hier gilt das Gleiche wie für die Zusatzbeschreibung.

Keine der untersuchten Plattformen bietet für die Abfrage der statischen Informationen eigens implementierte APIs an, so daß im Bezug auf die Spezifikation einer oder mehrerer Abfragefunktionen weitgehend frei gewählt werden kann. Eine Möglichkeit wäre beispielsweise eine einfache Abfrage der einzelnen Einträge über die Trap-ID.

Abbildung auf die Plattformschnittstellen

Abfragefunktionen können unter NMS in die entsprechenden Btrieve-Requests für die TRAP.BTV Datei umgesetzt werden; unter HPOV muß dazu eine Anwendung implementiert werden, die die gewünschten Informationen aus dem Trap Definition File ausliest.

9.3.3 Konfigurationsinformationen

Spezifizierung einer Schnittstelle

Hier kommen zunächst nur die Möglichkeiten der Alarmkonfigurierung, die in beiden Alarmsystemen bestehen, in Frage:

- Erzeugung eines akustischen Signals,
- Abspeichern des Alarms in einer Datenbank.

Andere Alarmdispositionen sind entweder nur auf einer der Plattformen konfigurierbar oder unterschiedlich implementiert:
So kann unter HPOV konfiguriert werden, ob ein Alarm auch optisch angezeigt wird (Map-Update), wogegen unter NMS Alarme mit einer bestimmten Dringlichkeitsstufe jeweils automatisch angezeigt werden. Beide Alarmsysteme erlauben außerdem, den Start externer Programme zu konfigurieren:
Während jedoch unter NMS für jeden Alarm individuell ein Programm gestartet werden kann, kann ein Alarm unter HPOV nur ein seiner Dringlichkeitsstufe zugeordnetes Programm starten. Die Spezifizierung eines Standards ist für diese Fälle ohne entsprechende funktionelle Erweiterung der Plattformen nicht möglich.

Abbildung auf die Plattformen

Die Konfigurationsinformationen für akustische Signale und Abspeicherung liegen in der Btrieve-Datenbank bzw. dem Trap Definition File; die Abbildung erfolgt daher wie für die statischen Trapinformationen.

9.3.4 Abgespeicherte Alarme

Spezifikation einer Schnittstelle

Die Alarmsysteme von NMS und HPOV speichern bei entsprechender Konfiguration durch den Benutzer jeweils die gleichen Alarme ab: unter NMS in der EVENT.BTV Datei derBtrieve-DB und in einer Paradox-DB unter HPOV.
Die Informationen, die jeweils in einem Alarmeintrag dieser DBen enthalten sind, ergeben sich aus den DB-Schemata (vgl. [Nov93c] und [HP95a]). Eine Untersuchung dieser Schemate ergibt, daß die HPOV-Alarmattribute eine Teilmenge der Attribute der NMS-Alarme darstellen. So enthält ein Alarm-Record unter NMS zusätzlich beispielsweise einen Zeitstempel für die Bestätigung durch den Verwalter, die Adresse des Objekts, das den Alarm erzeugt hat etc.
Die folgende Tabelle zeigt, welche Informationen in den Alarm-Records beider Alarmdatenbanken enthalten sind:

Art der Information	Feld unter HPOV	Feld unter NMS
Herkunft des Alarms:		

Objekttyp (z.B PC)	affected_object_type	device type
Objektname (z.B. PC ''Fred'')	device name	-- (s.u.)
Zeitstempel (Zeitpunkt, zu dem der Agent den Alarm generiert hat)		
	date	agent_time
	hour	
	minutes	
	seconds	
Dringlichkeit des Alarms	severity	severity
Folgestatus des betroffenen Objekts	status	state
textuelle Alarmbeschreibung	message	summary
textuelle Zusatzbeschreibung	extended description	detail

Ein Alarmeintrag unter NMS enthält kein Feld mit dem Namen des betroffenen Objekts; der Name kann jedoch über die im Alarm-Record enthaltene Objekt-ID von einer anderen Btrieve-Datenbank abgefragt werden.

Abbildung auf die Plattformen

Die Abbildung erfolgt auf die entsprechenden Btrieve- bzw. Paradox-Requests.

9.3.5 Alarmgenerierung

Spezifikation einer Schnittstelle

Die Spezifikation dieser Schnittstelle gestaltet sich deshalb schwieriger als die für die vorangegangenen Schnittstellen, da zum einen kein Standard existiert und zum anderen mit der HPOV-Implementierung auch nur ein Anschauungsbeispiel vorliegt.
Außerdem implementiert die HPOV-Schnittstelle Funktionen, die mit der Alarmgenerierung selbst nicht unmittelbar zusammenhängen und deshalb auch in der Schnittstellenspezifikation nicht enthalten sein müssen:

1. Die Registrierung der Alarme beim Alarmmanagement ist Teil der Systemkonfiguration und nicht des Netzmanagements. Dem Alarmmanagement werden dabei nur die Informationen übermittelt, die es benötigt, um Alarme erkennen zu können. Diese Aufgabe kann jedoch auch durch den Verwalter selbst ausgeführt werden und muß nicht wie unter HPOV programmatisch in einer Initialisierungsprozedur erledigt werden.

2. Die Möglichkeit zur programmatischen Alarmbestätigung ist überflüssig, da die Benutzeroberfläche eine entsprechende Funktion anbietet.

Die Spezifikation dieser Schnittstelle gestaltet sich deshalb schwieriger als die für die vorangegangenen Schnittstellen, da zum einen kein Standard existiert und

zum anderen mit der HPOV-Implementierung auch nur ein Anschauungsbeispiel vorliegt.
Einen Ansatz zur Spezifikation der Schnittstellenfunktionen liefert die funktionelle Betrachtung der Schnittstelle:
die Managementanwendung holt sich „ihre“Informationen über den Kommunikationsbaustein der Plattform, bewertet diese und sendet dann gegebenenfalls einen Alarm an das Alarmmanagementsystem. Voraussetzung dafür ist zunächst, daß eine Verbindung zum Alarmmanagement besteht, über die die Alarme gesendet werden können. Danach muß - entsprechend dem verwendeten Kommunikationsprotokoll - eine Protokolldateneinheit (PDU) aufgebaut werden, die alle vom Alarmmanagement im Zusammenhang mit einem Alarm benötigten Informationen enthält. Die fertige PDU wird dann an das Alarmmanagement der Plattform geschickt.
Dementsprechend werden mindestens vier Funktionen benötigt:

- **OpenAlarmSession()** für den Aufbau und die Initialisierung einer Verbindung. Die Funktion benötigt keine Parameter, da die Kommunikationspartner beide feststehen; ein Rückgabewert zeigt an, ob die Verbindung aufgebaut werden konnte oder nicht und gibt im Erfolgsfall einen Session-Handle zurück.

- **BuildAlarmPdu()** für den Aufbau der PDU. Parameter müssen für alle Daten bereitgestellt werden, die das Alarmmanagement für die Behandlung der Alarme benötigt. Für den Fall fehlender bzw. fehlerhafter Parameter können entsprechende Werte zurückgegeben werden.

- **SendAlarm()** um den Alarm an das Alarmmanagement weiterzugeben. Als Parameter wird hier die PDU übergeben.

- **CloseAlarmSession()** für den Abbau der Verbindung und die Freigabe von Ressourcen, die im Zusammenhang mit der Verbindung belegt wurden. Als Parameter wird dabei der beim Aufbau zurückgegebene Session-Handle verwendet.

In einem nächsten Schritt müssen die Parameter der PDU spezifiziert werden, die beim Aufbau der PDU mit übergeben werden müssen.
Einen Anhaltspunkt dafür, Parameter werden müssen, liefert die Betrachtung der Alarmmanagementsysteme von NMS und HPOV sowie der alarmgenerierenden Basisanwendungen SNMP Data Server bzw. Trap Manager und Polling Manager. Unter NMS werden sowohl die statischen Alarminformationen wie auch die Konfigurationsinformationen vom Alarmmanager gehalten; Alarme, die dem Alarmmanager übergeben werden, enthalten hier nur noch die alarmspezifischen Informationen. Demgegenüber werden unter HPOV die Konfigurationinformationen von den alarmgenerierenden Basisanwendungen gehalten; ein Alarm enthält

daher unter HPOV neben den alarmspezifischen Informationen auch die Konfigurationsinformationen.
Auf der Ebene der gemeinsamen Schnittstelle genügt es, nur die alarmspezifischen Daten als PDU-Parameter zu definieren; die unter HPOV benötigten Konfigurationsdaten können dann erst durch die Abbildung der Funktionen auf die Plattformen in die tatsächlichen Funktionsaufrufe integriert werden. Für die PDU ergeben sich damit die folgenden Parameter:

- der Alarmtyp als Schlüssel für den Zugriff auf die dazugehörigen statischen und Konfigurationsinformationen.
- der Typ und die Instanz des alarmauslösenden Objekts.
- der Zeitstempel, der angibt, wann der Alarm erzeugt wurde.
- evtl. eine Zusatzbeschreibung des Alarms (zusätzlich zu den alarmtypspezifischen Beschreibungen innerhalb der statischen Informationen), die alarmspezifische Daten einschließt. Beispiele sind die Formatstrings des Trap Definition File und der TRAP.BTV Datenbank.

Abbildung auf die Plattformen

Das Problem bei der Abbildung der spezifizierten Schnittstelle auf die Plattformen liegt - da bis jetzt nur HPOV APIs zur Alarmgenerierung anbietet - darin, daß auf den anderen beiden Plattformen die entsprechende Funktionalität nachimplementiert werden muß.
Eine Lösung kann durch den Einsatz von *Loopback-Gerätetreibern* erreicht werden. Ein Loopback-Gerätetreiber legt auf Software-Ebene ein Interface mit der IP-Netzadresse 127.0.0.1 an. Diese Adresse verweist auf den Rechner selbst, d.h. Pakete mit dieser Zieladresse werden von den Loopback-Treibern wieder an den lokalen Rechner („localhost “) zurückgeleitet. Loopback-Treiber sind auf allen TCP/IP-Systemen installiert. (vgl. [Hun95])
Der Loopback-Treiber kann von den Managementanwendungen benutzt werden, um eigene Alarme zu generieren. Voraussetzung ist allerdings, daß auf der jeweiligen Managementstation eine SNMP-Kommunikationsschnittstelle und ein entsprechendes Alarmmanagementsystem vorhanden sind:
Zunächst werden die Anwendungs-Traps, wie alle anderen Traps auch, in das jeweilige Alarmmanagementsystem integriert, also beispielsweise unter NMS mit Hilfe des MIB-Compilers (vgl. 8.2); Voraussetzung ist, daß entsprechende MIBs definiert werden.
Will nun eine Anwendung einen Alarm erzeugen, baut sie eine entsprechende Trap-PDU auf, die dann über die SNMP-Schnittstelle an die Netzadresse 127.0.0.1 gesendet wird. Die Trap-PDU wird dann vom Loopback-Treiber wie

normaler SNMP-Trap an die Managementstation (zurück-) geleitet, d.h. die Anwendung fungiert als SNMP-Agent.
Eine Umsetzung dieses Ansatzes hätte zusätzliche Vorteile:

- die Alarme wären an das Internet-Informationsmodell angepaßt und dementsprechend auch standardisiert.
- eine Ausweitung zur Manager-to-Manager Kommunikation unter SNMPv2 wäre leicht möglich: statt der Netzwerkadresse 127 wird dann die Adresse der Zielstation angegeben und der PDU-Typ entsprechend angepaßt.

Eine Umsetzung empfiehlt sich daher auch für HPOV.
Eine detaillierte Beschreibung der Abbildung kann dabei nur für NMS und HPOV gemacht werden:

- **OpenAlarmSession()**: unter HPOV ruft OpenAlarmSession die entsprechenden WinSNMP APIs SnmpStartup und SnmpOpen auf, während unter NMS eine DDE-Verbindung zum SNMP Data Server aufgebaut und initialisiert werden muß (DdeInitiate und DdeConnect).
- **BuildAlarmPdu()**: innerhalb der BuildAlarmPdu wird unter HPOV mit Hilfe der PDU- und Variable Binding Functions die gewünschte Trap-PDU aufgebaut, unter NMS eine SNMP_Request Struktur. Parameter?
- **SendAlarm()**: als Parameter werden der Handle auf die PDU bzw. die SNMP_Request Struktur übergeben. Unter HPOV wird mit SendAlarm die WinSNMP-Funktion SnmpSendMsg aufgerufen, unter NMS die DDE-Funktion DdeClientTransaction.
- **CloseAlarmSession()**: unter HPOV wird durch CloseAlarmSession SnmpClose, unter NMS DdeDisconnect aufgerufen.

Kapitel 10

Die Schnittstelle zum Topologiemanagement

Nach [Heg93] sind mit Topologieinformationen alle Informationen über die Konfiguration eines Netzes und der darin enthaltenen Ressourcen gemeint. Basisanwendungen im Bereich Topologiemanagement werden entsprechend ihrer Funktionalität in zwei Kategorien eingeteilt (vgl. [Heg93]): in Discovery- und in Autotopology-Anwendungen:
Discovery-Anwendungen sammeln unter Verwendung von Managementprotokollen soviele Topologieinformationen wie möglich und speichern diese in der Datenbank der Plattform ab. Zu ihren Aufgaben gehört auch, diese Informationen ständig zu aktualisieren, also Änderungen der Netztopologie zu erkennen und zu dokumentieren.
Autotopology-Anwendungen bauen darüberhinaus auf der Basis der gesammelten Informationen die Netztopologie des überwachten Netzes auf und stellen sie über den Oberflächenbaustein graphisch dar.
In diesem Kapitel werden zunächst die Topologiemanagementsysteme der Plattformen und die Anwendungsprogrammierern von ihnen zur Verfügung gestellten APIs vorgestellt. Dabei stellt sich heraus, daß nur NMS und HPOV in diesem Bereich Basisanwendungen und dementsprechend auch APIs implementieren.
Daran anschließend wird in einem kombinierten Verfahren aus Top-down und Bottom-up (vgl. 7.3.3) eine Schnittstelle spezifiziert und auf die Plattformen von Novell und HP abgebildet.

10.1 Das Topologiemanagement der Plattformen

10.1.1 Das Topologiemanagement unter NMS

Komponenten

Das Topologiemanagement unter NMS ist als verteilte Anwendung auf der NMS Konsole, dem NMS Server - einem speziellen NetWare Server, der die Hauptkomponenten des Topologiemanagements enthält - und anderen NetWare Servern implementiert ist. Im Einzelnen besteht das Topologiemanagementsystem aus folgenden Komponenten:

- Der NetExplorer Manager auf der NMS Konsole hält die Topologieinformationen in der NMS-Datenbank auf der Basis der vom NetExplorer gelieferten Daten auf dem aktuellen Stand.

- Der NetExplorer auf dem NMS Server kommuniziert auf der einen Seite mit den Komponenten, die die Topologiedaten abfragen (NXPIPX.NLM, NXPIP.NLM, NXPLANZ.NLM), und speichert die von diesen Komponenten gelieferten Daten auf dem NMS Server zwischen. Auf der anderen Seite übermittelt der NetExplorer alle Topologieinformationen, die neu sind oder sich geändert haben an den NetExplorer Manager. Wie alle folgenden Komponenten auch ist der NetExplorer als sogenanntes NetWare Loadable Module (NLM) implementiert. NLMs sind Module, die vom NetWare Betriebssystem-Kernel (ab Version 3.11) erst zur Laufzeit geladen werden. Die Module werden dadurch Teil des Betriebssystems und erweitern dessen Funktionalität.

- Das NLM NXPIPX.NLM auf dem NMS Server fragt Informationen über IPX-Komponenten in IPX-Netzen ab. NXPIPX.NLM verwendet dazu verschiedene proprietäre Protokolle wie RIP, SAP, NCP und andere Diagnostics oder SNMP. Die dabei gesammelten Topologieinformationen über neue Komponenten, Veränderungen an bestehenden oder über wieder entfernte Komponenten werden an den NetExplorer weitergeleitet.

- Das NLM NXPIP.NLM auf dem NMS Server holt sich die Topologieinformationen über IP-Router auf IP-Netzen via SNMP.

- Das NLM NXPLANZ.NLM auf dem NMS Server kommuniziert über SNMP mit evtl. auf NetWare Servern im Netz installierten NetWare LANalyzer Agenten. NetWare LANalyzer Agenten überwachen den Datenverkehr auf an den NetWare Server angeschlossenen Ethernet- bzw. Token Ring-Segmenten und liefern im wesentlichen Statistiken zur Auslastung der Segmente und ihrer Komponenten.

Ablauf

Die Topologiedaten werden in Zyklen abgefragt und an den NetExplorer weitergegeben, wobei die Abfolge der Zyklen vom Verwalter konfiguriert werden kann: entweder werden permanent Informationen gesammelt und weitergeleitet und ein Zyklus folgt unmittelbar auf den vorhergehenden oder die Zyklen werden an bestimmten Zeitpunkten gestartet. Der Startzyklus ist erst dann beendet, wenn keine neuen Komponenten mehr gefunden werden; in den folgenden Zyklen werden die vorhandenen Daten nur noch vervollständigt bzw. geändert.
Jeder Zyklus besteht seinerseits aus drei unabhängigen Prozessen, die mit den drei NMLs des NMS Servers korrespondieren:

- Der von NIXIP.NLM initiierte Prozeß holt sich sukzessive alle Routing Informationen aller IP-Router beginnend mit der Routing Tabelle des NMS Servers: Die Routing-Tabellen aller darin aufgeführten IP-Router werden über SNMP abgefragt, danach die in diesen Tabellen eingetragenen IP-Router und so fort.

- Analog „hangelt “sich der von NIXIPX.NLM ausgeführte Prozeß von Bindery zu Bindery und identifiziert so nach und nach alle IPX-Komponenten des Netzes.

- Die NetWare LANalyzer Agenten überwachen den Datenverkehr in den Segmenten, an die ihr NetWare Server angeschlossen ist, und identifizieren anhand der in den Datenpaketen verwendeten MAC-Adressen die einzelnen Segmentkomponenten. Der von NIXLANZ.NLM gestartete Prozeß holt sich diese Informationen über SNMP.

Die gesammelten Informationen werden an den NetExplorer übergeben, der sie zunächst in einem sequentiellen File (NETXPLOR.DAT) zwischenspeichert. Der NetExplorer leitet diese Daten auf Anfrage an den NetExplorer Manager weiter, der die Daten entsprechend dem zugrundeliegenden Informationsmodell interpretiert und in der NMS Datenbank abspeichert.

10.1.2 Das Topologiemanagement unter HPOV

Komponenten

Auch unter HPOV lassen sich innerhalb des Topologiemanagementsystems mehrere Komponenten unterscheiden:

- Der *IP-Discovery Prozeß* identifiziert wie NXPIP.NLM unter NMS sukzessive alle IP-Komponenten über die Routing- und ARP-Tabellen der IP-Router.

- Der *Extended Discovery Prozeß* erkennt alle IPX-Komponenten via NetWare Diagnostics und SNMP (vgl. NXPIPX.NLM) und liefert evtl. zusätzliche Informationen über schon bekannte IP-Komponenten.

- Die *Layout-Funktion* erzeugt automatisch eine graphische Darstellung der erkannten Komponenten.

Ablauf

Beide Discovery-Prozesse laufen periodisch (konfigurierbar) als Hintergrundprozesse und erweitern bzw. aktualisieren die Topologiedatenbank. Im Gegensatz zu NMS kann dabei konfiguriert werden welche Netze durchsucht werden sollen, der Adressbereich für die zu identifizierenden Komponenten oder welche Komponententypen erfaßt werden sollen.

10.1.3 Das Topologiemanagement unter SMS

Die Site-Hierarchie (vgl. 2.4) des Netzes, die Unterteilung der einzelnen Sites in Domänen (i.e. die Zusammenfassung von Servern und Clients aus verwaltungstechnischen Gründen, z.B. um einen einheitlichen Account innerhalb einer Domäne einrichten zu können) und die Server und Clients innerhalb einer Domäne werden unter SMS in einem sogenannten Site-Fenster dargestellt.
Dennoch ist unter SMS kein eigentlicher Topologiemanager implementiert; alle für die Darstellung notwendigen Informationen werden SMS vom Verwalter während der Installation und Konfiguration der Sites und Domänen sowie der Installation der Clients übergeben. So muß bei der Installation einer Site die Parent-Site angeben werden und Clients werden beispielsweise über Logon-Scripts in SMS integriert.

10.2 Die APIs zum Topologiemanagement der Plattformen

APIs zum Topologiemanagement werden nur von NMS und HPOV angeboten; die APIs, die dabei zur Verfügung gestellt werden, sind im wesentlichen Programmierschnittstellen zu den Topologie-DBen.
Darüberhinaus stellt HPOV eine Funktion *OVADLRegisterForAutodiscovery* für Discovery-Anwendungen - das schließt Autotopology-Anwendungen natürlich mit ein - bereit: erst wenn diese Funktion von einer Discovery-Anwendung in ihrer Initialisierungsprozedur (vgl. 3.2.2) aufgerufen wurde, kann sie *schreibend* auf die Topologie-DB zugreifen. Die Anwendung unterliegt damit auch dem vom Benutzer festgelegten, globalen Discovery-Scheduling für Discovery-Anwendungen.

10.3 Das Portabilitätskonzept

10.3.1 Spezifikation einer Schnittstelle

In 7.3.3 wurde angedeutet, daß Informations(teil)modelle, die ein Netz aus topologischer Sicht beschreiben weitgehend generisch sind.
Diese Annahme stützt sich darauf, daß bei der topologischen Beschreibung eines Netzes zum einen auf allgemeine, strukturelle Netzelemente zurückgegriffen wird und zum anderen darauf, daß über die Sicht des Topologiemanagements dieser Netzelemente weitgehend Konsens besteht.
So verwenden topologische Netzbeschreibungen beispielsweise die strukturellen Netzelemente „Netzwerk“und „Segment“; ein Netzwerk wird allgemein aus *topologischer* Sicht über seinen Typ (z.B. IP,IPX), seine Netzwerkadresse und eine evtl. vorhandene Subnet-Maske charakterisiert, ein Segment durch die Zugehörigkeit zu einem Netzwerk.
Aus diesen Gründen bietet sich eine kombinierte Methode aus Top-Down und Bottom-up für die Spezifizierung einer einheitlichen Schnittstelle zu den Topologie-DBen an; es muß also zunächst ein grobes Informationsmodell für die Netztopologie festgelegt werden (Top-down), das dann im nächsten Schritt an die Plattformschnittstellen angepaßt wird (Bottom-up).(vgl. 7.3.3)

1.Schritt: Top-down

Im ersten Schritt wird zunächst ein grobes Informationsmodell des Netzes aus topologischer Sicht entworfen:
Jedes Netz besteht aus kleineren Teilnetzen (Netzwerken, Subnets), die untereinander über Router verbunden sind; ein Netzwerk wird dabei über seine Netzwerkadresse, seinen Netzwerktyp (IP,IPX) und eine evtl. vorhandene Subnet-Maske eindeutig identifiziert.
Die Netzwerke ihrerseits unterteilen sich in ein oder mehrere Segmente, die über Brücken miteinander verbunden sind; ein Segment wird durch seine Zugehörigkeit zu einem Netzwerk charakterisiert.
Die einzelnen Knoten, die die Netz- und Systemkomponenten des Netzes repräsentieren, werden wiederum über Interfaces mit den Segmenten verbunden. Ein Interface wird dabei durch seine MAC-Adresse, seine Netzwerkadresse sowie dem Knoten und dem Segment, die durch das Interface verbunden werden, beschrieben.
Die Knoten selbst werden durch ihren Knotentyp - i.e. ihre Funktion im Netz wie z.B. Router, Server etc. - beschrieben.
Auf der Basis dieser topologischen Charakterisierung eines Netzes läßt sich *beispielsweise* folgendes, informelles Informationsmodell ableiten:

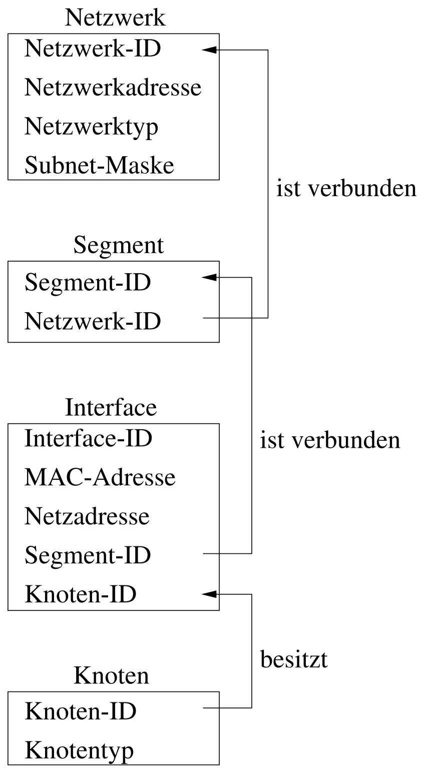

Abbildung 10.1: Informationsmodell mit Top-down Ansatz

Die Verbindungen werden dabei über die zugeordneten Identifier beschrieben, so z.B. wird ein Interface über eine zugeordnete Knoten-ID dem entsprechenden Knoten zugeordnet.

2.Schritt: Bottom-up

Das im ersten Schritt entwickelt grobe Informationsmodell wird im Bottom-up Verfahren in zweierlei Hinsicht verändert:

- Erweiterung durch neue topologische Managementinformationen:
  unter NMS wird ein Segment zusätzlich über seinen Segmenttyp (Ethernet, Tokenring) beschrieben.

- Das Informationsmodell wird an die bestehenden Schnittstellen angepaßt:
  Die Operationen auf Knoten (z.B. die Einfügeoperation) unter NMS und HPOV benötigen außer den Knotenidentifier oder den Knotentyp auch noch die MAC-Adresse, die Netzwerkadresse und die Segment-ID.

Nach diesen Veränderung (im Bild grau unterlegt) ergibt sich folgendes Bild des Informationsmodells:

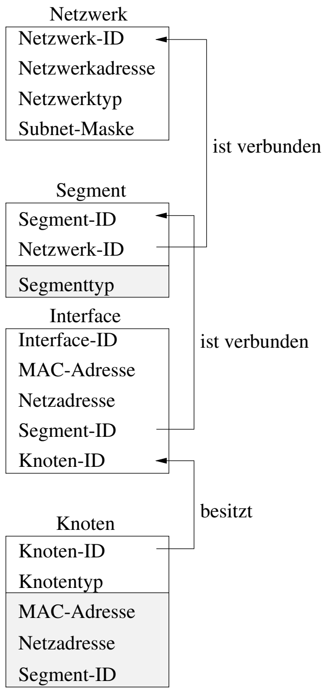

Abbildung 10.2: Über Bottom-up Ansatz erweitertes Informationsmodell

Aufbauend auf diesem Informationsmodell der Netztopologie lassen sich Schnittstellenfunktionen systematisch spezifizieren. Dabei ist zu bemerken, daß die es nicht nur *eine* Möglichkeit zur Schnittstellenspezifikation gibt; die Schnittstellenspezifikation kann in Abhängigkeit von der Erfüllung verschiedener Randbedingungen - wie z.B. Spezifizierung nur einer Menge von Basisfunktionen - abhängig gemacht werden.
Der folgende Abschnitt spezifiziert eine Menge von Basisfunktionen und bildet sie auf die entsprechenden Plattformfunktionen ab.

10.3.2 Abbildung auf die Plattformen

In diesem Abschnitt werden tabellarisch alle notwendigen Basisfunktionen, ihre (informelle) Beschreibung und ihre Abbildung auf die Plattformfunktionen dargestellt.

Netzwerk-Funktionen

Einfügen

Semantik:	fügt neues Netzwerk in die Topologie-DB ein	
Eingabeparameter:	Netzwerkadresse	
Rückgabeparameter:	Netzwerk-ID	
Abbildung:	NMS	HPOV
	DBEnumWorlds DBGetWorld DBAddNetwork	OVADLCreateNetwork

Löschen

Semantik:	entfernt ein Netzwerk aus der Topologie-DB	
Eingabeparameter:	Netzwerk-ID	
Abbildung:	NMS	HPOV
	DBDeleteNetwork	- (vgl. Bemerkung 3)

Aufzählen

Semantik:	zählt alle Netzwerke in der Topologie-DB auf	
Rückgabeparameter:	Netzwerkadresse, Netzwerktyp, Subnetmaske	
Abbildung:	NMS	HPOV
	Btrieve Requests (vgl. Bemerkung 1)	OVADLGetNextNetwork OVADLGetNetworkSubnetMask

Segment-Funktionen

Einfügen

Semantik:	fügt neues Segment in die Topologie-DB ein	
Eingabeparameter:	Netzwerk-ID, Segmenttyp	
Rückgabeparameter:	Segment-ID	
Abbildung:	NMS	HPOV
	DBAddSegment	OVADLCreateSegment (vgl. Bemerkung 2)

Löschen

Semantik:	entfernt ein Segment aus der Topologie-DB	
Eingabeparameter:	Segment-ID	
Abbildung:	NMS	HPOV
	DBDeleteSegment	OVADLDeleteSegment

Aufzählen

Semantik:	zählt alle Segmente eines Netzwerks auf	
Rückgabeparameter:	Netzwerk-ID, Segment-ID, Segmenttyp	
Abbildung:	NMS	HPOV
	Btrieve Requests (vgl. Bemerkung 1)	OVADLGetNextSegmentInNetwork

Interface-Funktionen

Einfügen

Semantik:	fügt neues Interface in die Topologie-DB ein	
Eingabeparameter:	MAC-Adresse, Knotenadresse, Segment-ID, Knoten-ID	
Rückgabeparameter:	Interface-ID	
Abbildung:	NMS	HPOV
	DBAddDeviceToBox	OVADLGetNodeInfo OVADLCreateIfc OVADLAddIfcToNode OVADLAddNodeToSegment

Löschen

Semantik:	entfernt ein Interface aus der Topologie-DB	
Eingabeparameter:	Interface-ID	
Abbildung:	NMS	HPOV
Fall 1:	Knoten besitzt nur ein Interface	
	DBDeleteBox	OVADLRemoveIfcsFromNode OVADLRemoveNodeFromSegment
Fall 2:	Knoten besitzt mehrere Interfaces	
	DBDeleteDevice	OVADLGetNodeInfo OVADLGetIfcInfo OVADLGetSegment OVADLRemoveIfcsFromNode

Aufzählen

Semantik:	zählt alle Interfaces eines Knotens auf	
Eingabeparameter:	Knoten-ID	
Rückgabeparameter:	MAC-Adresse, Netzwerkadresse, Segment-ID	
Abbildung:	NMS	HPOV
	Btrieve Requests (vgl. Bemerkung 1)	OVADLGetNodeInfo OVADLGetIfcInfo OVADLGetSegment

Knoten-Funktionen

Einfügen

Semantik:	fügt neuen Knoten in die Topologie-DB ein	
Eingabeparameter:	Knotentyp, MAC-Adresse, Netzwerkadresse, Segment-ID	
Rückgabeparameter:	Knoten-ID	
Abbildung:	NMS	HPOV
	DBAddBox	OVADLRegisterNodeInfo OVADLGetNextNodeInNetwork OVADLAddNodeToSegment

Löschen

Semantik:	entfernt einen Knoten aus der Topologie-DB	
Eingabeparameter:	Knoten-ID	
Abbildung:	NMS	HPOV
	DBDeleteBox	- (vgl. Bemerkung 3)

Bemerkungen:

1. Die Aufzählfunktionen unter NMS (DBEnum..) verwenden Callbackfunktionen [Nov93c], während HPOV einen Aufzählmechanismus jeweils über die Kontrollstrukturen der Programmiersprache und OVADLGetNext-Funktionen implementiert. Der gleiche Mechanismus kann auch auf NMS implementiert werden, wenn statt der NMS DB-APIs die Btrieve-APIs GET FIRST und GET NEXT (vgl. [Sch92]) für Datensätze verwendet werden.

2. Unter HPOV wird kein Segmenttyp implementiert.

3. Unter HPOV sind die Löschen-Funktionen oft nicht implementiert (z.B. für Knoten und Netzwerke); dies hat jedoch keinen Einfluß auf die „sichtbare“Funktionalität der Schnittstelle.

Aufzählen

Semantik:	zählt alle Knoten in einem Segment auf	
Eingabeparameter:	Segment-ID	
Rückgabeparameter:	Knoten-ID, Knotentyp	
Abbildung:	NMS	HPOV
	Btrieve Requests (vgl. Bemerkung 1)	OVADLGetNodeInSegment OVADLGetNodeInfo

Kapitel 11

Die Schnittstelle zum Konfigurationsmanagement

Eine der Hauptaufgaben des Netzmanagements ist es, die vorhandenen Ressourcen so zu verknüpfen und anzupassen, daß die gewünschte Kommunikationsleistung oder Systemfunktion erbracht werden kann (vgl. [Heg93]). Der funktionale Bereich innerhalb des Netz- und Systemmanagements, der sich mit dieser Aufgabe beschäftigt, ist das Konfigurationsmanagement.
Voraussetzung für die Erfüllung dieser Aufgabe ist die Kenntnis aller Netz- und Systemkomponenten. Diese Informationen sind in der Netzbeschreibung enthalten; die Netzbeschreibung gliedert sich in geographische, topologische sowie organisatorische Informationen und Komponenteninformationen(vgl. auch [Heg93]). Eine Teilaufgabe innerhalb des Konfigurationsmanagements ist demnach die Beschaffung dieser Informationen. Im Rahmen dieser Arbeit beschränkt sich Konfigurationsmanagement dabei auf die Komponenteninformationen, die nicht vom Topologiemanagement behandelt werden; die globalen Netzbeschreibungen werden ebenfalls dem Topologiemanagement zugeordnet.
In diesem Kapitel werden zunächst die Anwendungen auf jeder Plattform vorgestellt, die sich dem Konfigurationsmanagement zuordnen lassen. Eine Untersuchung der APIs im nächsten Schritt führt anschließend zur Erörterung möglicher Portabilitätskonzepte.

11.1 Konfigurationsmanagement auf den Plattformen

Konfigurationsmanagement unter NMS

NMS implementiert im Bereich Konfigurationsmanagement insgesamt drei Anwendungen:

- Den SNMP MIB-Browser für das Management von SNMP-Komponenten.

- Sogenannte Dialog Books für die Anzeige von Konfigurationsinformationen von Komponenten und Segmenten.
- Das NetWare RCONSOLE Werkzeug für die Konfigurierung von NetWare Servern.

SNMP MIB-Browser Mit dem MIB-Browser lassen sich SNMP MIB-Variablen sowohl abfragen als auch verändern. Diese Operationen werden auf der Basis sogenannter *Profiles* durchgeführt; ein Profil enthält:

- Eine textuelle Beschreibung des Profile.
- Den zu verwendenden Community String.
- Die Art der Operationsdurchführung: einmalig oder gepollt.
- Evtl. das Polling Intervall.
- Die Art der Ergebnisanzeige: tabellarisch oder graphisch (bei Polling).
- Die abzufragenden bzw. zu ändernden MIB-Variablen.

Generelle Voraussetzung für die Durchführung dieser Operationen ist allerdings, daß dem MIB-Browser die entsprechenden MIB-Informationen zugänglich gemacht werden. Dafür gibt es mehrere Gründe:

- In dem Dialogfenster zur Profile-Erstellung werden MIB-Objektgruppen und MIB-Variablen über ihre Namen angezeigt also z.B. als „icmp“oder „system“. Also müssen die Namen auch verfügbar sein.
- Der Benutzer kann sich im gleichen Dialogfenster auch textuelle Informationen über MIB-Variablen - i.e. die Beschriebung der Variable im DESCRIPTION-Feld ihrer ASN.1 Definition - anzeigen lassen.
- Der MIB-Browser führt auf der Grundlage der Profiles entsprechende Get- bzw. Set-Requests aus. Dazu muß jedoch den Variablennamen der passende Objektidentifier und der passende Syntaxtyp zugeordnet werden können.

Aus diesen Gründen werden alle benötigten MIBs durch den MIBC MIB-Compiler von NMS übersetzt und deren Informationen so dem MIB-Browser zugänglich gemacht. Der MIB-Compiler generiert bzw. erweitert eine binäre Datei; diese Datei bildet dabei die MIB-Strukturen entsprechend dem Objektidentifikator-Baum nach (vgl. [Nov92a]).

Dialog Books Zur Darstellung der Konfiguration von Komponenten und Segmenten verwendet NMS sogenannte Dialog Books. Ähnlich wie Bücher, enthält auch ein Dialog Book mehrere Seiten, die Dialog Box Pages. Die Dialog Box Pages zeigen jeweils logisch zusammengehörige Informationen an: so gibt es beispielsweise für Komponenten Seiten für die System Informationen, Disk Informationen oder Contact Informationen. Bei den Daten handelt es sich dabei jeweils um die in der Btrieve-DB abgespeicherten Daten.
Dabei muß darauf hingewiesen werden, daß NMS außer dem NetExplorer keine Basisanwendung implementiert, die entsprechende Informationen sammelt; NMS stellt lediglich eine DB zur Verfügung, die entsprechende Informationen aufnehmen kann. Der Netzverwalter hat jedoch die Möglichkeit, Konfigurationsinformationen per Hand über Dialogfenster einzugeben.

RCONSOLE RCONSOLE ist ein NetWare-Dienstprogramm, daß die Fernsteuerung einer Fileserver-Konsole von einer Arbeitsstation aus ermöglicht. Bildschirm und Tastatur des Servers werden dabei auf die Arbeitstation umgeleitet, so daß auf ihr alle Funktionen der Serverkonsole verfügbar sind.(vgl. [Kar94])

Konfigurationsmanagement unter HPOV

Das Konfigurationsmanagement unter HPOV implementiert eine Anwendung mit der gleichen Funktionalität wie der NMS SNMP MIB-Browser unter dem Namen SNMP Manager.
Unterschiede gibt es lediglich bei der Konfiguration der Abfrage- und Änderungsfunktionen: unter HPOV werden die betroffenen MIB-Variablen in sogenannten *Queries* festgelegt. Die Festlegung der Optionen für Polling und Ergebnisdarstellung erfolgt jedoch separat ohne Verbindung zur Query.
MIBs werden unter HPOV ebenfalls übersetzt und in der sogenannten Manager DB abgespeichert.

Konfigurationsmanagement unter SMS

Als Konfigurationsanwendung implementiert SMS den Inventory-Prozeß; Diese Anwendung inventarisiert alle Server und PCs des Netzes. Grundlage sind dabei die Server MIF Definition und die PC MIF Definition der DMTF (Desktop Management Task Force).

11.2 APIs zum Konfigurationsmanagement der Plattformen

NMS

NMS dokumentiert das Format der Binärdatei mit den übersetzten MIBs in [Nov92a] und ermöglicht auf dieser Basis den Zugriff auf die MIB-Informationen; separate APIs werden jedoch nicht zur Verfügung gestellt.
Darüberhinaus können einige APIs der Btrieve-DB zu den APIs des NMS-Konfigurationsmanagements gerechnet werden (vgl. 11.1):
Das DB-Schema, das die Btrieve-DB implementiert, basiert auf einem objektorientierten Modell des Netzes; das Modell legt Objektklassen und (potentielle) Beziehungen zwischen Objektinstanzen fest. Im DB-Schema werden diese Objektinstanzen und Beziehungen über Tabellen repräsentiert.
Abbildung 11.1 zeigt die wesentlichen Objektklassen und Beziehungen des Modells und verdeutlicht gleichzeitig die Aufteilung zwischen zwischen topologierelevanten auf der einen und konfigurationsrelevanten Objektklassen auf der anderen Seite; Die Unterteilung der DB-Tabellen und ihrer APIs ist dementsprechend. Dabei werden Knoten über Instanzen der Klasse BOX dargestellt, Interfaces über

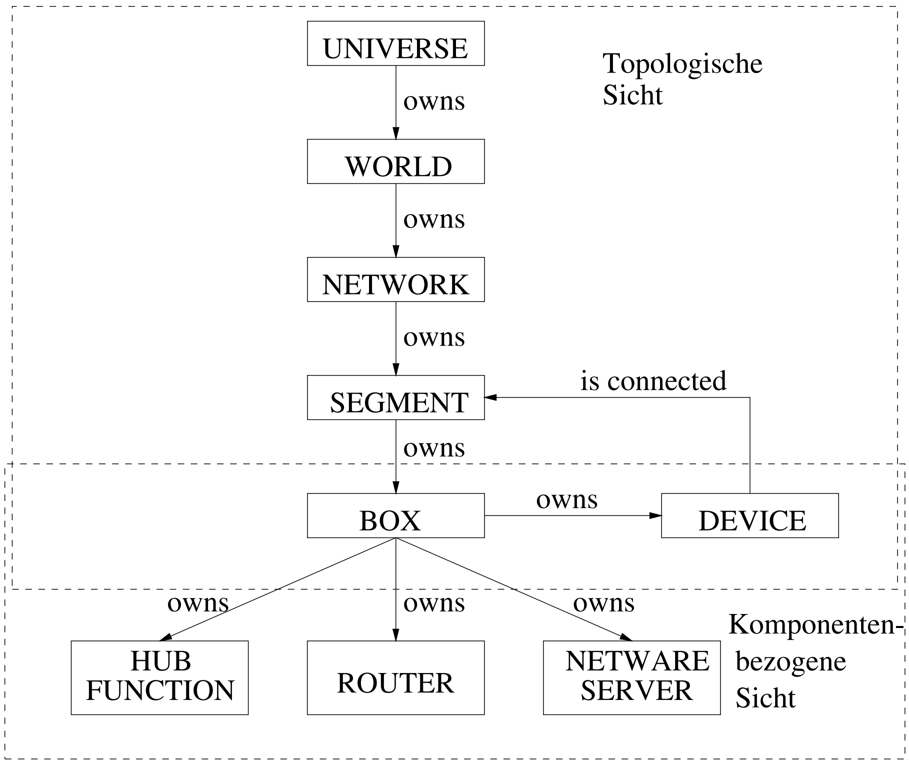

Abbildung 11.1: Topologie- und Konfigurationsrelevante Objekte des NMS-Informationmodells

Instanzen der Klasse DEVICE und die Knotenfunktionen über Instanzen verschiedener Funktionenklassen wie z.B. NETWARE SERVER und ROUTER.

HPOV

Im Gegensatz zu NMS implementiert kein Datenbankschema für Konfigurationsdaten, stellt dafür aber APIs (OVMib.. APIs) für den Zugriff auf die kompilierten MIBs zur Verfügung. Die APIs implementieren dabei im wesentlichen Funktionen für das „Durchwandern "des Objektidentifier-Baums sowie für den Zugriff auf den Namen, die Beschreibung, den Syntaxtyp und die Zugriffsrechte einer MIB-Variablen bei gegebenem Objektidentifier (vgl. [HP93]).

SMS

SMS stellt für den Zugriff auf die Server- und PC-Informationen die SMS Data APIs zur Verfügung; der Zugriff auf diese Daten kann jedoch auch direkt über die SQL Server APIs oder ODBC erfolgen.
Die SMS Data APIs ermöglichen den objektorientierten Zugriff auf die SMS-Datenbank, d.h. die Transfer-Einheit zwischen Anwendung und Datenbank ist das einzelne Objekt (vgl. [SMS95c]):
Eine Vermittlerfunktion zwischen Anwendung und Datenbank übernehmen dabei die sogenannten *Container*. Datenbankobjekte (wie z.B. inventarisierte Systemkomponenten), auf die zugegriffen werden soll, müssen dazu zunächst - evtl. gefiltert - aus der Datenbank in den Container übertragen werden; erst dann kann die Anwendung auf die Objekte zugreifen. Im Container werden die Objekte über sogenannte *Folder* repräsentiert, die selbst wieder aus Subfoldern oder *Skalaren* bestehen. Die Skalare stellen dabei die Attributwerte der Objekte im Container dar.
Die Folder sowie die Skalare der Folder im Container können mit Hilfe der SMS Data APIs jeweils entweder iterativ oder gezielt über ihren Identifier abgefragt werden.
Um umgekehrt neue Objekte abzuspeichern, müssen dementsprechende Skalare und Folder erzeugt werden, die dann der Datenbank übermittelt werden.

11.3 Portabilitätskonzept

11.3.1 Die Schnittstelle zu den statische MIB-Daten

Die portierbare Schnittstelle für die statischen MIB-Daten muß eine Funktionalität bereitstellen, die den Zugriff auf den ASN.1-Syntaxtyp, den Namen, die Beschreibung und die Zugriffsrechte jeder MIB-Variablen der compilierten MIBs ermöglicht (Top-Down).

Als zusätzlicher Orientierungspunkt (Bottom-up) für die Spezifikation der notwendigen Schnittstellenfunktionen können dabei die entsprechenden HPOV-APIs dienen (Minimierung des Implementierungsaufwands für die Zwischenschichten, vgl. 5.2).
Eine Spezifikation der im folgenden nur informell beschriebenen Funktionen stellt jedoch nur *eine* Möglichkeit dar, eine derartige Schnittstelle festzulegen:

- Funktionen, die zu einem gegebenen Objektidentifier den Objektidentifier des vorangehenden Nachbarknotens, des folgenden Nachbarknotens, des Elternknotens bzw. des ersten Kindknotens zurückgeben.
- Eine Funktion, die bei gegebenem Namen der MIB-Variable deren Objektidentifier liefert.
- Eine Funktion, die die statischen Daten einer MIB-Variable zurückgibt.

11.3.2 Die Schnittstelle zu Server- und PC-Informationen

Nur SMS implementiert eine Anwendung, die Konfigurationsinformationen über die Server und PCs des verwalteten Netzes sammelt.
Um hier eine einheitliche Schnittstelle schaffen zu können, müssen zunächst auch für NMS und HPOV entsprechende Managementanwendungen nachimplementiert werden. Eine Implementierung ist prinzipiell möglich, da sich die MIF-Objekte auf entsprechende SNMP MIB-Variablen abbilden lassen (vgl. [DMT95a]) und diese Informationen daher auch über SNMP abgefragt werden können.
Eine einheitliche Datenbankschnittstelle läßt sich durch den Einsatz von ODBC erreichen:
Das DB-Schema der SMS-Datenbank kann über ODBC abgefragt werden und dann ebenfalls über ODBC auf den ODBC-fähigen DBSen von NMS und HPOV implementiert werden. Der Zugriff auf die DBen kann dann einheitlich über ODBC erfolgen.

Kapitel 12

Die Schnittstelle zur graphischen Benutzeroberfläche

Die Plattformen NMS und HPOV bieten verschiedene Tools an, die von Anwendungen zur graphischen Darstellung von Informationen genutzt werden können. In diesem Kapitel werden die Graphik-Tools dieser zwei Plattformen beschrieben und hinsichtlich ihrer Portierbarkeit untersucht.

12.1 Die Graphik-Tools der Plattformen

Die Graphik-Tools unter NMS

Die Schnittstellen von NMS zur graphischen Benutzeroberfläche sind als DLLs bzw. als Kombination aus DLLs und statischen Bibliotheken implementiert (vgl. [Nov92b] und müssen dementsprechend eingebunden werden:
Die entsprechende Header-Datei muß im Source-Code der Anwendung enthalten sein und die entsprechende statische Bibliothek beim Linken angegeben werden; die DLL muß über eine entsprechend gesetzte Umgebungsvariable erreichbar sein. NMS bietet im wesentlichen folgende graphischen Tools an:

Dialog Book Tool: Mit diesem Tool kann eine Anwendung mehrere Dokumentseiten innerhalb *eines* MDI Child-Fensters (Multiple Document Interface) - i.e. der Arbeitsbereichs des (Haupt-)Dialog Book Fensters - oder über mehrere modale Dialogfenster, die vom Dialog Book Fenster aufgerufen werden, definieren.

Gauge Tool: Dieses Tool erlaubt die graphische Darstellung und Überwachung von Variablen anhand einer skalierten Säule, auf der der aktuelle Wert der Variablen, die bisherigen Min- und Max-Werte sowie die eingestellten Höchst- und Tiefstwerte (Thresholds) abgelesen werden können. Bei Über bzw. Unterschreitung der eingestellten Thresholds kann die Auslösung eines Alarms konfiguriert werden.

Graph Tool: Mit diesem Tool kann die zeitliche Veränderung eines Variablenwertes graphisch dargestellt werden.

Legend Tool: Mit Hilfe dieses Tools kann die Legende zu einer Graphik konfiguriert werden.

Status Bar Tool: Das Status Bar Tool erlaubt die Generierung eines benutzerdefinierten Steuerelements, das u.a. Felder für die aktuelle Uhrzeit, das aktuelle Datum, Textanzeige über Ticker Tape, statischen Text etc. enthalten kann.

Graphik-Tools unter HPOV

HPOV bietet insgesamt sieben VBX-Steuerelemente an, über die SNMP-Variablen angezeigt und SNMP-Komponenten so kontrolliert werden können, u.a.:

Meter Control: Diese Steuerelemente ermöglichen die Darstellung von Integer- und Counter-Variablenwerten über graphischen Darstellung analoger Meßinstrumente (Meters).

LED Control: Damit werden die Werte „binärer"Variablen über eine graphisch dargestellte LED angezeigt.

Text Box Control: Die Text Box zeigt alphanumerischer Variablen als Text bzw. Octet String Variablen als hexadezimale Werte an.

AC Jack Control: Mit diesem Steuerelement kann der Status der Stromversorgung angezeigt werden.

12.2 Portabilität

Da die NMS Graphik-Tools in separaten Windows-Bibliotheken implementiert sind, sind sie unter Windows 3.1 bzw. allgemein unter 16-bit Windows auch portierbar.
Eine Portierung auf SMS ist dagegen nicht möglich:
SMS-Managementanwendungen sind 32-bit Anwendungen, da SMS selbst für 32-bit Umgebungen implementiert ist. Eine Einbindung der 16-bit Graphik-Bibliotheken von NMS in 32-bit Anwendungen ist aber nicht möglich (vgl. [Kru94]).
Auch eine Implementierung von 16-bit Managementanwendungen in Verbindung mit dem 16-bit Subsystem von NT (vgl. Kapitel 13) ist deswegen ausgeschlossen,

weil Anwendungen auf dem Subsystem keinerlei Verbindung zu anderen (32-bit) Prozessen haben (vgl. [Sta94]).

Kapitel 13

Die Betriebssystemschnittstelle

Jede der drei Plattformen setzt auf einem Betriebssystem auf; Managementanwendungen können daher zusätzlich zu den Funktionen der Plattformen auch Funktionen, die vom Betriebssystem über APIs zur Verfügung gestellt werden, aufrufen.
Dieses Kapitel untersucht, welche Probleme in diesem Zusammenhang bei der Portierung auftreten können und wie diese Probleme durch die Verwendung der Microsoft Foundation Classes (MFC) zu einem großen Teil vermieden werden können.
Inwieweit alternativ zu den MFCs auch der Einsatz der ObjectWindows Library (OWL) von Borland möglich wäre wird hier nicht untersucht.

13.1 Schnittstellen auf den Plattformen

NMS und HPOV laufen beide unter dem Betriebssystem Windows 3.1 (genau genommen ist Windows nur eine Betriebssystemerweiterung von DOS, vgl. [Hon92]), während SMS auf dem portablen und vollständigen Betriebssystem Windows NT aufsetzt.
Beide Betriebssysteme stellen Anwendungsprogrammierern APIs für die Programmierung in C bzw. C++ zur Verfügung. Im Unterschied zu Windows 3.1, daß als 16-bit-Betriebssystem implementiert wurde, ist Windows NT ein 32-bit-System; die über die APIs aufrufbaren Funktionen sind dementsprechend 16-bit- bzw. 32-bit-Funktionen.

13.2 Probleme bei der Portierung

Eine der Ursachen für die Probleme, die bei der Portierung von Anwendungen Windows 3.1 nach NT und umgekehrt auftreten, liegt eben darin, daß Windows 3.1 Anwendungen 16-bit- und NT-Anwendungen 32-bit-Anwendungen sind.

13.2.1 Portierung von Windows 3.1 auf NT

Unter Windows NT steht ein 16-bit-Subsystem zur Verfügung, das alle Umgebungseigenschaften für Windows 3.x Anwendungen zur Verfügung stellt. Grundlage dieses Subsystems ist die sogenannte Virtual DOS Machine (VDM), ein Prozeß, der einen kompletten DOS-Computer simuliert, und in dessen Adreßraum das Subsystem als eigenständige Anwendung abläuft.(vgl. [Sta94])
Trotzdem können beim Ablauf eines 16-bit-Programms auf dem Subsystem Probleme auftreten:

- Windows NT verfügt über ein verbessertes Dateisystem mit Sicherheitsmechanismen ([Kru94]). Die dabei realisierte Sicherheitstechnik erlaubt aber keinen absoluten und unmittelbaren Zugriff auf Dateien und Speicherbereiche ([Sta94]). Entsprechende Aufrufe von Win16-Funktionen führen daher zu Fehlermeldungen.

- Win16-Funktionen und Win32-Funktionen gleicher Funktionalität unterscheiden sich u.U. in den verwendeten Funktionsprototypen, z.B. werden andere Funktionsparameter verwendet.

- Die Breite verschiedener Datentypen ist unter Windows3.1 und Windows NT unterschiedlich, z.B. ist der Datentyp Integer unter Windows 3.1 16 Bit und unter NT 32 Bit breit ([Kru94]).

- Das Win32-API unterstützt im Gegensatz zu dem Win16-API keine Einbindung von VBX-Steuerelementen (Visual Basic Extensions), sondern nur OCX-Steuerelemente (OLE Custom Extensions).

Eine Konvertierung existierender Anwendungen für 16-bit-Windows in richtige 32-bit-Anwendungen ist zwar grundsätzlich möglich, erfordert jedoch einen hohen Aufwand ([Kru94]).

13.2.2 Portierung von NT auf Windows 3.1

Für Windows 3.1 existiert ein SDK, der die Win32-API unterstützt: Win32s (neueste Version Win32s v1.3); mit Win32s ist es möglich, 32-bit-Anwendungen auch unter Windows 3.1 auszuführen ([Kru94]). Aber auch hier treten Probleme auf:

- Das Win32-API enthält viele neue Funktionen, die von Win32s - das lediglich auf 16-bit-Windows aufsetzt - nicht unterstützt werden wie z.B. Mehrfach-Threads ([Kru94]):
  Threads sind die ausführenden Einheiten eines Prozesses; der Prozeß - als Instanz einer Anwendung - wird in diesem Zusammenhang lediglich als der Adreßraum angesehen, in dem der Programmcode abgelegt ist ([Ric94]).

Werden diese in Win32s zwar vorhandenen Funktionen aufgerufen, wird nur eine Fehlermeldung ausgegeben.

- Ebenso wie in die umgekehrte Richtung besteht auch hier bei der Portierung das Problem, die Funktionsprototypen anzupassen.
- Anpassung unterschiedlich breiter Datentypen.

13.3 Portabilität durch die Programmierung mit MFCs

VC++ ist ein Entwicklungssystem für Programmierung von Windows-Anwendungen auf der Basis von C++ ([Kru94]). Neben Entwicklungstools, wie z.B. dem AppWizard, ein Quelltextgenerator, der ein lauffähiges Grundgerüst erzeugt oder dem ClassWizard, der die Erzeugung und Verwaltung neuer C++-Klassen unterstützt, ist Klassenbibliothek, sogenannte Microsoft Foundation Class Library (MFC-Library) der zentrale Bestandteil von VC++.
Die MFC Library ist im Hinblick auf Portabilität entwickelt worden, so daß die meisten Anwendungen, die die MFC benutzen, leicht von Windows 3.1 auf Win32-Plattformen (Windows NT, Windows 95) portiert werden können. Im allgemeinen genügt dazu ein erneutes Kompilieren und Binden. ([Kru94])
Erreicht wird diese Portabilität dadurch, daß die die Windows APIs in den MFCs gekapselt werden und so direkte Aufrufe von Windows-Funktionen überflüssig werden. Die Anpassung der gekapselten Windows-Funktionsaufrufe an die jeweilige Umgebungen übernimmt dann ein entsprechender Präprozessor und der Compiler. Das Problem der in dem Win16-API nicht implementierten Funktionen bleibt allerdings bestehen.
Die Unterstützung der unterschiedlichen Steuerelemente betrifft auch die MFCs: VC++ 1.5 mit den MFC 2.5 läuft unter Windows 3.1, Versionen ab VC++ 2.0 und MFC 3.0 laufen nur noch unter Windows NT. Die MFC 2.5 unterstützen dementsprechend die Einbindung von VCX-Controls, während ab MFC 3.0 OCXe verwendet werden müssen. Allerdings ist laut Aussage Microsoft zu erwarten, daß die entsprechenden OCX-Controls mit VC++ 4.0 verfügbar sind.

Kapitel 14

Zusammenfassung und Ausblick

Portabilität auf SMS

SMS ist nicht als PC-Management-Plattform im üblichen Sinn konzipiert und stellt Anwendungsprogrammierern dementsprechend auch keine APIs zur Ergänzung und Erweiterung bereits implementierter Basisanwendungen zur Verfügung. Darüberhinaus unterscheiden sich die auf SMS implementierten Anwendungen aufgrund der anderen Zielsetzung - Systemmanagement von PCs und Servern in unternehmensweiten Netzen - in ihrer Funktionalität deutlich von derjenigen der Basisanwendungen von NMS und HPOV (Software Distribution, Remote Troubleshooting and Control).
Die Portabilität von Managementanwendungen auf SMS läßt sich aus diesen Gründen nur durch die Nachimplementierung entsprechender Anwendungen und Schnittstellen erreichen; für die Netzkommunikation können dagegen die entsprechenden VC++ -Module zur SNMP-Kommunikation verwendet werden (vgl. Einleitung Kapitel 8).

Portabilität auf NMS und HPOV

Die Funktionalität der Basisanwendungen in den einzelnen Managementbereichen ist für NMS und HPOV weitgehend identisch; Unterschiede bestehen dabei lediglich beim Alarmmanagement hinsichtlich der Schnittstelle zur Alarmgenerierung und bei der unter NMS zusätzlich bereitgestellten DB für Konfigurationsinformationen.

Das Problem der Portierbarkeit resultiert jedoch hier in erster Linie aus den unterschiedlichen Schnittstellenimplementierungen.
Das schwerwiegendste Teilproblem bildet dabei die einheitliche Anbindung der jeweiligen Datenbanken. Die Anbindung spezieller proprietärer und nicht ODBC-fähiger DBen kann jeweils nur über die Entwicklung ebenso spezieller gemeinsamer Schnittstellen erreicht werden. Ein erster Schritt zur Verbesserung der Por-

tierbarkeit von Managementanwendung kann hier die Verwendung von DBSen sein, für die die Verwendung einer standardisierten Schnittstelle zur Anbindung - wie z.B. ODBC - zumindest *möglich* ist.
Für ODBC-fähige DBen liegt dann das Problem in der Implementierung jeweils unterschiedlicher DB-Schemata; die jeweiligen DB-Schemata beschreiben die gleichen Objekte auf der Basis unterschiedlicher Tabellen, Tabellennamen und Attributnamen. Deswegen kann ODBC als Standardschnittstelle zur DB-Anbindung bereits implementierter DBen auf den Plattformen nicht eingesetzt werden.
Portabilität könnte beispielsweise dadurch erreicht werden, daß die betreffenden DB-Schemata vereinheitlicht werden; die jeweiligen DB-Objekte - z.B. Alarme, Netzwerke, Segmente etc. - müssten einheitlich modelliert werden, d.h. im DB-Schema über die gleichen Tabellen mit den gleichen Attributen und den gleichen Tabellen und Attributnamen beschrieben werden; die Anbindung über ODBC wäre dann problemlos möglich. Eine Vereinheitlichung ist in vielen Fällen wie z.B. im Bereich der Topologiedatenbanken schon deshalb leicht möglich, weil sowohl bei den Objekten, die in den DB-Schemata beschrieben werden, als auch bei deren Beschreibung weitgehend Konsens herrscht.

Ein anderes Problem ist die Definition einer Schnittstelle für die Alarmgenerierung durch Anwendungen; der Einsatz des Loopback-Mechanismus - auch auf HPOV - bietet hier die Möglichkeit, die vorhandene SNMP-Schnittstelle zu verwenden. Die standardisierte SNMP-Schnittstelle wäre gleichzeitig auch eine standardisierte Schnittstelle zur Alarmgenerierung. Voraussetzung ist allerdings, daß auch für diese Alarme Traps definiert werden und die entsprechenden MIBs in die Alarmsysteme der Plattformen integriert werden wie alle anderen Alarm-MIBs auch.

Anhang A

WinSNMP

Die WinSNMP/Manager API - im folgenden mit WinSNMP abgekürzt - spezifizieren eine Programmierschnittstelle für SNMP-basierte Netzmanagementanwendungen unter MS-Windows GUI/Betriebssystemen, also Windows 3.0 und 3.1, WfW 3.11 sowie Windows NT (vgl. [Nat94]).
WinSNMP ist das Ergebnis der Zusammenarbeit einer Gruppe von Herstellern, Entwicklern und Benutzern. Ziel war es, eine SNMP-Schnittstelle zu spezifizieren, die von SW-Herstellern implementiert und von Anwendungsprogrammierern benutzt werden kann, und so die Verbreitung SNMP-basierter Managementanwendungen in MS-Windows Umgebungen zu fördern. Dabei sollten jedoch keinerlei Einschränkungen hinsichtlich der Benutzung von SNMPv1 oder SNMPv2 und der von diesen Protokollen unterstützten Funktionalität gemacht werden.([Nat94])
Hauptmerkmale einer WinSNMP-Implementierung sind:

- Die ASN.1-Darstellung der Komponenten einer SNMP-Nachricht in einer logischen Zwischenschicht, die Kodierung nach BER sowie verschiedene Protokolldetails wie z.B. der genaue Aufbau einer SNMP-PDU werden gekapselt.

- Die Protokollversionen SNMPv1 und SNMPv2 werden durch die einheitliche Schnittstellen unterstützt. Anwendungsprogrammierer können sowohl v1- wie auch v2-Requests senden, ohne sich dabei darum kümmern zu müssen, welche Version der Agent der Zielmaschine unterstützt. Da die beiden Protokolle verschiedene Formate verwenden, muß die Implementierung evtl. eine Konvertierung vornehmen. SNMP-Responses bzw. Traps werden jeweils im v2-Format an die Anwendung weitergegeben; auch hier erfolgt die evtl. notwendige Konvertierung durch die Implementierung gemäß [RFC93a]. Die Analyse der empfangenen Traps muß ebenfalls auf der Basis des SNMPv2-Formats für Traps erfolgen.

- Unterstützung verschiedener Zuordnungsservices: einige WinSNMP-Funktionen benötigen Entities bzw. Contexts als Argumente. Den Entities entsprechen in der SNMPv1- bzw. SNMPv2-Terminologie Agentenadressen bzw. Parties, den Contexts Community-Strings bzw. Context IDs. Eine WinSNMP-Implementierung unterstützt für jede Protokollversion zwei verschiedene Translationmodes, die festlegen, in welcher Form diese Argumente übergeben werden. Eine Agentenadresse kann entweder als benutzerfreundlicher Name (z.B. Main_Hub) oder in der dotted Version (z.B. 192.151.207.34) übergeben werden.

- Unterstützung der Anwendung bei der wiederholten Übertragung von Requests (Retransmission): in erster Linie ist die Anwendung selbst für den Bereich Retransmission zuständig. Aufgabe der Implementierung ist es dabei, die Anwendung mit Hilfe abgespeicherter, zielrechnerspezifischer Vorgaben für die Anzahl der Wiederholungsversuche (Retries) und das Zeitlimit zwischen zwei Wiederholungen (Timeout) zu unterstützen. Darüber hinaus *kann* eine Implementierung auch die Ausführung der Wiederholungübertragung durch die Implementierung selbst auf der Basis dieser (Default-)Werte unterstützen.
  Das Konzept der lokalen Datenbank stellt dabei die Basis sowohl der Zuordnungsservices als auch für die Wiederholungsübertragung dar; alle erforderlichen Daten werden in dieser konzeptuellen Datenbank gehalten.

An einer neuen Version der WinSNMP-Spezifikation (WinSNMPv2.0) wird derzeit noch gearbeitet. Haupthindernis dabei ist - nach Aussage von Bob Natale - die ausstehende Entscheidung der IESG (Internet Engineering Steering Group) über den Status des aktuellen SNMPv2-Dokuments, das auch Ausgangspunkt der zukünftigen WinSNMP-Spezifikation sein wird; wahrscheinlich ist eine Spezifikation auf der Basis von SNMPv2C - i.e. SNMPv2 ohne dementsprechendes Sicherheitsmanagement (unterstützt wird nur der Community-basierte (deshalb C) Sicherheitsmechanismus von v1).

Anhang B

ODBC (Open Database Connectivity)

ODBC ist ein Standard-API für den Zugriff auf Daten in relationalen und nicht-relationalen Datenbank-Management-Systemen (DBMS)(vgl. [Gei95]). In diesem Anhang sollen die Zielsetzungen, die Randbedingungen und die Motive erläutert werden, die im Zusammenhang mit der Entwicklung von ODBC von Bedeutung waren. Darüberhinaus wird die ODBC-Architektur vorgestellt sowie einige Designentscheidungen, die helfen sollen die Zielvorgaben umzusetzen. Soweit nicht anders angegeben folgt die Darstellung dabei inhaltlich [Gei95].

Zielsetzung

Mit der Entwicklung von ODBC wurde das Ziel verfolgt, eine Standard-API für die Anbindung verschiedener DBMS zu spezifizieren und implementieren. Mit DBMS sind dabei jedoch alle Datenbankprodukten gemeint, die in irgenteiner Form Daten speichern und verwalten. Insbesondere sollte ODBC die Anbindung folgender Kategorien von Datenbankprodukten ermöglichen:

- Herkömmlicher relationaler DBMSe wie z.B. Microsoft SQL Server oder Oracle.
- Einfache sowie durch ISAM-Funktionen (Index Sequential Access Methods) verbesserte Datei-Management-Systeme wie z.B. Btrieve (vgl. [Sch92])
- Desktop-Datenbanken wie MS-Access oder Paradox.

Randbedingungen

Daneben waren bei der Entwicklung von ODBC noch mehrere Randbedingungen vorgegeben:

- Die Funktionaltät der angesprochenen DBMS sollte nicht eingeschränkt werden.
- Die Performance sollte weiterhin in etwa der der ursprünglichen APIs des jeweiligen DBMS entsprechen.
- Mehrere Anwendungen sollen gleichzeitig auf die gleiche DB zugreifen können.

Motivation

Einige der Gründe, die zur Entwicklung von ODBC führten, waren:

- Aus der Sicht des Anwendungsprogrammierers sollte ODBC u.a. folgende Vorteile bringen: zum einen sollte bei der Ankopplung an ein beliebiges DBMS der erforderliche Programmieraufwand durch die Verwendung einer standardisierten anstelle einer herstellerspezifischen Schnittstelle verringert werden. Zum anderen sollte ein und dasselbe Programm auf mehreren DBMS lauffähig sein.
- Auch von Herstellerseite gab es mehrere Gründe ODBC zu unterstützen: einmal mußte Kundenwünschen nach Unabhängigkeit von einem bestimmten DBMS Rechnung getragen werden. Darüberhinaus können Kunden dadurch aus einer viel größeren Vielzahl von Werkzeugen für ganz bestimmte Aufgaben auswählen, da jeder Software-Hersteller Lösungen anbieten kann. Ein zusätzlicher Grund ist, das die Einhaltung von ANSI- und ISO-Standards bei der Auftragsvergabe (z.B. durch die US-Regierung) eine große Rolle spielt (vgl. Abschnitt: ODBC als Standard).

Architektur

Die ODBC-Architektur basiert auf der Client-Server-Modell; dieses Modell ist allgemein genug, um einer Vielzahl von Topologien gerecht werden zu können u.a. Desktop-DBn oder Netzwerk-DBn. Die Anwendung dieses Modells erlaubt es, alle für die DB-Anbindungen notwendigen Umformungen auf der Client-Seite vorzunehmen; der Server kann auf der anderen Seite nicht erkennen, über welche Programmierschnittstelle er angesprochen wird. Die Performance und die Funktionalität des ursprünglichen DBMS, für die das Datenprotokoll und die SQL-Fähigkeiten des DBMS bestimmend sind, bleiben so weitgehend erhalten.
Die ODBC-Architektur unterscheidet im einzelnen vier Komponenten:
Anwendungen. Die Anwendungen interagieren über die Benutzeroberfläche mit dem Benutzer und rufen die ODBC-Funktionen auf.
Treiber-Manager. Der Treiber-Manager lädt die von den Anwendungen benötigten Treiber und verwaltet danach die Interaktionen zwischen Anwendung und Treiber. Dabei kann der Treiber-Manager mehrere Anwendungen und mehrere

Treiber gleichzeitig verwalten.
Treiber. Treiber verarbeiten die ODBC-Aufrufe, setzen entsprechende SQL-Aufrufe für bestimmte Datenquellen ab und geben die Ergebnisse an die Anwendungen zurück. Die Treiber übernehmen dabei auch alle Aufgaben im Zusammenhang mit der Kommunikations- und Datenbanksoftware, die für den Zugriff auf die Zieldaten nötig sind.
Datenquellen. Datenquellen bestehen aus den Datenmengen und den entsprechenden Umgebungen, bei denen es sich um Betriebssysteme, DBMSs oder Netzwerke handeln kann.

Designentscheidungen

Abfragesprache
Da sich nach der Standardisierung von SQL (Structured Query Language, neuester ANSI-Standard:SQL92) diese Datenbankabfragesprache als Sprache für relationale DBMS durchgesetzt hat (vgl. [Mar94]), stellt auch ODBC Anwendungen SQL zur Verfügung. ODBC ist dabei als sogenanntes Call Level Interface (CLI) definiert; ein CLI besteht aus einer Menge von Funktionsaufrufen in einer 3GL-Programmiersprache wie z.B. C, COBOL oder FORTRAN. Im Gegensatz dazu sind bei Embedded SQL-APIs die SQL-Anweisungen direkt in eine Hostsprache eingebettet; die Verwendung von Embedded SQL erfordert daher einen Präkompiler, der die SQL-Syntax der Syntax der Hostsprache anpaßt. Viele DBMS-Hersteller bieten neben Embedded SQL-APIs auch CLIs.
Die Zielsetzung sowohl bei der Einbettung als auch bei der CLI-Spezifikation ist, das Problem der Nicht Vollständigkeit von SQL als Programmiersprache der 4. Generation zu lösen. Nicht vollständig zu sein bedeutet für eine Programmiersprache, daß nicht jeder Algorithmus in ihnen formulierbar ist; der Grund ist, daß SQL keine Kontrollstrukturen bietet ([Mar94]).
Auch darüberhinaus bieten die beiden Programmiermodelle oft Vorteile, da die Komplexität von SQL-Anweisungen in Verzweigungen und Schleifen in der Regel reduziert werden kann und die Programmlogik durchschaubarer wird ([Mar94]).

Nebenläufigkeit Um zu erreichen, daß mehrere Anwendungen gleichzeitig den Treiber-Manager aufrufen können und der Treiber-Manager wiederum mehrere Anwendungen und Treiber gleichzeitig verwalten kann, werden Treiber-Manager und Treiber als DLLs realisiert. Dabei wird die Treiber-Manager-DLL über eine Importbibliothek eingebunden, während die Treiber-DLLs vom Treiber-Manager explizit selbst geladen und verwaltet werden (vgl. Anhang C).

Konformitätsstufen

Die Einführung von Konformitätsstufen für Standards (vgl. Support Levels für WinSNMP) gibt Entwicklern allgemein die Möglichkeit, ihren Bedürfnissen entsprechend auch nur Teile des Standards zu implementieren. Damit soll auch die schnellere Entwicklung von Standard-Implementierungen - hier also ODBC-Treibern - gefördert werden. Für ODBC sind aus diesen Gründen in zwei Bereichen jeweils drei Konformitätstufen eingeführt worden: zum einen für die Funktionsaufrufe des CLI und zum anderen für die unterstützte SQL-Ebene. Die SQL-Grammatiken der einzelnen SQL-Ebenen (Minimales SQL, Kern SQL und Erweitertes SQL) unterscheiden sich dabei hinsichtlich der Funktionalität der einzelnen Abfrageoperationen (komplexere Abfragen) und vom Umfang der unterstützten Datentypen.

ODBC als Standard

ODBC hat sich aus mindestens zwei Gründen bereits als Industriestandard etabliert:

1. Fast alle DBMS-Hersteller und die wichtigsten unabhängigen Softwarehersteller unterstützen ODBC, etwa Microsoft, Lotus, IBM, Novell etc.

2. ODBC basiert auf einem CLI der X/Open SQL Access Group, einem Zusammenschluß mehrerer DBMS-Hersteller mit dem Ziel, eine Grundlage für die Portabilität SQL-basierter Produkte verschiedener Hersteller zu schaffen. Diese CLI Spezifikation ist die Grundlage der in nächster Zeit erwarteten ANSI- und ISO-Standards. ODBC 3.0 (für 1996 geplant) soll an den ISO CLI-Standard angepaßt werden.

Anhang C

Dynamic Link Libraries

Dynamic Link Libraries (DLLs) sind Linkbibliotheken, die den Mechanismus der dynamische Bindung unterstützen; dynamische Bindung bedeutet, daß die von einer Anwendung benötigten Routinen erst zur Laufzeit eingebunden werden.
Das Konzept der dynamischen Bindung hat sich als vielseitig und nützlich erwiesen; so ist praktisch der Windows-Kernel - i.e. die Module KERNEL.EXE, GDI.EXE und USER.EXE - als Sammlung von DLLs realisiert worden [Hon92].
In diesem Anhang werden die im Zusammenhang mit der Entwicklung der dynamischen Bindung verknüpfte Zielsetzung und die sich aus ihr ergebenden Vorteile kurz beschrieben; danach werden verschiedene Methoden der dynamischen Bindung vorgestellt.

Zielsetzung

Die Entwicklung des Mechanismus der dynamischen Bindung wurde durch ein Projekt initiiert, in dessen Rahmen ein Mechanismus zur gemeinsamen Benutzung von Bibliotheksfunktionen zwischen verschiedenen Prozessen entwickelt werden sollte [Nor93].

Vorteile

Aus der dynamischen Bindung ergeben sich gegenüber der statischen Bindung, bei der der Funktionscode vom Linker aus einer statischen Bibliothek in die .EXE-Datei der Anwendung kopiert wird, u.a. folgende Vorteile:

- Einsparung von Festplattenkapazität und Hauptspeicher:
  Routinen, die von mehreren Anwendungen gebraucht werden und statisch eingebunden werden, existieren danach in mehreren separaten Kopien sowohl im Hauptspeicher als auch auf Festplatte [Pet90].

- Bei der Änderung einzelner Routinen müssen die betroffenen Anwendungen nicht neu gelinkt werden.

Methoden

Aufgabe des Linkers ist es, Funktionsreferenzen innerhalb von Programmen aufzulösen und den entsprechenden Objektcode einzubinden. Beim statischen Binden werden die entsprechenden Bibliotheken nach der betreffenden Funktion durchsucht, der dort vorhandene Code an eine bestimmte Stelle der .EXE-Datei kopiert und an Stelle des ursprünglichen Aufrufs ein entsprechender CALL gesetzt.
Für die dynamische Einbindung von Routinen gibt es im wesentlichen zwei Methoden:

1. Der Linker erzeugt anstelle der ursprünglichen Funktionsaufrufe sogenannte Relokationseinträge. Relokationseinträge sind Datenblöcke, die alle Informationen darüber enthalten, woher der Funktionscode zur Laufzeit geholt und wie er dann ausgeführt werden kann. Dazu zählen der Name der DLL und der Name der Funktion innerhalb der DLL bzw. alternativ die Ordinalzahl, mit der die Funktion dieser DLL exportiert wurde (EXPORT-Abschnitt der Moduldefinitionsdatei der DLL, vgl. unten).[Hon92]
   Es gibt zwei Möglichkeiten, dem Linker die nötigen Informationen zur Verfügung zu stellen:

   - Alle von einer Anwendung benötigten DLL-Funktionen werden zusammen mit dem Funktionsnamen bzw. der Ordinalzahl innerhalb des IMPORTS-Abschnitts der Moduldefinitionsdatei der Anwendung aufgelistet [Hon92]. Zu jeder Windows-Anwendung gehört eine Moduldefinitionsdatei, die neben Angaben zur Größe des Stacks und des Heaps einer Anwendung u.a. auch Charakteristika der Code- und Datensegmente enthält.
   - Dem Linker wird beim Aufruf eine DLL-spezifische Importbibliothek mit übergeben aus der dieser die Relokationseinträge für *alle* Funktionen einer DLL entnehmen kann.

   Ruft nun während des Programmlaufs eine Anwendung eine DLL-Funktion auf, so bewirkt der an dieser Stelle vom Linker eingefügte Relokationseintrag beim ersten Aufruf zweierlei: zum einen wird ein sogenannter Thunk (Vorspannroutine) der Modul-Verwaltungsblocks (i.e. fester Speicherbereich einer Anwendung bzw. DLL, in der Windows Verwaltungstabellen führt) aufgerufen, der den Relokationseintrag so modifiziert, daß beim nächsten Aufruf gleich ein entsprechender Thunk des DLL-Modulverwaltungsblocks angesprungen wird. Zum anderen wird dieser DLL-Thunk aufgerufen. Dieser DLL-Thunk hat nur eine Vermittlerfunktion: er macht nichts anderes als einen Sprung zu der Adresse, in der sich der erste Befehl der DLL-Funktion befindet. Die DLL-Thunks müssen nach dem Aus- und Wiedereinladen der DLL in den Hauptspeicher jeweils den neuen Verhältnissen angepaßt werden. [Hon92]

2. Die Anwendung lädt explizit die DLL zur Laufzeit und ermittelt und verwaltet die Einsprungspunkte in die DLL selbst. Dazu wird die DLL mit Hilfe der Windows-Funktion *LoadLibrary* geladen; LoadLibrary gibt einen Handle auf die DLL zurück. Die Einsprungspunkte können dann mit *GetProcAddress()* ermittelt werden. Ein Aufruf für eine Funktion „foo"sieht beispielsweise so aus:
   $GetProcAddress(hLib, pfFunctionList[i], „foo");$
   Die Einsprungspunkte einer DLL werden also jeweils in einem Array verwaltet. [Gei95]
   Dies hat den Vorteil, daß mehrere DLLs mit den gleichen Funktionsnamen gleichzeitig geladen und verwaltet werden können (vgl. Anhang B).

Anhang D

Glossar

CMIP: (Common Management Information Protocol) Protokoll für den Transfer von Managementinformationen auf OSI-Netzwerken.

CMOT (Common Management Information Services and Protocol over TCP) Implementierung des OSI-Managementprotokolls auf der Basis des TCP/IP-Protokollstacks.

DDE: (Dynamic Data Exchange) Kommunikationsprotokoll für den Datenaustausch zwischen Windows-Anwendungen.

DDL: (Dynamic Link Library) Programmbibliothek, deren Routinen erst zur Laufzeit des Programms eingebunden werden.

FTP: (File Transfer Protocol) Datenaustauschprotokoll über UDP.

Handle: Kennzahl, die unter Windows dazu dient, ein Objekt zu identifizieren. Die Verwendung von Handles erlaubt es dem Memory Manager von Windows Objekte nach Bedarf zu verschieben; der Zugriff auf das Objekt geschieht unabhängig von seiner Lage im Speicher immer mit dem selben Handle.

ICMP: (Internet Control Message Protocol) Protokoll für den Austausch von Fehlermeldungen und anderen Steuerinformationen zwischen Gateways und Hosts; die Übertragung der ICMP-Datagramme geschieht über IP.

NCP: (NetWare Core Protocol) proprietäres Serviceprotokoll auf NetWare Servern.

NetBEUI: (NetBIOS Extended User Interface) nicht routingfähiges Transportprotokoll unter NetBIOS, das verbindungslose wie auch verbindungsorientierte Dienste unterstützt.

NetBIOS: von IBM entwickelte LAN-Kommunikationsschnittstelle, die sowohl verbindungslose wie auch verbindungsorientierte Dienste unterstützt.

NLM: (NetWare Loadable Modules) Programm-Module, die vom NetWare Betriebssystem-Kernel (ab Version 3.11) zur Laufzeit nach Bedarf geladen werden können; die NLMs werden dadurch integrierter Bestandteil des Betriebssystems und erweitern damit dessen Funktionalität.

Ping: (Packet Internet Groper) Programm, das ICMP-Echoanforderungen an einen Host sendet und auf dessen Erwiderung wartet.

RAS: (Remote Access Service) von Windows NT angebotener Dienst für den Fernzugriff auf Netzwerke. Der RAS-Client wird dabei über den RAS-Server in das betreffende Netzwerk integriert; der Dienst selbst kann über Modem, X.25 oder ISDN realisiert werden.

RIP: (Router Information Protocol) proprietäres Protokoll auf Novell NetWare Servern zum Austausch von Routing Informationen; jeder RIP-Router sendet alle 60 s ein RIP-Paket mit der Liste aller Segmente, an die er angeschlossen ist.

SAP: (Service Advertising Protocol) proprietäres Protokoll auf Novell NetWare Servern für den Austausch von Informationen über die von ihnen angebotenen Dienste. Jeder SAP-Server trägt in die Binderies aller verfügbaren File-Server seinen Namen, Servicetyp, Netzwerkadresse, Nodeadresse über SAP ein und unterstützt so mögliche Clients bei der Suche nach Servern und ihren Diensten; darüberhinaus werden von SAP auch Broadcast-Pakete zum Auffinden von Services unterstützt. SAP wird außerdem für die periodische Identifizierung von Servern im Netz verwendet: jeder Server muß sich alle 60 s mit einem Server Identifikationspaket im Netzwerk melden. Namen und Adressen der Server, die sich so gemeldet haben, werden in den Binaries der anderen Server abgespeichert.

SNA: (Systems Network Architektur) von IBM 1974 entwickeltes Architekturmodell.

SNA LU 6.2: im Rahmen von SNA implementiertes Protokoll für die Kommunikation zwischen Prozessen.

TCP: (Transmission Control Protocol) verbindungsorientiertes Protokoll für die Vollduplex-Übertragung von Daten.

TCP/IP: bezeichnet eine Familie von Protokollen, die vor allem während der Entwicklung des ARPANET entstanden sind. Zur Protokollfamilie gehören neben TCP und IP auch ICMP, UDP und Anwendungsprotokolle wie z.B. SNMP, FTP, TFTP, TELNET.

TELNET: auf TCP basierendes Protokoll, das dem Benutzer erlaubt, sich auf einem entfernten System im Netzwerk einzuloggen (remote login).

TFTP: (Trivial File Transfer Protocol) einfaches Datenaustauschprotokoll über UDP; TFTP implementiert im Gegensatz zu FTP keine Berechtigungsüberprüfung, keine Anzeige der Verzeichnisinhalte etc.

UDP: (User Datagram Protocol) im Gegensatz zu TCP verbindungsloses Transportprotokoll.

Literaturverzeichnis

[BRO87] Brockhaus Enzyklopädie in 24 Bänden Bd. 2, 1987. ISBN 3-7653-1102-2.

[DMT94] DMTF. *PC Systems Standard Groups Definition*, 1994. Release Version 1.0.

[DMT95a] DMTF. *Desktop Management Interface Reference*, 1995. Revision 1.1.

[DMT95b] DMTF. *DMTF Server MIF Definition*, 1995. Draft Version 0.3.

[Gei95] Kyle Geiger. *Inside ODBC.* Microsoft Press Deutschland, Unterschleißheim, 1995. ISBN 3-86063-359-7.

[Heg93] S. Abeck; H.-G. Hegering. *Integriertes Netz- und Systemmanagement.* Addison-Wesley, Bonn, 1993. ISBN 3-89319-508-4.

[Hei94] David Griffiths; Mathias Hein. *SNMP Simple Network Management Protocol Version 2.* Thomson Publishing, Bonn, 1994. ISBN 3-929821-51-6.

[Heu95] Saake Gunter; Andreas Heuer. *Datenbanken: Konzepte und Sprachen.* Thomson Publishing, Bonn, 1995. ISBN 3-929821-31-1.

[Hon92] Peter Wilken; Dirk Honekamp. *Windows 3.1.* Addison-Wesley, Bonn, 1992. ISBN 3-89319-463-0.

[HP93] Hewlett Packard, Santa Clara, California, USA. *HP OpenView for Windows Programmer's Reference Manual*, 1993.

[HP94] Hewlett Packard, Santa Clara, California, USA. *HP OpenView for Windows Workgroup Node Manager User's Guide*, 1994.

[HP95a] Hewlett Packard, Santa Clara, California, USA. *HP OpenView for Windows Version 7.2 Programmer's Guide*, 1995.

[HP95b] Hewlett Packard, Santa Clara, California, USA. *HP OpenView for Windows Workgroup Node Manager Technical Evaluation Guide*, 1995.

[Hun95] Craig Hunt. *TCP/IP Netzwerk Administration.* O'Reilly International Thomson Verlag, Bonn, 1995. ISBN 3-930673-02-9.

[Jan93] Wolfgang Schott; Rainer Janssen. *SNMP - Konzepte, Verfahren, Plattformen.* DATACOM-Verlag, Bergheim, 1993. ISBN 3-89238-077-5.

[Kar94] Turgay Karslioglu. *Novell NetWare 4.01 Praxisbuch.* IWT Verlag, Vaterstetten b. München, 1994. ISBN 3-88322-453-7.

[Kru94] David J. Kruglinski. *Inside Visual C++ (Version 1.5).* Microsoft Press Deutschland, Unterschleißheim, 1994. ISBN 3-86063-342-2.

[Mar94] Jörg Fritze; Jürgen Marsch. *SQL - Eine praxisorientierte Einführung.* Vieweg, Braunschweig, 1994. ISBN 3-528-15210-9.

[Mor94] Matthias Oberdorfer; Ralf Morgenstern. *Visual C++ 1.5.* Addison Wesley, Bonn, 1994. ISBN 3-89319-760-5.

[Nat94] Bob Natale. *Windows SNMP - An Open Interface for Programming Network Management Applications using the Simple Network Management Protocol under Microsoft Windows WinSNMP/Manager API Version 1.1.* American Computer and Electronics Corporation, Gaithersburg, Juni 1994.

[Nor93] Paul Yao; Peter Norton. *Borland C++ Programmierung unter Windows.* Wolfram's Verlag, Attenkirchen, 1993. ISBN 3-86033-119-1.

[Nov92a] Novell, San Jose, California, USA. *Binary format for SNMP MIBs Programmer's Reference Guide*, Oktober 1992.

[Nov92b] Novell, San Jose, California, USA. *GUI Tools Programmer's Reference Guide*, Dezember 1992.

[Nov93a] Novell, San Jose, California, USA. *Alarm Manager Programmer's Reference Guide*, Juni 1993.

[Nov93b] Novell, San Jose, California, USA. *Application Integrator Programmer's Reference Guide*, Juni 1993.

[Nov93c] Novell, San Jose, California, USA. *NetWare Management System Database Schema and Application Programming Interface*, Juni 1993.

[Nov93d] Novell, San Jose, California, USA. *NetWare Management System Software: Getting Started*, Juni 1993.

[Nov93e] Novell, San Jose, California, USA. *Overview of the NetWare Management System Software Development Kit*, Juni 1993.

[Nov93f] Novell, San Jose, California, USA. *SNMP Data Server Programmer's Reference Guide Version1.0*, Juni 1993.

[Pet90] Charles Petzold. *Programmierung unter Windows*. Microsoft Press, München, 1990. ISBN 3-86063-312-0.

[RFC90a] J.D. Case; M. Fedor; M.L. Schoffstall; C. Davin. *RFC 1157: Simple Network Management Protocol (SNMP)*, 1990.

[RFC90b] M.T. Rose; K. McCloghrie. *RFC 1155: Structure and identification of management information for TCP/IP-based internets*, 1990.

[RFC91a] M. Rose; K. McCloghrie. *RFC 1212: Concise MIB Definition*, 1991.

[RFC91b] M.T. Rose. *RFC 1215: Convention for Defining Traps for Use with the SNMP*, 1991.

[RFC91c] M.T. Rose; K. McCloghrie. *RFC 1213: Management Information Base for network management of TCP/IP-based internets:MIBII*, 1991.

[RFC93a] M.T. Rose; J. Case; K. McCloghrie. *RFC 1452: Coexistence between version1 und version2 of the Internet-standard Network Management Framework*, 1993.

[RFC93b] M.T. Rose; J.Case; K. McCloghrie. *RFC 1449: Transport Mappings for version 2 of the Simple Network Management Protocol (SNMPv2)*, 1993.

[Ric94] Jeffrey M. Richter. *MS Windows NT - Weiterführende Programmierung*. Microsoft Press, Unterschleißheim, 1994. ISBN 3-86063-328-7.

[Ros93] Marshall T. Rose. *Einführung in die Verwaltung von TCP/IP Netzen*. Verlage Carl Hanser und Prentice Hall, Wien, London, 1993. ISBN 3-446-16444-8 (Hanser).

[Sch92] Christoph Scheid. *Novell NetWare Btrieve*. Addison Wesley, Bonn, 1992. ISBN 3-89319-404-5.

[SMS95a] Microsoft. *MS Systems Management Server Administrators Guide (CD-ROM)*, 1995.

[SMS95b] Microsoft. *MS Systems Management Server Reviewers Guide (CD-ROM)*, 1995.

[SMS95c] Microsoft. *SMS SDK Online Help from Back Office SDK (CD-ROM)*, 1995.

[SNI95] SNI, Augsburg. *Lösungstudie für das Projekt „Server Management "*, 1995.

[Sta94] Reiner Standke. *Windows NT und Windows NT Advanced Server.* IWT Verlag, Vaterstetten, 1994. ISBN 3-88322-478-2.

[Vos94] Gottfried Vossen. *Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme.* Addison Wesley, Bonn, 1994. ISBN 3-89319-566-1.

[Zen93] Andreas Zenk. *Lokale Netze - Kommunikationsplattform der 90er Jahre.* Addison Wesley, Bonn, 1993. ISBN 3-89319-567-X.