INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwurf und Implementierung eines CMIP/SNMP Gateways

Bearbeiter:	Michael Langer
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Alexander Keller Dr. Bernhard Neumair

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwurf und Implementierung eines CMIP/SNMP Gateways

Bearbeiter:	Michael Langer
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Alexander Keller Dr. Bernhard Neumair
Abgabetermin:	15. November 1996

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbstständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. November 1996

. . . .
(Unterschrift des Kandidaten)

Zusammenfassung

Die OSI-Managementarchitektur ist wegen ihrer mächtigen Funktionalität besonders für das Netz- und Systemmanagement von großen und heterogenen Kommunikationsnetzen geeignet. Allerdings wird sie wegen ihrer Komplexität und damit teuren Implementierung von sehr wenigen Managementwerkzeugen unterstützt. Die Internet-Managementarchitektur spielt hingegen als de-facto-Standard im Bereich des Managements von lokalen Rechnernetzen eine wichtige Rolle. Aufgrund ihrer Einfachheit gilt sie vielfach als Übergangslösung, denn langfristig sollen die OSI-Normen unterstützt werden. Es werden also diese beiden Managementarchitekturen noch einige Zeit nebeneinander existieren. Die Interoperabilität zwischen diesen und deren Integration werden aufgrund der immer stärker werdenden Forderung nach einheitlichem Management zu entscheidenden Voraussetzungen.

Um diese Migration zu erleichtern, wurde in dieser Diplomarbeit ein CMIP/SNMP Gateway entworfen und prototypisch implementiert. Dieses Managementgateway integriert das Internet-Management in die OSI-Managementarchitektur. Es stellt somit einem OSI-Manager den Zugriff auf SNMP-Ressourcen zur Verfügung, indem es die SNMP-Managementinformationen auf OSI-Managementobjekte abbildet. Dazu wurden zuerst verschiedene Möglichkeiten für die Abbildung sowohl der Informations- als auch der Kommunikationsmodelle der beiden Architekturen aufeinander diskutiert. Anschließend wurde diese Top-Down Sichtweise auf ein Managementgateway mit der Entwicklungsumgebung *IBM TMN WorkBench for AIX* zusammengeführt. Das heißt, die vorgestellten Informations- und Kommunikationsmodellabbildungen wurden auf die Agentenarchitektur der *IBM TMN WorkBench for AIX* projiziert. Durch den Einsatz dieser Entwicklungsumgebung traten Einschränkungen auf, die die schließlich prototypisch implementierte Architektur des CMIP/SNMP Gateways maßgeblich beinflußten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Aufgrund der zunehmenden Größe und Heterogenität von Rechnernetzen gewinnt die Aufgabe des Netz- und Systemmanagements immer mehr an Bedeutung. Ein wichtiger Schritt, um dieser steigenden Komplexität auch in Zukunft gewachsen zu sein, ist die Realisierung von offenem und einheitlichem Management.
Dazu existieren bereits eine Vielzahl an unterschiedlichen, standardisierten Managementarchitekturen: Das OSI-Management der ISO wird vor allem wegen seiner großen Mächtigkeit und Komplexität in Telekommunikationsnetzen eingesetzt, die Verbreitung des Internet-Managements des IAB/IETF betrifft hauptsächlich den Bereich (des Managements) der lokalen Rechnernetze. Die *Object Management Architecture* (OMA) der Object Management Group setzt ihren Schwerpunkt eher im Bereich des System- und Anwendungsmanagements. Weitere Managementarchitekturen sind zum Beispiel das TMN (Telecommunication Management Network) der ITU, OMNIPoint des *Network Management Forums* oder DME (Distributed Management Environment) der *Open Software Foundation*.

1.1 Motivation

Eine sehr bedeutende Architektur stellt das OSI-Management dar. Diese Normen werden allerdings von der Mehrzahl der auf dem Markt verfügbaren Managementwerkzeuge wegen ihrer Komplexität und deshalb teuren Implementierung nicht unterstützt. SNMP hingegen spielt als de-facto-Standard heutzutage eine wichtige Rolle im Management von TCP/IP-Rechnernetzen. Es ist sehr viel einfacher, wenngleich auch weniger mächtig, als der OSI-Ansatz. Daher gilt SNMP vielfach als Übergangslösung, langfristig soll es vom OSI-Management abgelöst werden. Diese beiden Managementarchitekturen werden daher noch einige Zeit nebeneinander existieren. Die Interoperabilität zwischen diesen und deren Integration werden aufgrund der immer stärker werdenden Forderung nach einheitlichem Management zu entscheidenden Voraussetzungen. Ein wichtiger Schritt in diese Richtung ist die Schaffung einer Möglichkeit, SNMP-Ressourcen von einem OSI-

Manager aus überwachen und steuern zu können. Eine potentielle Möglichkeit für die Integration von Managementarchitekturen im allgemeinen und des Internet-Managements in die OSI-Managementarchitektur im speziellen stellen Managementgateways dar (siehe Abbildung 1.1).

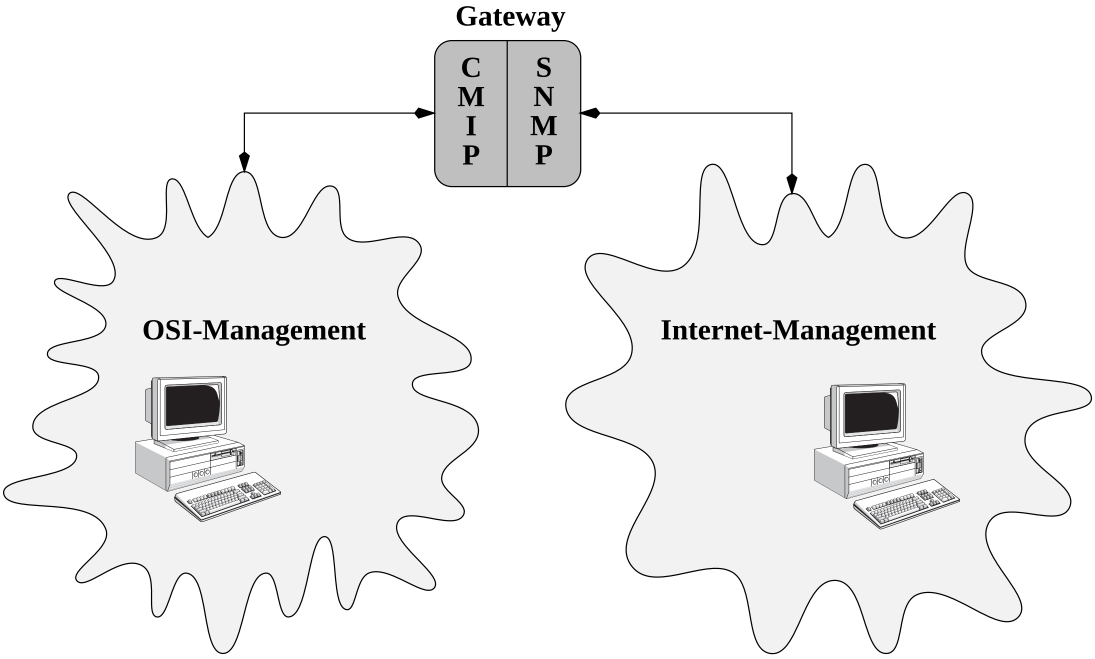


Abbildung 1.1: Integration von Managementarchitekturen

1.2 Aufgabenstellung

Durch die Integration des Internet-Managements in die OSI-Managementarchitektur ist es möglich, die komplexen und mächtigen Funktionen der integrierenden Architektur auf die Ressourcen der integrierten Architektur anzuwenden. Bei der Realisierung dieser Einbettung mittels eines Managementgateways werden keine Veränderungen in den Komponenten der beiden Managementarchitekturen notwendig und die Interoperabilität zwischen diesen kann transparent gestaltet werden. Damit wird die oben erwähnte, langfristige Migration von SNMP-basiertem Management hin zu objektorientiertem OSI-Management finanziell, technisch und verwaltungstechnisch in vertretbarem Aufwand möglich, da die bestehenden, „alten“ Systeme weiterverwendet werden können.
Das Ziel dieser Diplomarbeit besteht in der Realisierung eines Gateways zwischen den Managementprotokollen CMIP und SNMP. Dabei steht nicht die Protokollkonvertierung in Vordergrund, sondern vielmehr muß ein sinnvoller Übergang zwischen den Informationsmodellen, die CMIP und SNMP zugrunde liegen, gefunden

werden. So soll zuerst die Architektur eines CMIP/SNMP Gateways entworfen und anschließend diese prototypisch implementiert werden.
Die Entwicklungsumgebung *IBM TMN WorkBench for AIX* soll dabei maßgeblich Einfluß auf diese Architektur nehmen. Damit wird die Zielsetzung der Diplomarbeit dahingehend erweitert, daß eine Machbarkeitsstudie für den Entwurf und die Implementierung des CMIP/SNMP Gateways mit den Möglichkeiten und Einschränkungen dieser Entwicklungsumgebung erstellt werden soll.
Das CMIP/SNMP Gateway soll einem OSI-Manager eine dem OSI-Informationsmodell-konforme Sichtweise auf die zu überwachenden und zu steuernden SNMP-Ressourcen ermöglichen. Dazu muß folgende Funktionalität vom Gateway bereitgestellt werden:

1. Lesender und schreibender Zugriff auf SNMP-Ressourcen und damit auf relevante Managementinformationen. Die Realisierung des Zugriffs soll transparent erfolgen, das heißt, ein OSI-Manager kann und soll nicht erkennen, ob die Managementinformation von einem OSI-Agenten oder von einem SNMP-Agenten stammt.

2. SNMP-Traps sollen vom Gateway empfangen und auf OSI-Notifikationen abgebildet werden.

Die entgegengesetzte Richtung, die Integration des OSI-Managements in die Internet-Architektur mittels eines SNMP/CMIP Gateways ist nicht Teil der Aufgabenstellung. Dieses Gateway müßte eine SNMP-Sichtweise auf OSI-Ressourcen bereitstellen. Dies ist aber wenig sinnvoll, denn die mächtigen Funktionalitäten der zu überwachenden OSI-Agenten können nicht von den SNMP-Managern genutzt werden, da die Konzepte der Internet-Managementarchitektur dafür nicht ausreichend sind.

1.3 Gliederung

In dieser Diplomarbeit werden zu Beginn verschiedene relevante Managementarchitekturen (CMIP, SNMP und TMN), Integrationsansätze für Managementarchitekturen und zwei Internet-MIBs vorgestellt (Kapitel 2).
Anschließend befaßt sich Kapitel 3 sowohl mit den schon vorhandenen Arbeiten der IIMC (*ISO-Internet Management Coexistence*), als auch mit einer Einführung in die *IBM TMN WorkBench for AIX*. Mit dieser Entwicklungsumgebung wurde das Gateway implementiert. Dieser Abschnitt soll auch schon Hinweise darauf geben, welche Möglichkeiten oder Einschränkungen sich beim Einsatz der *IBM TMN WorkBench for AIX* ergeben.
Danach werden die grundlegenden Ansätze und Konzepte von Managementgateways beschrieben. Es wird aufgezeigt, welche Problemstellungen bei der Integration des Internet-Managements in die OSI-Managementarchitektur auftreten und

dafür jeweils verschiedene Lösungsmöglichkeiten diskutiert. Damit ergibt sich eine Top-Down Sichtweise auf die unterschiedlichen Architekturen eines CMIP/SNMP Gateways (Kapitel 4).
Im zentralen Kapitel 5 erfolgt die Zusammenführung der bisher erarbeiteten Sachverhalte (siehe Abbildung 1.2): Die Top-Down Sichtweise aus Kapitel 4 wird mit

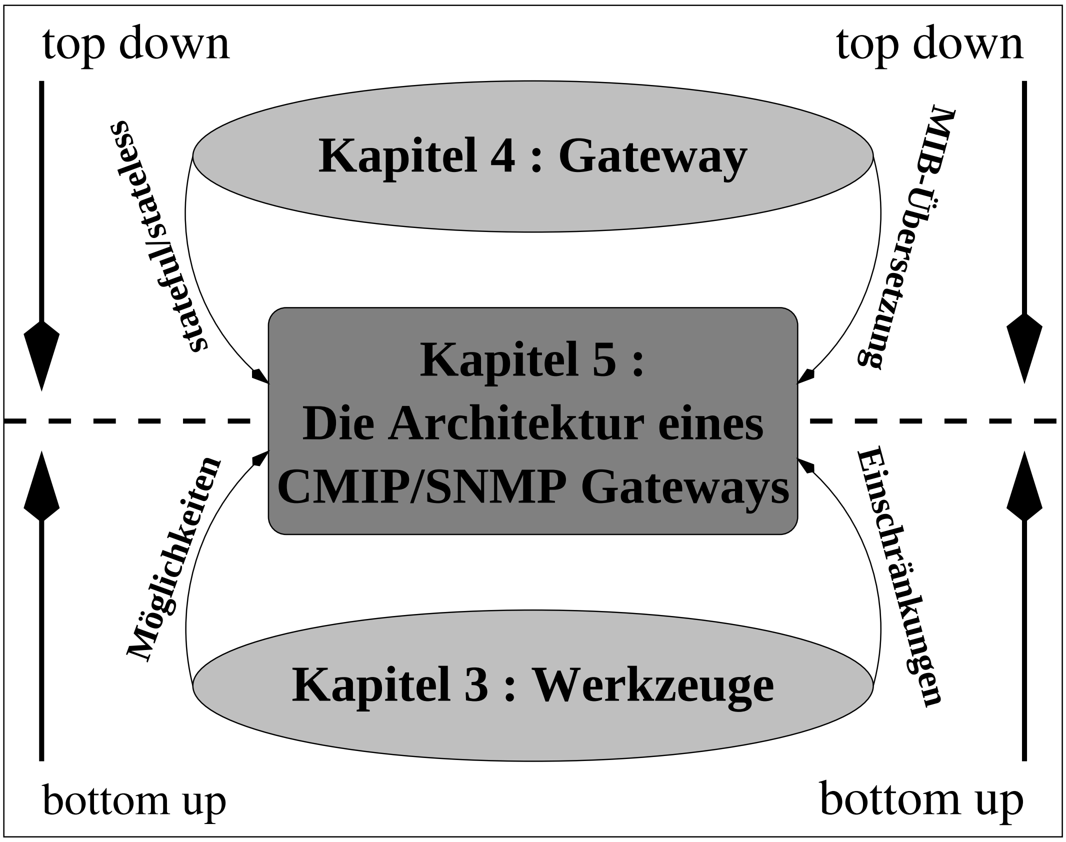


Abbildung 1.2: Zusammenführung der möglichen Sichtweisen

der Bottom-Up Analyse, die entsteht, wenn die Einschränkungen/Möglichkeiten aus Kapitel 3 auf die Realisierung eines Gateways angewendet werden, verbunden. Das Ergebnis ist schließlich die Architektur eines CMIP/SNMP Gateways, die in Kapitel 6 prototypisch implementiert wird.
Die Ergebnisse der Diplomarbeit werden in Kapitel 7 zusammengefaßt.

Kapitel 2

Grundlagen

Für den Entwurf und die Realisierung eines CMIP/SNMP Gateways werden verschiedene Grundlagen benötigt. So sollen im ersten Abschnitt dieses Kapitels die Konzepte aufgezeigt werden, auf denen sowohl die OSI- als auch die Internet-Managementarchitektur basieren. Diese Architekturen werden jeweils in vier Modelle[1] (Organisations-, Informations-, Kommunikations- und Funktionsmodell) aufgeteilt, um damit deren unterschiedliche Aspekte des Managements zu beschreiben. Anschließend soll die TMN-Managementarchitektur vorgestellt werden.
Der zweite Abschnitt befaßt sich mit verschiedenen Ansätzen, die der Integration von Managementarchitekturen zugrunde liegen können.
Zuletzt werden noch zwei Internet-MIBs betrachtet, die für die Implementierung des CMIP/SNMP Gateways relevant sind.

2.1 Managementarchitekturen

2.1.1 Internet-Managementarchitektur

Die Internet-Managementarchitektur des IAB (Internet Activity Board) und der IETF (Internet Engineering Task Force) hat sich im Bereich des Managements von lokalen Rechnernetzen durchgesetzt (siehe zum Beispiel [HeAb93], [Stal93], [Black92] oder [HeNeWi95]). Sie zeichnet sich vor allem dadurch aus, daß die Devise, die bereits vom IAB für die Kommunikationsprotokollfamilie TCP/IP herausgegeben wurde, übernommen wurde: einfache und implementierungsfreundliche Spezifikationen. Der Kern dieser Architektur (*SNMPv1*[2]) setzt sich im wesentlichen aus drei Dokumenten zusammen, die das sogenannte *Internet Network Management Framework* bilden:

[1]Nach [HeAb93] muß jede Managementarchitektur, die für ein integriertes Management in heterogener Umgebung geeignet sein soll, diese Modelle bereitstellen.

[2]Diese Diplomarbeit setzt sich nur mit SNMPv1 auseinander, da der später vorgestellte MIB-Compiler ausschließlich SNMPv1-MIBs bearbeitet.

1. *Structure and Identification of Management Information for TCP/IP-Based Internets*, RFC 1155, May 1990
2. *A Simple Network Management Protocol*, RFC 1157, May 1990
3. *Management Information Base for Network Management of TCP/IP-Based Internets : MIB-2*, RFC 1213, März 1991

Organisationsmodell

Der Internet-Managementarchitektur liegt ein hierarchisches Organisationsmodell zugrunde. Es wird ein „Manager" und ein „Agent" definiert, die über das Managementprotokoll „SNMP" miteinander kommunizieren (siehe Abbildung 2.1). In dieses Konzept werden auch „Proxy-Agenten" integriert, das heißt, es können

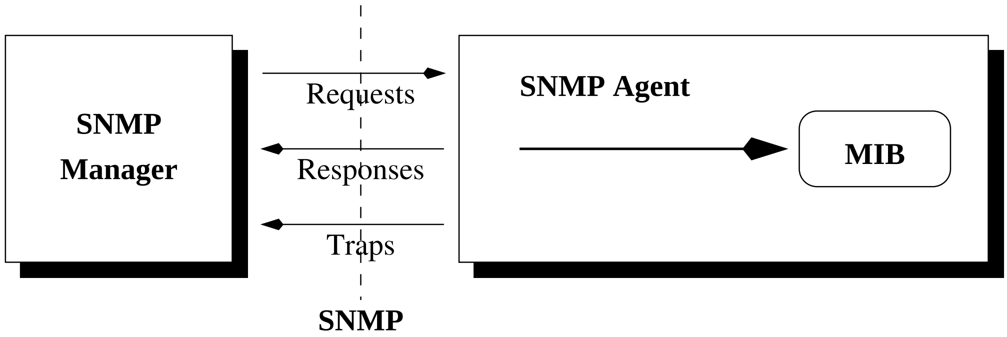


Abbildung 2.1: Internet Manager-Agent Beziehung

auch Ressourcen (Geräte, Komponenten) überwacht und gesteuert werden, die das SNMP-Protokoll nicht unterstützen. In der Version 2 von SNMP wird weiterhin eine Kommunikation zwischen zwei Managern angeboten, wodurch neue Hierarchiebildungen ermöglicht werden.

Informationsmodell

Das Internet-Informationsmodell ist weit weniger mächtig, als zum Beispiel das der im nächsten Abschnitt vorgestellten OSI-Architektur. Es werden zwar Managementobjekte definiert, diesen liegen aber keine objektorientierten Ansätze zugrunde. Nach [HeNeWi95] basiert das Internet-Informationsmodell auf einem „objektbasierten Ansatz". Bei diesen Managementobjekten können es sich entweder um Variablen oder Tabellen handeln. Die Managementobjekte werden in Form eines Baumes (Internet-Registrierungsbaum) angeordnet. Dieser dient sowohl zur Registrierung der zu überwachenden Ressourcen als auch zur Gruppierung von inhaltlich zusammengehörigen Objekten. Schließlich erfolgt die Namensgebung eines Managementobjekts anhand eines Pfades von der Wurzel des Registrierungsbaums hin zu dem Objekt selbst (OID, Object Identifier). Tabellen und Gruppen

werden im Registrierungsbaum in den Knoten, Variablen dagegen in den Blättern dargestellt.
Die Definition von Managementobjekten erfolgt in einer Template-Sprache, die auf ASN.1 basiert.
Die Information, die ein SNMP-Agent bereitstellt, wird als *Management Information Base (MIB)* bezeichnet. Dabei handelt es sich nach [HeAb93] um einen „konzeptionellen Behälter“, also um eine Art von Schnittstellendefinition zu den zu überwachenden Ressourcen. Es existieren sehr viele verschiedene MIBs, in Abschnitt 2.3 werden zwei, die für diese Diplomarbeit relevant sind, vorgestellt.
Der einfache Ansatz des Internet-Informationsmodells beinhaltet entscheidende Nachteile: Unter anderem können durch die fehlenden objektorientierten Vererbungs- und Strukturierungsmöglichkeiten zum einen schon definierte Managementobjekte nicht in einer anderen MIB wiederverwendet werden, zum anderen ist damit die Menge an schon definierten Informationen unübersehbar.

Kommunikationsmodell

Das Kommunikationsmodell des Internet-Managements basiert auf dem *Simple Network Management Protocol (SNMP)*. SNMP setzt auf dem verbindungslosen Transportprotokoll *User Datagram Protocol (UDP)* auf und definiert vier verschiedene Protokollinteraktionen:

1. *GetRequest-PDU*: Ein SNMP-Manager kann diese PDU erzeugen, um damit Variableninhalte aus der MIB eines SNMP-Agenten zu lesen. Die Ergebnisse dieser Anfrage werden vom Agenten in einer *GetResponse-PDU* zum Manager zurückgegeben.

2. *GetNextRequest-PDU*: Diese PDU ermöglicht es einem SNMP-Manager, die MIB eines SNMP-Agenten zu durchlaufen. Dazu gibt er in einer *GetNext-Request-PDU* die OID eines Objekttyps oder einer Objektinstanz an. In der zu diesem Request vom SNMP-Agenten zurückgeschickten *GetResponse-PDU* ist die OID und der Wert der in der lexikographischen Ordnung nächsten Objektinstanz enthalten. Durch wiederholtes Auslösen der *Get-NextRequest-PDU* kann zum Beispiel somit eine komplette SNMP-Tabelle zu einem Manager übertragen werden.

3. *SetRequest-PDU*: Ein SNMP-Manager kann durch das Auslösen dieser PDU Objektinstanzen (Variablen) mit neuen Werten im SNMP-Agenten belegen. Das Ergebnis über den Erfolg der Veränderung wird vom SNMP-Agenten mittels einer *GetResponse-PDU* zum Manager zurückgeschickt. Damit werden auch steuernde Eingriffe ermöglicht.

4. *Trap-PDU*: Ein SNMP-Agent kann wichtige interne Ereignisse ohne explizite Aufforderung durch einen SNMP-Manager mittels einer Meldung (Trap) übertragen.

SNMP stellt für die Zugriffsberechtigung auf SNMP-Agenten ein Sicherheitskonzept bereit: In jeder SNMP-PDU wird ein sogenannter *Community String* übergeben. Dieser *Community String* erfüllt die Aufgabe eines Paßworts für einen MIB-Zugriff. Ein SNMP-Agent bearbeitet nur diejenigen Anforderungen, deren *Community Strings* mit den konfigurierten Strings des Agenten übereinstimmen. Durch das unverschlüsselte Übertragen der *Community Strings* gilt das Sicherheitskonzept als ungenügend.

Eine allgemeine Verbesserung verspricht die Version 2 von SNMP. Sie bietet untern anderem eine Weiterentwicklung des *GetNextRequests* (*GetBulkRequest*) und eine Integration von asynchronen Ereignismeldungen in das Informationsmodell an. Das in den RFCs 14XX erweiterte Sicherheitskonzept (Authentifizierung, Verschlüsselung) setzte sich nicht durch und so wurde in den RFCs 19XX wieder auf den Community-basierten Ansatz zurückgegriffen.

Funktionsmodell

In der Internet-Managementarchitektur existiert kein Funktionsmodell. Aufgrund der Forderung nach einfachen Agenten und damit einer kleinen Menge an Funktionalität im SNMP-Agenten, werden die komplexen Managementaufgaben zu den SNMP-Managern verlagert. Als einen ersten Ansatz für das Funktionsmodell des Internet-Managements kann die RMON-MIB (Remote Network Monitoring Management Information Base, RFC 1757) gesehen werden.

Distributed Protocol Interface (DPI)

In [HeNeWi95] werden mit der Verlagerung der Schwerpunkte des integrierten Managements vom Netzmanagement in Richtung System- und Anwendungsmanagement offene Schnittstellen innerhalb eines Agentensystems motiviert.
So laufen zum Beispiel auf einem Host verschiedene Anwendungen und einige dieser Anwendungen stellen eine eigene MIB für das Management bereit. Diese Anwendungen mit ihren MIBs sowie das benötigte Betriebssystem stammen sicherlich von einer Vielzahl von unterschiedlichen Herstellern. Ein Agentensystem muß also für alle diejenigen Anwendungen auf einem Host, die ein MIB-Modul bereitstellen, sowohl dieses MIB-Modul als auch ein Protokollmodul, für die Kommunikation mit Managersystemen, zusammenbinden. Dafür wird eine flexible, dynamische und herstellerunabhängige Schnittstelle im Agentensystem benötigt, die nach [HeNeWi95] unter anderem folgende Aufgaben erfüllen soll (siehe auch Abbildung 2.2, ebenfalls nach [HeNeWi95]):

- Registrierung der MIB-Moduln beim Protokollmodul (zusätzlich muß angegeben werden, für welche Ressourcen das jeweilige Modul zuständig ist).

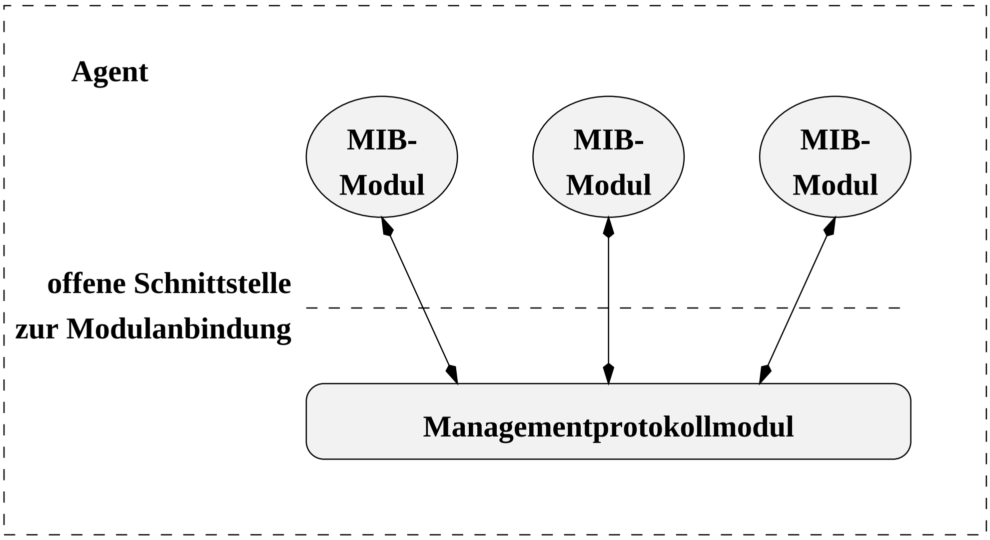


Abbildung 2.2: Offene Schnittstelle in Agentensystemen

- Das Protokollmodul übergibt die empfangenen Anforderungen an die entsprechenden MIB-Moduln weiter. Die Ergebnisse der MIB-Moduln werden vom Protokollmodul entgegengenommen und zu den entsprechenden Managern zurückgeschickt.

- Ebenso müssen Ereignismeldungen von den MIB-Moduln vom Protokollmodul zu Managersystemen weitergeleitet werden.

Ein Beispiel für eine derartige Schnittstelle ist *SNMP DPI* (Distributed Protocol Interface). Sie bietet eine offene Schnittstelle innerhalb eines Agentensystems speziell für die Internet-Managementarchitektur. Verschiedene MIB-Moduln (Subagenten) können sich nun über DPI bei dem SNMP-Protokollmodul (Hauptagent) flexibel und dynamisch an- bzw. abmelden. Dabei registrieren sie einen bestimmten Teilbaum des Internet-Registrierungsbaums. Alle SNMP-Requests an ein Managementobjekt in einem Teilbaum werden vom Hauptagenten zu dem Subagenten weitergeleitet, der für diesen Teilbaum registriert wurde. DPI bildet dazu alle möglichen SNMP-Anforderungen auf eigene Operationen ab. Somit wird SNMP-Funktionalität auf DPI verlagert.
Die Vorstellung von DPI im Zusammenhang mit dieser Diplomarbeit beruht darauf, daß der in 2.3.2 vorgestellte *LRZ Systemagent* einen Subagenten darstellt, der über DPI an einen Hauptagenten gekoppelt werden kann. Der verwendete Hauptagent beinhaltet zusätzlich zu dem SNMP-Protokollmodul noch ein MIB-Modul für die in 2.3.1 vorgestellte *MIB-2*. Somit stand für diese Diplomarbeit ein Agentensystem zur Verfügung, das sowohl die *MIB-2* als auch die *LRZ Systemagenten MIB* bereitstellt.

2.1.2 OSI-Managementarchitektur

Von der ISO und der ITU (ehemals CCITT) wurde eine Architektur für das Management von „offenen“ Systemen, das heißt von Systemen, die konform zu dem OSI-Schichtenmodell sind, festgelegt (siehe zum Beispiel [HeAb93], [Stal93], [Black92] oder [HeNeWi95]). Die hauptsächliche Verbreitung dieser Managementarchitektur befindet sich wegen ihrer großen Komplexität und damit schwieriger und vor allem teurer Implementierungen nicht in lokalen Rechnernetzen, sondern in Telekommunikationsnetzen. Der *OSI-Management Framework* dient als Rahmenwerk und wird von einer Reihe von weiteren Dokumenten verfeinert. Dabei wird die OSI-Managementarchitektur in vier Modelle aufgeteilt, wobei jedes einen anderen Aspekt des Managements beschreibt:

Organisationsmodell

Ebenfalls wie beim Internet-Management liegt der OSI-Managementarchitektur ein hierarchisches Orgranisationsmodell zugrunde. Es werden zwei Rollen definiert, eine „Manager“- und eine „Agent“-Rolle. Im Unterschied zum Internet-Management kann aber ein OSI-System gleichzeitig verschiedene Rollen annehmen (siehe Abbildung 2.3 aus [HeAb93]). Die Zusammenarbeit zwischen Manager

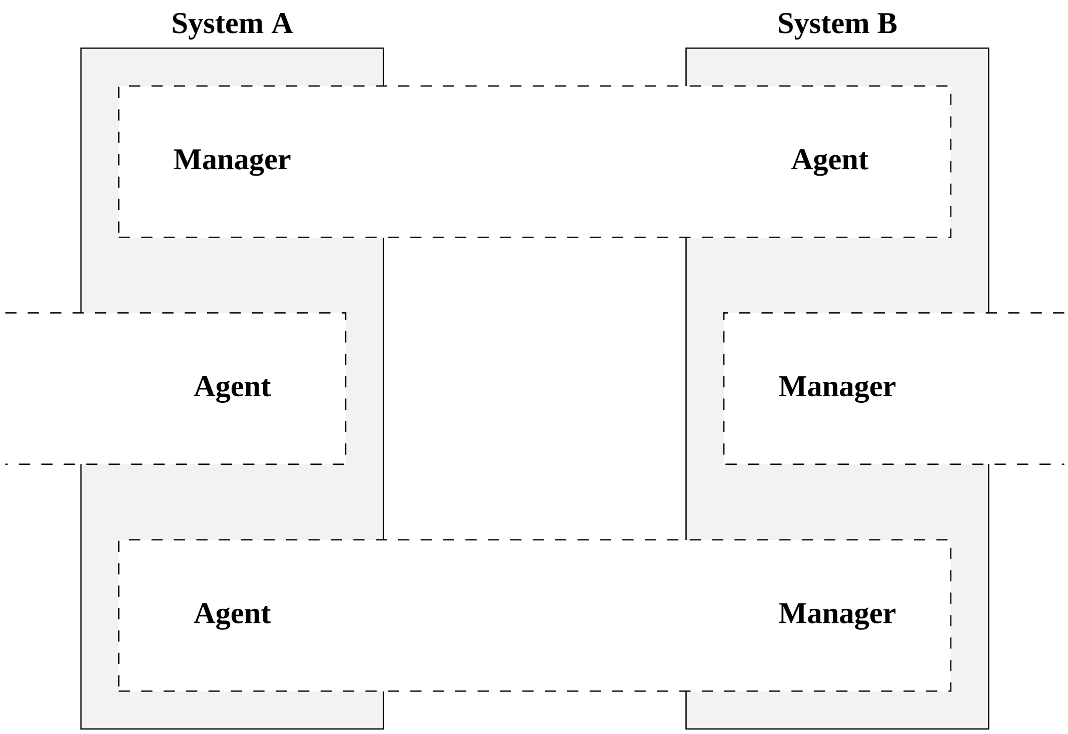


Abbildung 2.3: Die OSI-Managementrollen

und Agent zeigt Abbildung 2.4: Ein Manager kann über das CMIP-Protokoll An-

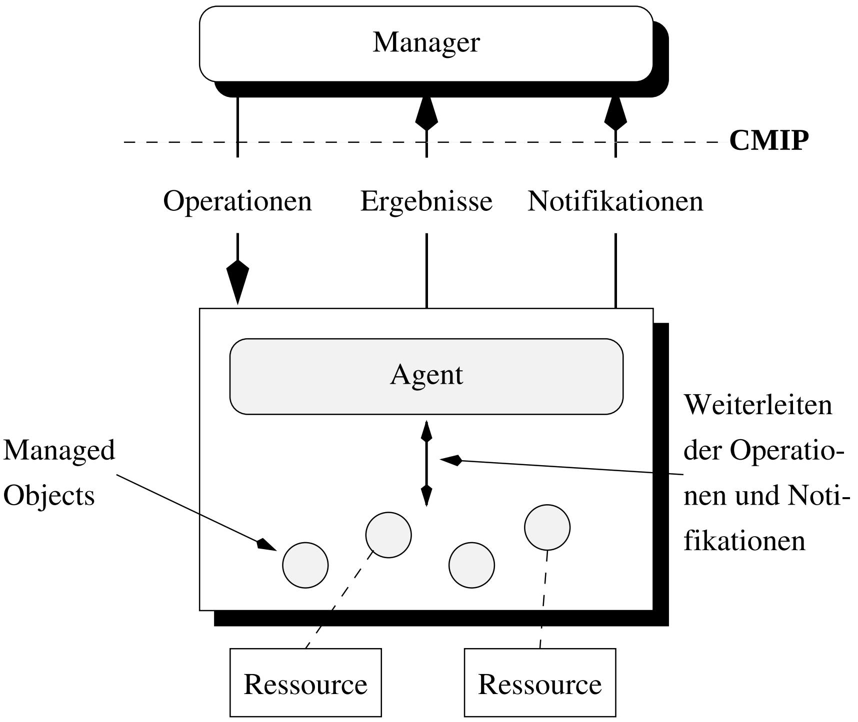


Abbildung 2.4: OSI Manager-Agent Beziehung

forderungen (Operationen) an einen Agenten stellen. Dieser nimmt die Requests entgegen, bearbeitet sie und schickt die Ergebnisse an den Manager zurück. Wichtige interne Ereignisse im Agenten können unaufgefordert verschiedenen Managern gemeldet werden (Notifikationen).
Weiterhin wird ein Domänenkonzept festgelegt (siehe zum Beispiel [HeAb93]).

Informationsmodell

Das OSI-Informationsmodell wird im Dokument *Structure of Management Information* (SMI, ISO10165) beschrieben. Es handelt sich dabei um einen objektorientierten Ansatz, das bedeutet die Verwendung von Datenabstraktion, Vererbung, Containment und Polymorphie. Managementobjekte (Managed Objects, MO) sind die Abstraktion von realen Ressourcen. Wie in Abbildung 2.5 dargestellt wird, kann mit Hilfe von Managementoperationen auf das Managementobjekt selbst oder auf enthaltene Attribute zugegriffen werden. Wichtige interne Ereignisse in einem Managementobjekt können mittels Notifikationen weitergeleitet werden. Die Attribute eines Managementobjekts haben zwei Aufgaben: erstens die Manipulation eines Managementobjekts zu ermöglichen und zweitens die Eigenschaften (in Abhängigkeit von der realen Ressource) eines Managementobjekts zu beschreiben. Für jedes Managementobjekt existiert weiterhin eine informelle Verhaltensdefinition (Behaviour). Die Manipulation von Managementobjekten

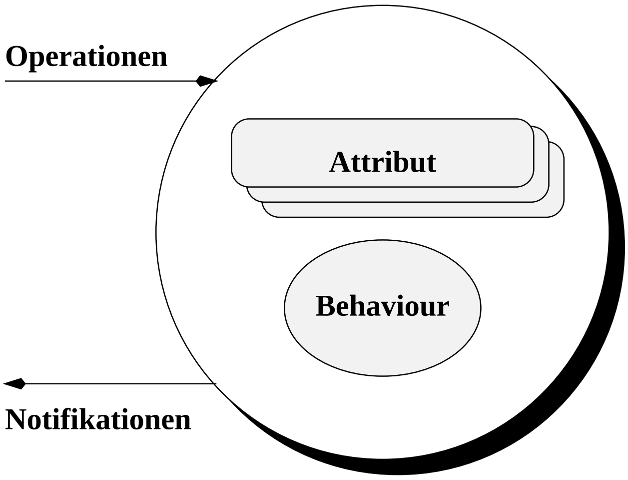


Abbildung 2.5: OSI Managementobjekt

und damit die möglichen Managementoperationen auf diesen kann in zwei Kategorien eingeteilt werden:

1. Operationen auf Managementobjekten:
   Der Wirkungsbereich betrifft das ganze Managementobjekt: Kreieren bzw. Löschen von Managementobjekten oder das Auslösen einer Aktion auf einem Managementobjekt.

2. Operationen auf Attributen:
   Diese betreffen nur die Attribute oder Attributgruppen: get, set, add, remove, replace.

Da dem OSI-Informationsmodell ein objektorientierter Ansatz zugrundeliegt, werden Managementobjekte als Instanziierungen von Managementobjekt-Klassen (Managed Object Class, MOC) begriffen. Eine Managementobjekt-Klasse ist eine Gruppierung von Objekten mit gleichen Eigenschaften. Aufgrund der objektorientierten Eigenschaft der Vererbung kann eine Managementobjekt-Klasse als eine Unterklasse von einer oder mehreren Oberklassen (Mehrfachvererbung) abgeleitet werden, wobei sie alle Eigenschaften aller Oberklassen erbt. Somit kann sowohl eine Wiederverwendung als auch eine Spezialisierung von schon definierten Managementobjekt-Klassen erfolgen. Ebenso wird polymorphes Verhalten von Objekten unterstützt. Die Managementobjekte eines OSI-Systems bilden dessen MIB. Die Definiton von Managementobjekt-Klassen erfolgt in einer einfachen Template-Sprache (GDMO, *Guidelines for the Definiton of Managed Objects*, ISO10665-4), die auf ASN.1 basiert (für weitere Details und Beispiele siehe [HeAb93]).
Die folgenden drei Bäume beschreiben die Zusammenhänge, die Managementobjekten und Managementobjekt-Klassen zugrundeliegen:

1. **Registrierungsbaum**
   Für jede definierte Managementobjekt-Klasse muß folgendes registriert werden:

   - Management Object Class
   - Attribut-Definitionen
   - Aktionen
   - Notifikationen
   - Packages

   Der Registrierungsbaum kann dabei als Nachschlagewerk oder Bibliothek angesehen werden, in welchem alle Informationen verzeichnet sind, auf den die Definition von neuen Managementobjekt-Klassen aufbauen kann.

2. **Vererbungsbaum**
   Wie schon oben erwähnt, kann eine Managementobjekt-Klasse von anderen Managementobjekt-Klassen abgeleitet sein. Die damit entstehende Vererbungshierarchie wird im Vererbungsbaum dargestellt (siehe Abbildung 2.6). Die Wurzel des Vererbungsbaums ist stets die Managementobjekt-Klasse

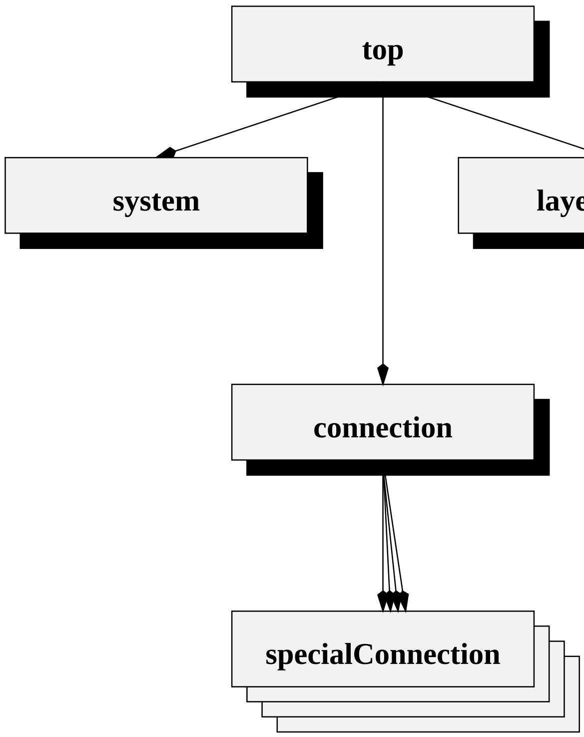
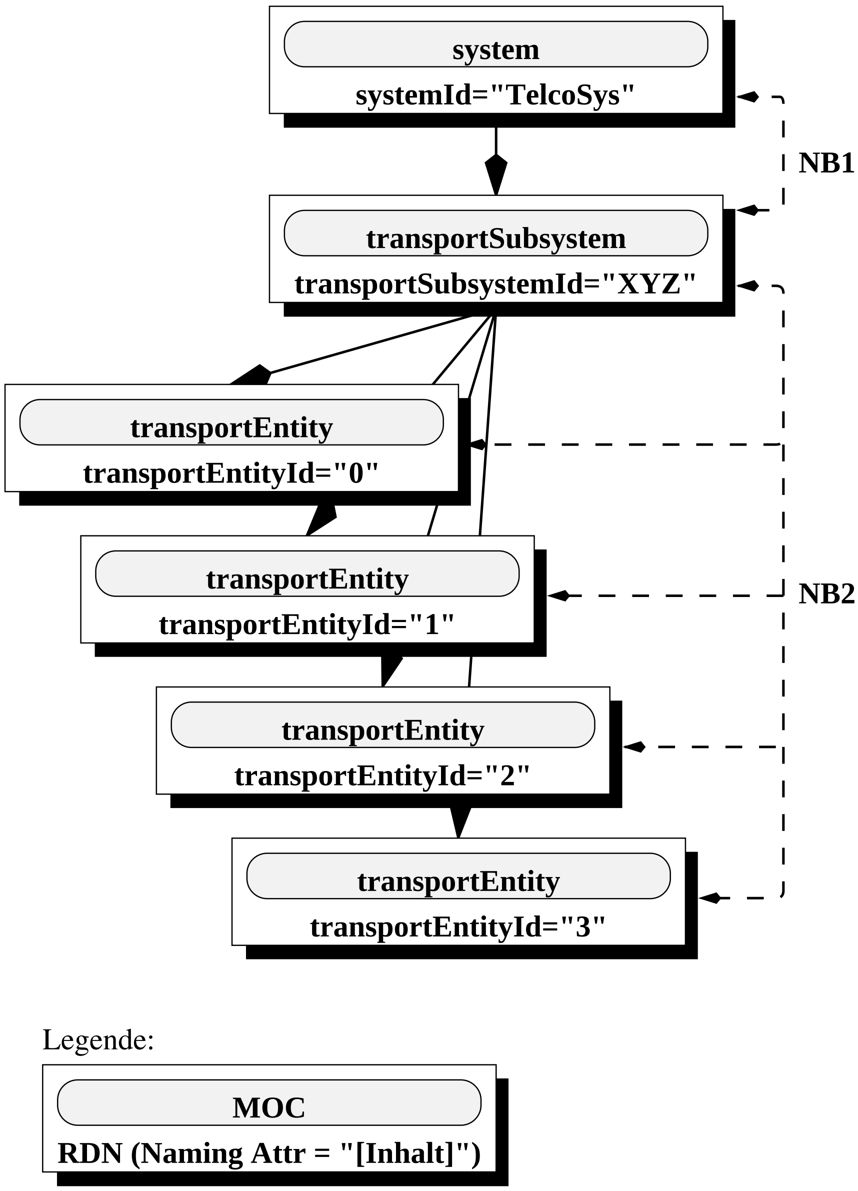


Abbildung 2.6: Beispiel einer OSI-Vererbungshierarchie

„top“.



NB1 : SUBORDINATE OBJECT CLASS **transportSubsystem**
NAMED BY SUPERIOR OBJECT CLASS **system**
WITH ATTRIBUTE **transportSubsystemId**

NB2 : SUBORDINATE OBJECT CLASS **transportEntity**
NAMED BY SUPERIOR OBJECT CLASS **transportSubsystem**
WITH ATTRIBUTE **transportEntityId**

Abbildung 2.7: Beispiel einer OSI-Enthaltenseinshierarchie

3. **Enthaltenseinsbaum**
   Der Containment-Baum gibt die Enthaltenseinshierarchie von Instanzen von Managementobjekt-Klassen an. Damit wird durch diesen Baum die Struktur der MIB angegeben. Die Wurzel des Enthaltenseinsbaum ist stets eine Instanz der OSI-Managementobjekt-Klasse „system“.
   Ein Beispiel für eine Enthaltenseinshierarchie stellt Abbildung 2.7 dar: Eine Instanz der Managementobjekt-Klasse „system“ stellt die Wurzel dar und enthält eine Instanz der Managementobjekt-Klasse „transportSubsystem“. Diese Instanz enthält mehrere Instanzen der Managementobjekt-Klasse „transportEntity“. Die Beziehungen von Instanzen von Managementobjekt-Klassen in der Enthaltenseinshierarchie werden von sogenannten *NAME BINDINGs* festgelegt. So muß, um auf oben vorgestelltes Beispiel zurückzukehren, für die Managementobjekt-Klasse „transportSubsystem“ ein NAME BINDING existieren, das definiert, daß eine Instanz dieser Klasse in einer Instanz der Klasse „system“ enthalten ist. Weiterhin wird mit diesem NAME BINDING die Namensgebung der Instanzen geregelt:
   Bisher wurden nur die Managementobjekt-Klassen in den Registrierungsbaum eingehängt und damit einem eindeutigen Namen zugeordnet (Pfad von der Wurzel zum Objekt). Für die Bezeichnung von Instanzen in der Containmenthierarchie und damit in der MIB des Agenten gelten folgende Regeln:
   Jede Mangementobjekt-Klasse muß ein sogenanntes *Naming Attribut* enthalten. Das *Naming Attribut* ergibt zusammen mit seinem Inhalt den *Relative Distinguished Name (RDN)* und stellt den Namen der Instanz dar. Der *Distinguished Name (DN)* einer Instanz ist ein eindeutiger Name, der sich jeweils aus den RDNs der Instanzen zusammensetzt, wenn man von der Wurzel der Enthaltenseinshierarchie zu der jeweiligen Instanz läuft:

RDN	DN
systemId=TelcoSys	systemId=TelcoSys
transportSubsystemId=XYZ	systemId=TelcoSys; transportSubsystemId=XYZ

transportEntityId=0	systemId=TelcoSys;
	transportSubsystemId=XYZ;
	transportEntityId=0

   Der Distinguished Name wird für die Lokalisierung eines Managementobjeks in der MIB verwendet.

Kommunikationsmodell

Das OSI-Kommunikationsmodell teilt sich in drei Kategorien auf, das *Schichtenübergreifende Management (Systems Management)*, das *Schichtenmanagement (Layer Management)* und das *Protokollmanagement (Layer Operation)*.

Das schichtenübergreifende Management betrifft das gesamte Management „offener" Systeme und basiert auf dem Management-Kommunikationsdienst *Common Management Information Service* (*CMIS*, ISO9595). CMIS stützt sich dabei auf das zugehörige Managementprotokoll *Common Management Information Protocol* (*CMIP*, ISO 9596) ab. CMIS und CMIP werden in der OSI-Schicht 7 eingebettet und benötigen damit einen kompletten OSI 7-Schichten-Protokollstack. Folgende Dienste werden von CMIS bereitgestellt:

- Dienste für das Ausführen von Managementoperationen:
  - **M-GET** (bestätigter Dienst): Lesen von Attributwerten von Instanzen von Managementobjekt-Klassen.
  - **M-SET** (bestätigter/unbestätigter Dienst): Setzen oder Modifizieren von Attributwerten von Instanzen von Managementobjekt-Klassen.
  - **M-CREATE** (bestätigter Dienst): Kreieren einer Instanz einer Managementobjekt-Klasse.
  - **M-DELETE** (bestätigter Dienst): Löschen von Instanzen von Managementobjekt-Klassen.
  - **M-ACTION** (bestätigter/unbestätigter Dienst): Auslösen einer Aktion auf Instanzen von Managementobjekt-Klassen.
  - **M-CANCEL-GET** (bestätigter Dienst): Die Bearbeitung eines *M-GET*-Dienstes abbrechen.
- Ein Dienst für das Weiterleiten von Ereignissen:
  - **M-EVENT-REPORT** (bestätigter/unbestätigter Dienst): Übertragen von Notifikationen einer Instanz einer Managementobjekt-Klasse.

Diese CMIS-Dienstprimitiven werden von der CMIP-Protokollmaschine entgegengenommen und zum Zielsystem übertragen.
Mit der Auslösung von den Dienstprimitiven werden unter anderem Parameter übergeben, mit denen eine Selektion der Instanzen von Managementobjekt-Klassen angegeben werden kann. Diese Selektion bewirkt, daß die angegebene Dienstprimitive nur auf diesen ausgewählten Instanzen ausgeführt wird. Die Managementobjekt-Auswahl beruht dabei auf

1. dem **Scope**, der einen Teilbaum (oder eine Schicht in diesem) der Enthaltenseinshierarchie darstellt und

2. dem **Filter**, der die Managementobjekte des vom *Scope* definierten Teilbaums (bzw. Schicht) weiter einschränkt, indem bestimmte Attributwerte verglichen werden.

Das Schichtenmanagement sieht Funktionen, Dienste und Protokolle vor, die spezifisch für eine Schicht und damit unabhängig von höheren Schichten sind.

Mit dem Protokollmanagement stehen Möglichkeiten zur Verfügung, Protokoll- bzw. Schicht-spezifische Bestandteile (zum Beispiel Timer oder Fenstergrößen) zu überwachen und zu steuern.

Funktionsmodell

Das sehr umfangreiche Funktionsmodell der OSI-Managementarchitektur gliedert sich in 5 Bereiche (den *Systems Management Functional Areas*, SMFA):

- **Konfigurationsmanagement**
- **Fehlermanagement**
- **Leistungsmanagement**
- **Sicherheitsmanagement**
- **Abrechnungsmanagement**

Für diese SMFAs existieren eine Reihe von allgemeinen Funktionen (*Systems Management Function*, SMF), zum Beispiel *Event Report* oder *Log Control*, siehe 3.2.1 (Detailliertere Beschreibungen der einzelnen SMFs befinden sich zum Beispiel in [Stal93].).

2.1.3 TMN-Managementarchitektur

Öffentliche Netzbetreiber haben mit der gleichen Situation zu kämpfen, wie sie auch bei privaten Netzen vorliegt: unterschiedliche proprietäre Managementlösungen verwalten jeweils Teilnetze. Die ITU legt im *Telecommunication Management Network (TMN)* Referenzmodell eine Managementarchitektur fest, wobei dafür ein eigenes Managementnetz vorgesehen ist. Es ist speziell auf die Bedürfnisse der öffentlichen Netzbetreiber zugeschnitten und soll ein integriertes Management unterstützen.
Die Vorstellung dieser Managementarchitektur erfolgt in drei Schritten:

1. Function Blocks
2. Reference Points
3. TMN-Architektur

Function Blocks

Eine der grundlegenden Konzepte der TMN-Architektur ist das der *Function Blocks*. Diese Einheiten definieren die Managementfunktionalität, die ein TMN zur Verfügung stellt:

- **Operations System Function (OSF)**:
  Dieser Funktion liegt eine TMN-Komponente zugrunde, die Managementinformation verarbeitet, um TMNs zu überwachen und zu steuern. Sie stellt das eigentliche Managersystem dar. Es können zum Beispiel verschiedene Managementapplikationen, mit einer Topologiedarstellung des Netzes als Mittelpunkt, in ein OSF eingebunden werden.

- **Network Element Function (NEF)**:
  Ein Netzelement stellt eine Komponente dar, die einem Teilnehmer eines Telekommunikationsnetzes einen bestimmten Dienst zur Verfügung stellt. Die NEF ermöglicht das Überwachen und Steuern der Netzelemente.

- **Workstation Function (WSF)**:
  Diese Funktion stellt eine Benutzerschnittstelle zum TMN bereit.

- **Mediation Function (MF)**:
  Dabei handelt es sich um ein Managementgateway („intermediate Agent"), das eine Verbindung zwischen einem NEF und einem OSF darstellen kann. So werden die Managementinformationen von NEFs mittels der MF einer höheren Ebene (OSFs) bereitgestellt. Dabei können folgende Funktionen unterstützt werden: Sammeln, Aufbereiten, Zusammenfassen oder Selektieren von Managementinformationen. Weiterhin kann hier eine Protokollkonvertierung stattfinden.

- **Q Adapter Function (QAF)**:
  Die QAF verbindet proprietäre und weitere, nicht TMN-konforme, Systeme mit dem TMN. Ebenso wie die NEF stellt die QAF einen Agenten für die zu managenden Ressourcen bereit, aber in diesem Fall bildet der Agent zusätzlich ein existierendes Protokoll auf eine TMN-Schnittstelle ab.

Reference Points

Die unterschiedlichen funktionalen Einheiten (*Function Blocks*) können über sogenannte *Reference Points* miteinander oder mit Systemen außerhalb des TMN kommunizieren. Diese Referenzpunkte stellen somit Schnittstellen dar und müssen durch Dienste/Protokolle spezifiziert werden. Es existieren folgende Referenzpunkte:

- **f**: Eine Schnittstelle zwischen einer WSF und einer OSF bzw. einer MF.

- **q3**: Dieser Referenzpunkt stellt die Schnittstelle zwischen den QAFs, den NEFs oder den MFs zu den OSFs dar. Es können aber auch zwei OSFs über diese Schnittstelle miteinander kommunizieren.

- **qx**: Eine Schnittstelle zwischen QAFs und NEFs zu MFs bzw. zwischen zwei MFs.

- **x**: Dieser Referenzpunkt verbindet verschiedene TMNs miteinander.

- **g**: Der Verbindungspunkt zwischen einem menschlichen Benutzer und der WSF. Diese Schnittstelle befindet sich außerhalb der TMN-Architektur.

- **m**: Diese stellt eine Schnittstelle von nicht TMN-konformen Systemen mit QAFs dar. Diese befindet sich ebenfalls außerhalb der TMN-Architektur.

Die Definition dieser Schnittstellen ist noch nicht abgeschlossen. Einen der wichtigsten Referenzpunkte stellt *q3* dar. Er definiert ein CMIP-Interface zwischen zwei funktionalen Einheiten. *qx* ist eine vereinfachte Variante von *q3*.

TMN-Architektur

Die TMN-Architektur definiert verschiedene Schichten, in welcher die bisher erläuterten funktionalen Einheiten und Referenzpunkte folgendermaßen integriert werden (siehe Abbildung 2.8 nach [IBMPrGI95]):

- **Network Element Layer (NEL)**:
  In dieser Schicht befinden sich die QAFs und die NEFs. Sie binden die zu überwachenden und zu steuernden Ressourcen in das TMN ein und liefern damit relevante Managementinformationen.

- **Element Management Layer (EML)**:
  Hier existieren entweder OSFs (für die Überwachung und Steuerung der Netzelemente) oder MFs (für das Aufbereiten von Managementinformationen von den Netzelementen für höhere Schichten).

- **Network Management Layer (NML)**:
  Die in dieser Schicht realisierten OSFs managen die Systeme in den darunterliegenden Schichten.

- **Service Management Layer (SML)**:
  Diese Schicht ermöglicht das Management von Telekommunikationsdiensten.

- **Business Management Layer (BML)**:
  Damit werden Applikationen bereitgestellt, die, auf der Grundlage der OSFs, Dienste für zum Beispiel die Abrechnung oder die Auftragseingabe realisieren.

Die Begründung für die Vorstellung der TMN-Managementarchitektur im Zusammenhang mit dieser Diplomarbeit liegt darin, daß die in Abschnitt 3.2 vorgestellte Entwicklungsumgebung *IBM TMN WorkBench for AIX* nicht nur die Implementierung von OSI-, sondern auch von TMN-Managementlösungen unterstützt. Daher wird auch im Laufe dieser Ausarbeitung auf einige Konzepte der TMN-Managementarchitektur zurückgegriffen.

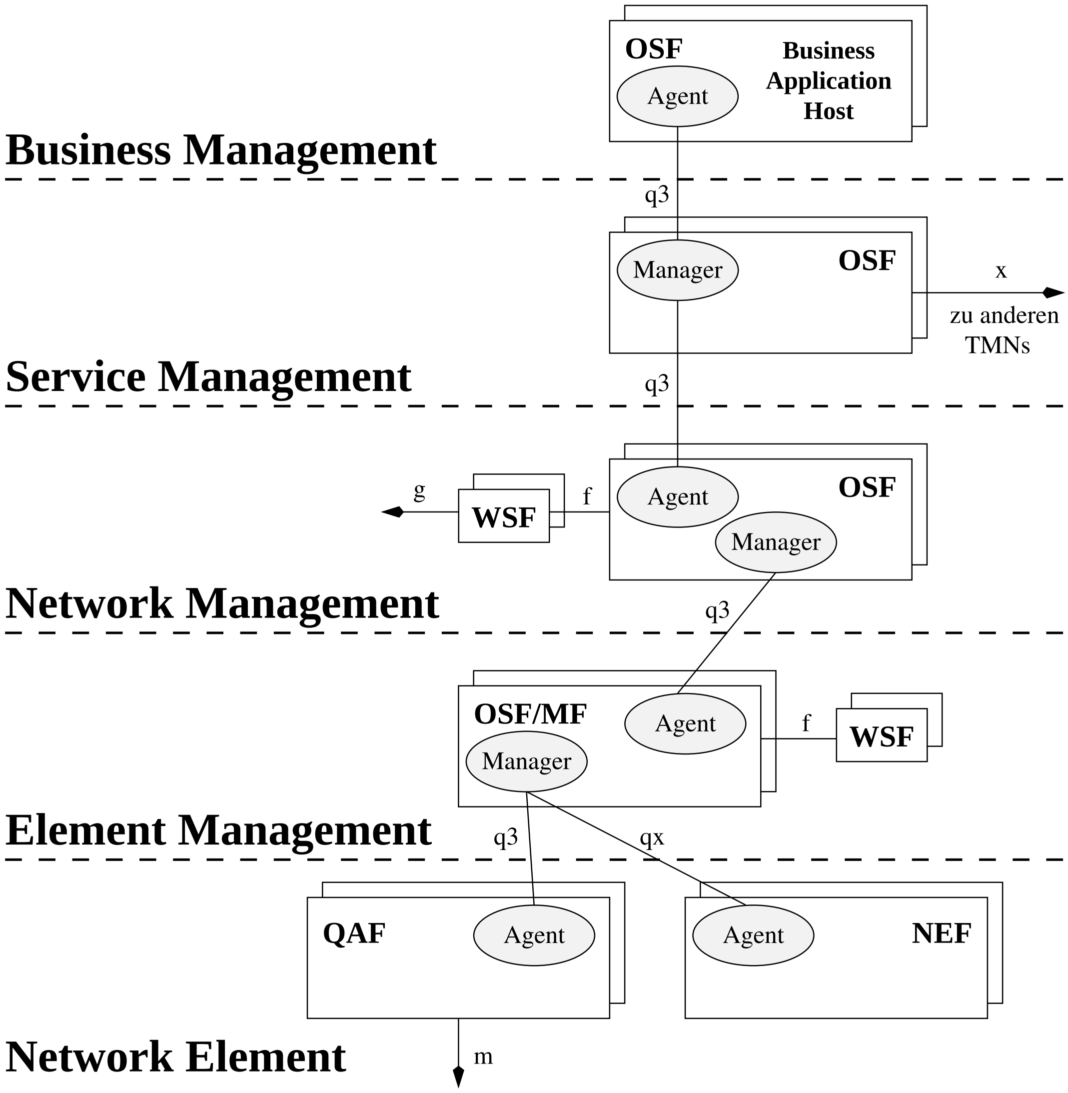


Abbildung 2.8: TMN-Schichten, funktionale Einheiten und Referenzpunkte

2.2 Integration von Managementarchitekturen

Eine der zentralen Forderungen des „integrierten Managements“ ist die Verknüpfung von Managementarchitekturen. Denn erst Interoperabilität zwischen Managementarchitekturen ermöglicht ein einheitliches Management in heterogenem Umfeld. Für die Integration einer Managementarchitektur in eine andere können drei konzeptionell verschiedene Ansätze bestimmt werden:

1. Multiarchitekturelle Plattformen
2. Managementgateways
3. Multiarchitekturelle Agenten

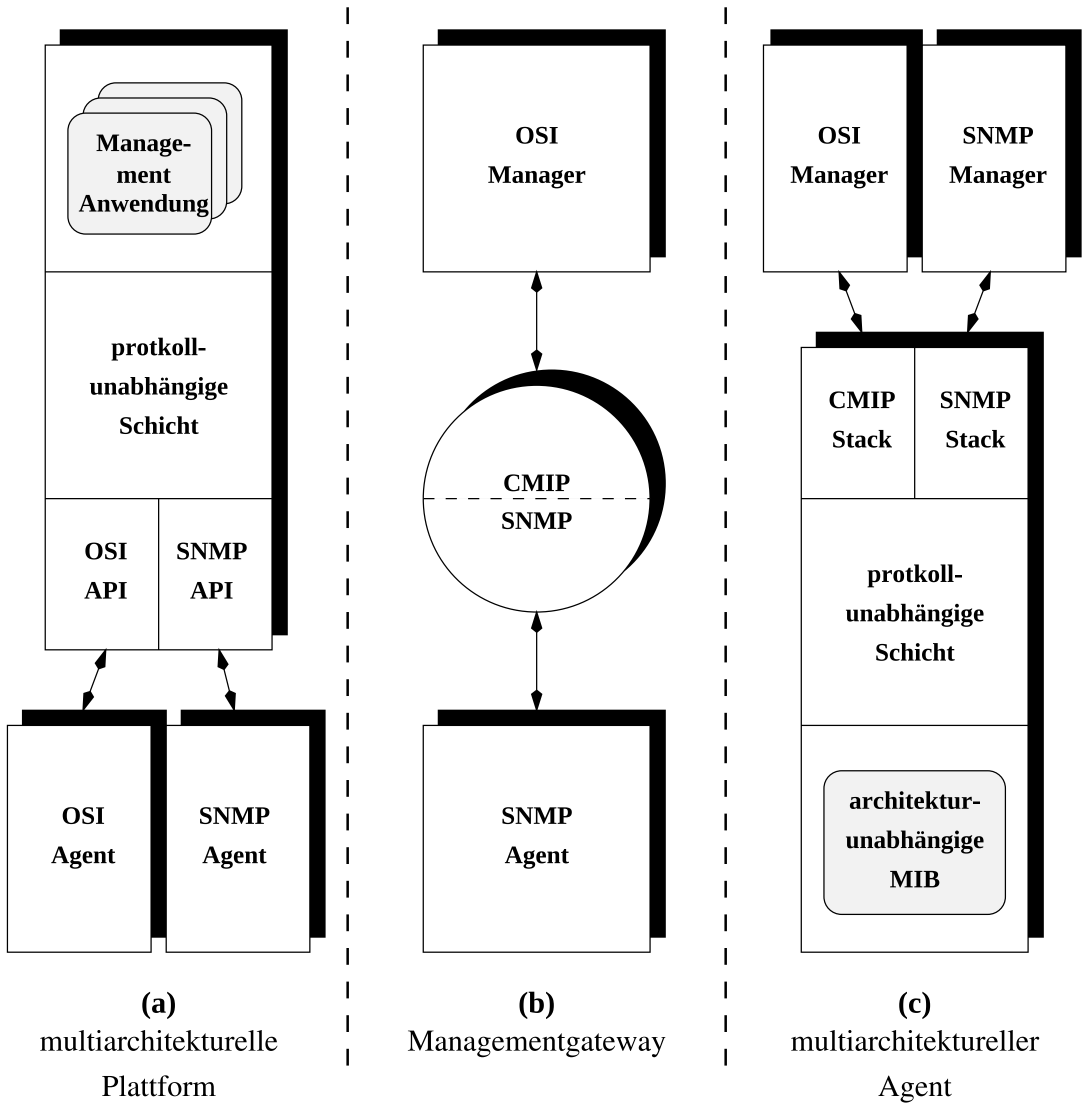


Abbildung 2.9: Ansätze zur Integration von Managementarchitekturen

2.2.1 Multiarchitekturelle Plattformen

Dieser Ansatz stellt ein dreigeteiltes Managersystem dar, siehe Abbildung 2.9 (a). Auf der obersten Ebene werden die Managementapplikationen mit den entsprechenden Benutzerschnittstellen angesiedelt. Die mittlere Schicht stellt eine protokollunabhängige Schnittstelle zu den in der untersten Ebene existierenden Protokollmaschinen zur Verfügung. Die Beispielplattform in Abbildung 2.9 (a) unterstützt sowohl die OSI- als auch die Internet-Managementarchitektur und für jede ist eine eigene Protokollmaschine implementiert. Die Managementanwendungen erhalten über die mittlere, protokollunabhängige Schicht (ein Beispiel dafür ist *XMP*) Funktionen bereitgestellt, die den Zugriff auf die jeweiligen darunterliegenden Protokollmaschinen transparent gestalten. Damit werden die Managementanwendungen unabhängig von Managementarchitekturen, aber nicht von der protokollunabhängigen Schicht.
Diese multiarchitekturellen Plattformen werden aufgrund der Implementierung der protokollunabhängigen Schicht zu sehr komplexen Managersystemen. Damit ist es aber möglich, auf der Basis sämtlicher integrierter Managementarchitekturen architekturunabhängige Managementanwendungen zu realisieren.

2.2.2 Managementgateways

Eine zweite Möglichkeit für die Integration einer Managementarchitektur in eine andere stellt die Verwendung von Managementgateways dar. Managementgateways bilden dabei alle vier Teilmodelle (Organisations-, Informations-, Kommunikations- und Funktionsmodell) der einen Managementarchitektur auf die der anderen ab (siehe Abbildung 2.9 (b)). Das Beispielgateway muß den Zugriff des OSI-Managers auf den SNMP-Agenten transparent gestalten, das heißt, der OSI-Manager soll nicht unterscheiden können, ob er einen OSI-Agent oder einen SNMP-Agent überwacht und steuert. In der Literatur können auch weitere Bezeichnungen für Managementgateways gefunden werden, so zum Beispiel *Application Gateway* in [KaSe93] oder *Proxy-Agent* in [HeAb93].

2.2.3 Multiarchitekturelle Agenten

Im dritten und letzten Ansatz findet die Verknüpfung von zwei oder mehreren Managementarchitekturen im Agenten statt (siehe Abbildung 2.9 (c)): Auf den Beispielagenten können sowohl OSI-Manager als auch SNMP-Manager zugreifen. Er implementiert dazu für jede zu unterstützende Managementarchitektur eine eigene Protokollmaschine. Eine protokollunabhängige Schnittstelle führt alle Anforderungen von allen Protokollmaschinen an der architekturunabhängigen MIB aus. Ein Beispiel für einen derartigen protokollunabhängigen Agenten wird in [BrLeMa93] beschrieben.

Diese multiarchitekturellen Agenten werden aufgrund der Realisierung der architekturunabhängigen MIB zu sehr komplexen Agentensystemen.

2.2.4 Diskussion

Der große Nachteil der multiarchitekturellen Plattformen bzw. Agenten ist, daß jeweils existierende Plattformen- bzw. Agentenimplementierungen verändert werden müssen.
Ein multiarchitektureller Agent hat weiterhin den Nachteil, daß für jede zu unterstützende Architektur eine eigene Protokollmaschine implementiert werden muß. Es kann in einem sehr heterogenen Kommunikationsnetz nicht vorausgesetzt werden, daß überall alle Protokolle zur Verfügung stehen. Somit kann dieser Agent nicht überall eingesetzt werden.
Eine multiarchitekturelle Plattform hat zusätzlich den Nachteil, daß nicht nur die Plattform selbst, sondern auch alle Managementapplikationen, die auf dieser existieren, verändert werden müssen.
Der Vorteil von Managementgateways ist, daß in sämtlichen bestehenden Plattformen, Managementanwendungen oder Agenten keine Veränderungen notwendig sind und somit weiterverwendet werden können. So wird in dieser Diplomarbeit dieser Vorteil ausgenutzt und für die Integration des Internet-Managements in die OSI-Managementarchitektur ein Managementgateway eingesetzt.

2.3 Relevante Management Information Bases

In den folgenden beiden Abschnitten sollen zwei Internet-MIBs kurz vorgestellt werden, da ihnen bei der Implementierung des Gateways eine wichtige Rolle zukam:

2.3.1 MIB-2

Nachdem im Mai 1990 die erste Standard-MIB (MIB-1) für das Internet-Management (RFC 1156) veröffentlicht wurde, gab es nur Monate später schon die zweite Version, die allgemein mit *MIB-2* bezeichnet wird. Die MIB-2 ist dabei eine Obermenge der MIB-1, mit zusätzlichen Objekten und zusätzlichen Gruppen.

Kriterien für die in der MIB-2 enthaltenen Objekte

- Ein Objekt soll entweder für das Fehlermanagement oder für das Konfigurationsmanagement relevant sein.

- Es werden nur solche Objekte betrachtet, die durch Mißbrauch nur begrenzten Schaden verursachen. Dieser Punkt spiegelt die Tatsache wieder, daß dem Managementprotokoll Sicherheitsmängel zugrundeliegen.

- Es wird ein praktischer Nutzen gefordert.
- Es wurde festgeschrieben, um redundante Variablen zu vermeiden, daß ein Objekt nicht von bereits existierenden Objekten abgeleitet werden kann.
- In der MIB-1 gab es noch ein weiteres Kriterium: Es wurde eine Obergrenze für die Objektanzahl (maximal 100 Objekte) festgelegt. Bei der Erstellung der MIB-2 wurde dieser Punkt bereits nicht mehr berücksichtigt.
- Implementierungsspezifische Objekte (zum Beispiel für BSD-Unix) wurden ausgeschlossen.
- Es wurde berücksichtigt, daß kritische Codeteile nicht mit der Definition von Objekten zusätzlich belastet werden. Die allgemeine Richtlinie lautet: ein Zähler pro kritischem Codeteil pro Schicht.

Die Struktur der MIB-2

Die Abbildung 2.10 zeigt die Struktur der MIB-2: Die insgesamt 10 Gruppen

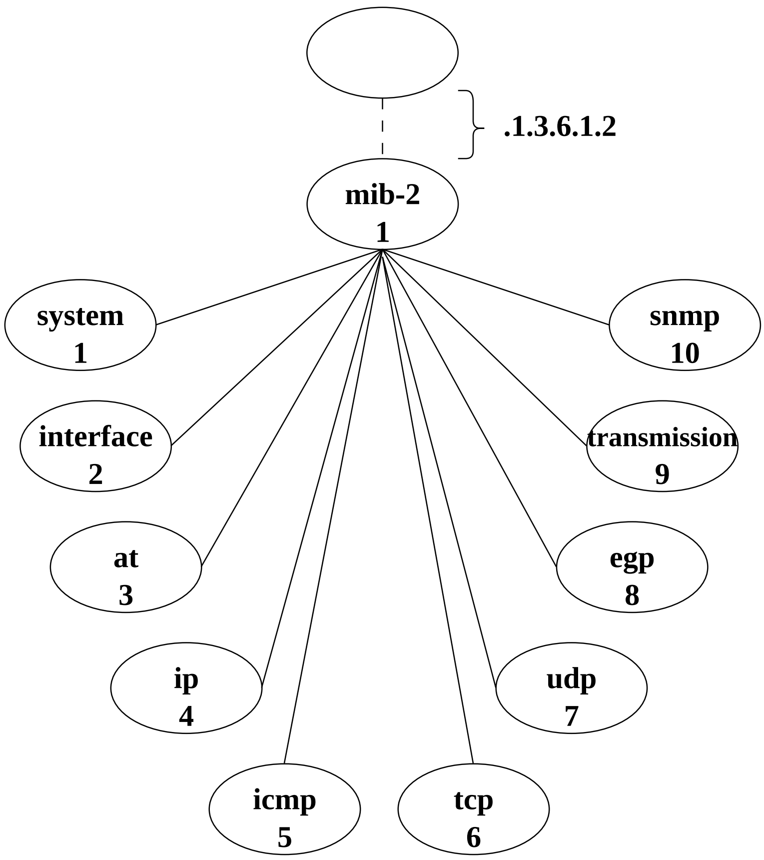


Abbildung 2.10: Die Gruppen in der MIB-2

gliedern dabei die gesamte Information in folgender Weise auf:

- `system`:
  Konfigurationsinformationen über den zu überwachenden Node.

- `interface`:
  Diese Gruppe setzt sich aus der Anzahl der von diesem Node bereitgestellten Schnittstellen und einer Tabelle, die für jede Schnittstelle den Typ, Informationen zur Konfiguration, Zustandsattribute und Statistikzähler bereitstellt, zusammen.

- `at`:
  Informationen zur Auflösung von Adressen.

- Die Protokollgruppen `egp`, `icmp`, `ip`, `snmp`, `tcp` und `udp`:
  In jeder dieser Gruppe werden Statistikzähler für unter anderem

  - ein-/ausgehende PDUs,
  - Typ der PDUs und
  - aufgetretene Fehler

  geführt. Weiterhin werden protokollspezifische Informationen wie Routing- oder Verbindungstabellen gehalten.

- `transmission`:
  Informationen über spezifische Typen von Netzschnittstellen wie Ethernet, Token Ring usw.

2.3.2 Eine Internet-MIB für das Management von UNIX-Workstations

„Zum heutigen Zeitpunkt muß trotz zahlreicher Bemühungen der Hersteller die Problematik, integriertes Management von UNIX-Workstations sicherzustellen, noch immer als ungelöst betrachtet werden: Es existieren zwar Werkzeuge, die eine Vielzahl von Parametern eines Endsystems erfassen und mit Hilfe eines standardisierten Managementprotokolls an die Managementplattform weiterleiten; aktive Eingriffsmöglichkeiten durch den Administrator fehlen jedoch völlig. Andererseits existieren Produkte, die zwar Eingriffe zur Steuerung des Systems erlauben, die Kommunikation mit der Managementplattform geschieht jedoch lediglich auf der Grundlage eines herstellerspezifischen, proprietären Protokolls, dessen Schnittstellen nicht offengelegt sind. Zusätzlich sind beide Ansätze von einer Bottom-Up-Vorgehensweise geprägt, die nur selten die tatsächlichen Bedürfnisse der Netzbetreiber berücksichtigt. Von einer umfassenden, integrierten Managementlösung ist man daher noch weit entfernt.

Im Gegensatz zu kommerziell erhältlichen Werkzeugen haben die Mitarbeiter am Lehrstuhl von Professor Heinz-Gerd Hegering in ihren Arbeiten einen Top-Down-Ansatz verfolgt, der das typische Aufgabenspektrum eines UNIX-Systemadministrators abdeckt. Es wurden daher Möglichkeiten für das Einrichten und Löschen von Benutzerkennungen und -gruppen sowie deren Quoten für den Zugriff auf Betriebsmittel (Plattenplatz, Druckseiten) ebenso vorgesehen, wie Funktionen zum Erfassen und ggf. Stoppen der momentan aktiven Prozessen oder Mounten/Unmounten von Dateisystemen. Es ist somit nicht nur passives Monitoring von UNIX-Workstations möglich, sondern auch das aktive Eingreifen in den Betrieb des Systems. Als Managementprotokoll wird SNMP in der Version 2 verwendet.

Im Rahmen dieser Arbeit wurde eine Systemmanagement-MIB entwickelt und

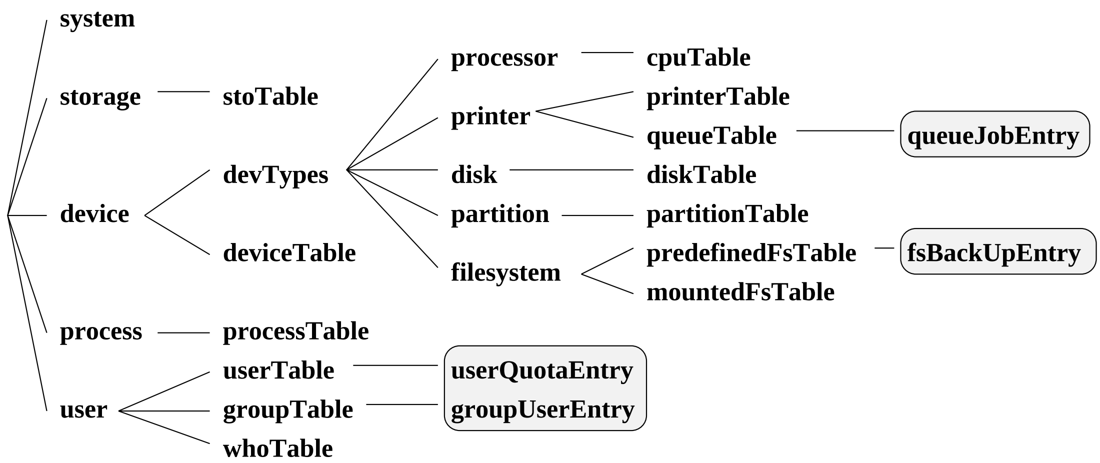


Abbildung 2.11: Eine MIB für das Management von UNIX-Workstations

der dazu gehorige Agent auf verschiedenen Betriebssystemplattformen (IBM AIX, HP-UX, SunOS, Solaris) implementiert. Diese MIB ist in Abbildung 2.11[3] schematisch dargestellt; sie umfaßt 195 MIB-Variablen und 15 Tabellen und stellt (unter anderem) ein Modell folgender Komponenten eines UNIX-Systems zur Verfügung:

- Speicher (Hauptspeicher, Swap-Bereich)
- Geräte (Prozessoren, Drucker, Platten und deren Dateisysteme usw.)
- Prozesse
- Benutzer (Kenndaten, Gruppen, Quoten usw.)

Im Falle der Benutzer- und Gruppenverwaltung tritt jedoch durch eine Einschränkung des Internet-Informationsmodells unweigerlich folgendes Problem auf: in der Regel sind auf einem UNIX-System mehrere Benutzer eingetragen, die ihrerseits wieder zu mehreren Gruppen gehören. Tabellen innerhalb von Tabellen

[3]nach [GuNe95]

sind jedoch explizit durch den entsprechenden Internet-Standard [RFC1155] untersagt.“[4]
Daher wird in dieser Diplomarbeit nur von folgender, in Abbildung 2.12 dargestellter, eingeschränkter MIB ausgegangen: Es wurden von der originalen Sys-

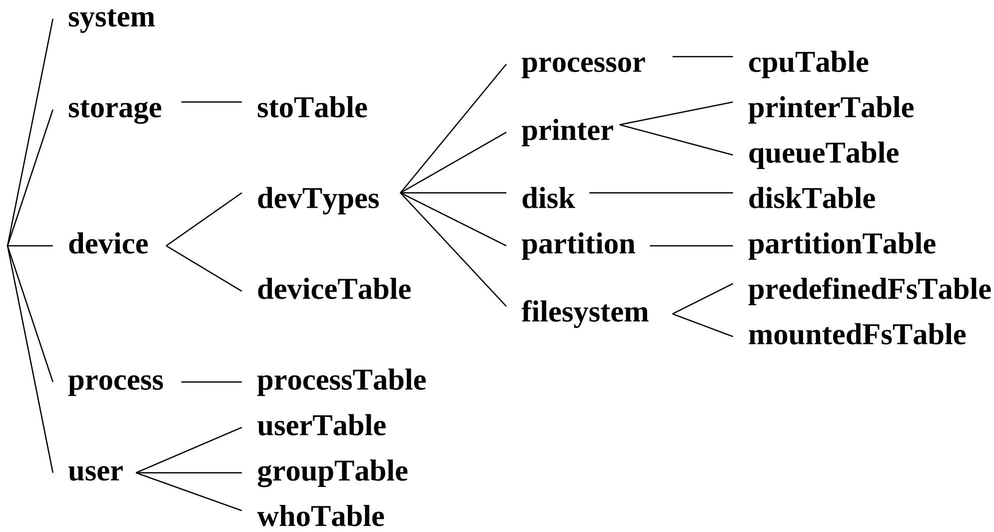


Abbildung 2.12: Diese Systemagenten-MIB ohne Tabellen innerhalb von Tabellen

temagenten-MIB alle Tabellen innerhalb anderer Tabellen weggelassen. Dies war erforderlich, da der in 3.1.2 vorgestellte MIB-Compiler diese Abweichung vom Standard nicht akzeptierte.

[4]nach [HeKeNe96]

Kapitel 3

Vorhandene Algorithmen und Werkzeuge

In diesem Kapitel werden die Algorithmen und Werkzeuge beschrieben, auf deren Grundlage bzw. mit deren Hilfe das CMIP/SNMP Gateway realisiert wurde.
Dabei handelt es sich zum einen um den vom NM-Forum standardisierten Algorithmus für die Konvertierung von *Internet MIB Macro Format* in *ISO/ITU GDMO Format*. Die Einführung eines, auf diesen Spezifikationen basierenden, allgemein verfügbaren Compilers folgt anschließend.
Zum anderen wird die *IBM TMN WorkBench for AIX* Produktpalette für das Entwerfen und Entwickeln von Managementlösungen für die TMN-Architektur vorgestellt.

3.1 Die Arbeiten der IIMC

In Zuge des zunehmenden Bedürfnisses an der Integration des Internet-Managements in die OSI-Architektur entstanden Anfang der 90er Jahre wichtige Vorarbeiten im Rahmen der *ISO-Internet Management Coexistence (IIMC)*. Das IIMC-Paket umfaßt dabei folgende Dokumente:

- **IIMCIMIBTRANS** : Die Übersetzung von Internet-MIBs zu OSI GDMO MIBs, siehe [IIMCIMIBTR] oder den nächsten Abschnitt.
- **IIMCOMIBTRANS** : Die Übersetzung von OSI GDMO MIBs zu Internet-MIBs, siehe [IIMCOMIBTR].
- **IIMCMIB-II** : Die Übersetzung der Internet MIB-2 [RFC1213] zu einer OSI GDMO MIB, siehe [IIMCMIB-II] oder Teile daraus im nächsten Abschnitt.
- **IIMCPROXY** : ISO/CCITT to Internet Proxy, siehe [IIMCPROXY] oder auch eine Beschreibung in 4.4.1.

- **IIMCSEC** : ISO/CCITT to Internet Management Security, siehe [IIMCSEC].

Für die Realisierung eines CMIP/SNMP Gateways ist eine Abbildung des SNMP- auf das OSI-Informationsmodell notwendig (siehe Kapitel 4). Diese Informationsmodellabbildung erfolgt durch eine MIB-Übersetzung, deren Algorithmus im folgenden Abschnitt vorgestellt wird. Anschließend wird ein MIB-Compiler beschrieben, der diese Informationsmodellabbildung zu automatisieren versucht.
Da dieser MIB-Compiler nur mit SNMPv1-MIBs arbeitet, wird sich die spätere Architektur des CMIP/SNMP Gateways auch nur mit SNMPv1 auseinandersetzen.

3.1.1 Der IIMC Übersetzungsalgorithmus

Die *IIMC* veröffentliche in einem Internet Draft [IIMCIMIBTR] einen Algorithmus für die Übersetzung von Internet MIBs[1] zu ISO/ITU SMI-konformen MIBs[2]. Dieser Algorithmus soll in diesem Abschnitt kurz vorgestellt werden, damit dessen Anwendung im späteren Teil der Diplomarbeit erfolgen kann. Für detailliertere Betrachtungen wird auf den Internet Draft verwiesen.

Registrierung und Namensgebung

Die Registrierung und Namensgebung sind für die eindeutige Identifikation von Managementinformation entscheidende Prozeduren. Die Registrierung sichert dabei die Eindeutigkeit des Element-Typs der Managementinformation. Die Namensgebung ist für die Zusammensetzung des *Distinguished Name* einer Instanz und damit für die Lokalisierung dieser Instanz in einer MIB verantwortlich.
Bei der Übersetzung von Internet MIBs in ISO/ITU GDMO MIBs entstehen neue ISO/ITU Objekt-Klassen, Attribute und Notifikationen, die zuerst registriert werden müssen. Zusätzlich müssen auch die *NAME BINDINGs*, die für das entsprechende Einsetzen der Instanz in die Naming-Hierarchie verantwortlich sind, registriert werden.
Managementinformation, die nach den Prinzipien der Internet SMI entwickelt wurde, wird mit Hilfe eines ASN.1 OBJECT IDENTIFIER (OID) registriert. Dieser OID bezeichnet dabei einen eindeutigen Pfad im „internet"-Ast oder einen Ast in einem proprietären Teilbaum des ISO-Registrierungsbaumes. Diese Eindeutigkeit in der Registrierung der Internet-Managementinformation wird dadurch ausgenutzt, daß die aus der Internet-Mangementinformation resultierende GDMO-Managementinformation in einem anderen Teilbaum des ISO-Registrierungsbaumes mit der Internet-OID abgespeichert wird. Das folgende Beispiel zeigt die Registrierung:

[1] das heißt, sie sind nach der Internet-SMI definiert

[2] Dieser Algorithmus aus dem Internet Draft wurde inzwischen vom NM-Forum standardisiert.

- Registrierungsteilbäume für die übersetzten Managementinformationen :
  Für Klassen und Attribute : iimcAutoObjAndAttr = .1.2.124.360501.14.1
  Für NAME BINDINGs : iimcAutoNameBinding = .1.2.124.360501.14.2
  Für Naming Attribute : iimcAutoNaming = .1.2.124.360501.14.3

- Internet-Managementinformation : sysContact
  Registriert unter : .1.3.6.1.2.1.1

- Übersetzte GDMO-Managementinformation : sysContact (Attribut)
  Registriert unter : .1.2.124.360501.14.1.1.3.6.1.2.1.1
  (iimcAutoObjAndAttr + Internet OID)

NAME BINDINGS und Naming Attribute werden entsprechend in *iimcAutoNameBinding* bzw. *iimcAutoNaming* registriert.

Die Namensgebung für ein OSI-Managementobjekt erfolgt durch ein gesondert gekennzeichnetes Attribut, das sogenannte *Naming Attribut*. Der Name des Attributs und sein Inhalt ergeben zusammen den *Relative Distinguished Name (RDN)* der Instanz. Der Internet Draft spezifiziert dabei für das *Naming Attribut* eine Regel für die Namensgebung, eine Regel für die Registrierung, die Wert-Definition und die automatische Generierung des *Naming Attributes* für jede ISO/ ITU Klasse, die aus der Übersetzung einer Internet MIB entsteht:

- Die Regel für die Namensgebung des Naming Attributs lautet : Füge an den Namen der Objekt-Klasse, zu welcher das Attribut gehört, den Suffix „Id“ an.

- Die Registrierung des Naming Attributs erfolgt im Teilbaum *iimcAutoNaming* mit der Internet OID der Objekt-Klasse, zu welcher das Attribut gehört.

- Der Wert des Attributes wird entweder ASN.1 NULL oder der Inhalt des Internet Objekts *INDEX* zugewiesen:
  Für Managementobjekt-Klassen, von denen nur eine Instanz pro Internet MIB existieren darf (zum Beispiel, die GDMO Klassen, die aus den Internet MIB-2 Gruppen tcp, ip, system entstanden sind), wird der Wert des Attributes auf ASN.1 NULL gesetzt.
  Für Managementobjekt-Klassen, von denen mehrere Instanzen pro Internet MIB existieren können (dem entsprechen die GDMO Klassen, die SNMP-Tabellenzeilen repräsentieren, zum Beispiel aus der Internet MIB-2 tcpConnEntry, ipAddrEntry), wird dem Wert des Attributes der jeweilige *INDEX* der SNMP-Tabellenzeile zugewiesen.

Der Algorithmus

Der Algorithmus für die Übersetzung einer Internet MIB in eine GDMO MIB setzt sich aus folgenden Regeln zusammen:

- Übersetzen von Gruppen:
  Internet Gruppen (zum Beispiel aus der MIB-2: system, ip ,tcp ...) werden zu Managementobjekt-Klassen übersetzt:
  Internet-MIB-2-Gruppe „system“:

```
-- groups in MIB-II
...
system       OBJECT IDENTIFIER ::= { mib-2 1 }
...
```

  Übersetzte GDMO-Klasse „internetSystem“:

```
internetSystem MANAGED OBJECT CLASS
        DERIVED FROM
           "CCITT Rec. X.721 (1992) |
                ISO/IEC 10165-2 : 1992": top;
        CHARACTERIZED BY internetSystemPkg PACKAGE
        BEHAVIOUR
           internetSystemPkgBehaviour BEHAVIOUR
           DEFINED AS
           !BEGINPARSE
           REFERENCE
              !!This managed object class maps to the
              Internet system group with object id
              {mib-2 1} in RFC 1213.!!;
           DESCRIPTION
              !!If an agent is not configured to have a
              value for any of these variables, a string
              of length 0 is returned.
              [...]
           ENDPARSE!;;
        ATTRIBUTES
           internetSystemId              GET,
           sysDescr                      GET,
           sysObjectID                   GET,
           sysUpTime                     GET,
           sysContact                    GET-REPLACE,
           sysName                       GET-REPLACE,
           sysLocation                   GET-REPLACE,
```

```
   sysServices                  GET;
NOTIFICATIONS
         "IIMC MIB Translation":internetAlarm;;;
REGISTERED AS
   {1 2 124 360501 14 1 1 3 6 1 2 1 1};
```

Dieses Beispiel zeigt zusätzlich, daß bei der Übersetzung Namenskonflikte auftreten können: Im Internet-Registrierungsbaum wird eine SNMP-Gruppe mit dem Namen „system“ definiert (siehe 2.3.1). Ebenso existiert als Root-Objekt für die Enthaltenseins-Hierarchie eine OSI-Klasse mit dem Namen „system“[3]. Die Internet-Gruppe darf also nicht ohne Namensänderung direkt übersetzt werden: Der Name der OSI-Klasse, die aus der Übersetzung dieser SNMP-Gruppe resultiert wird zu „internetSystem“. Der gleiche Fall liegt bei der „LRZ Systemagenten MIB“ vor (siehe 2.3.2): Es existiert auch hier eine SNMP-Gruppe mit dem Namen „system“. Die aus dieser Gruppe resultierende OSI-Klasse erhält den Namen „lrzSystem“.
Falls im weiteren von einer Klasse mit dem Namen „system“ gesprochen wird, handelt es sich dabei immer um die OSI-Klasse „system“[4], die als Root-Objekt der Containment-Hierarchie dient. Ist dagegen von einer SNMP-Gruppe „system“ die Rede, kann es sich entweder um eine Gruppe in der *MIB-2* oder in der *LRZ Systemagenten MIB* handeln.

- Übersetzen von Tabellen:
  Es erfolgt keine direkte Übersetzung der Tabellendefinition (zum Beispiel aus der MIB-2: tcpConnTable, ipAddrTable,...) selbst. Es werden nur die Internet Tabellenzeilen (zum Beispiel aus der MIB-2: tcpConnEntry, ipAddrEntry ...) zu Managementobjekt-Klassen übersetzt:
  Internet-MIB-2-Gruppe „ipAddrEntry“:

```
...
ipAddrEntry OBJECT-TYPE
      SYNTAX  IpAddrEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
         "The addressing information for one of this
         entity´s IP addresses."
      INDEX   { ipAdEntAddr }
      ::= { ipAddrTable 1 }

IpAddrEntry ::=
```

[3]Diese Klasse wird in [ISO10165-2] definiert.
[4]Wie sie in [ISO10165-2] definiert wird.

```
            SEQUENCE {
                ipAdEntAddr
                    IpAddress,
                ipAdEntIfIndex
                    INTEGER,
                ipAdEntNetMask
                    IpAddress,
                ipAdEntBcastAddr
                    INTEGER,
                ipAdEntReasmMaxSize
                    INTEGER (0..65535)
            }
      ...
```

Übersetzte GDMO-Klasse „ipAddrEntry“:

```
ipAddrEntry MANAGED OBJECT CLASS
        DERIVED FROM
           "CCITT Rec. X.721 (1992) |
                ISO/IEC 10165-2 : 1992": top;
        CHARACTERIZED BY ipAddrEntryPkg PACKAGE
        BEHAVIOUR
           ipAddrEntryPkgBehaviour BEHAVIOUR
           DEFINED AS
           !BEGINPARSE
           REFERENCE
           !!This managed object class maps to the
           ipAddrEntry object with object id {ipAddrTable 1}
           in RFC 1213.!!;
           DESCRIPTION
              !!The addressing information for one of this
              entity´s IP addresses.!!;
              INDEX RFC1213-MIB.ipAdEntAddr;
           ENDPARSE!;;
        ATTRIBUTES
           ipAddrEntryId            GET,
           ipAdEntAddr              GET,
           ipAdEntIfIndex           GET,
           ipAdEntNetMask           GET,
           ipAdEntBcastAddr         GET,
           ipAdEntReasmMaxSize      GET;;;
        REGISTERED AS
           { 1 2 124 360501 14 1 1 3 6 1 2 1 4 20 1};
```

- Alle anderen Objekte der Internet-SMI:
  Alle anderen Internet Objekte werden zu Attributen in den entsprechenden Klassen, zum Beispiel aus der MIB-2: sysContact, sysLocation werden zu ISO/ITU Attributen in der Klasse, die aus der Internet Gruppe „system“ entstanden ist:
  Internet-MIB-2-Variable „sysContact“:

```
        [...]
        sysContact OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-write
              STATUS  mandatory
              DESCRIPTION
                 "The textual identification of the
                 contact person for this managed node,
                 together with information on how to
                 contact this person."
        ::= { system 4 }
        [...]
```

  Übersetztes GDMO-Attribut „sysContact“:

```
sysContact ATTRIBUTE
        WITH ATTRIBUTE SYNTAX
                IIMCRFC12131354ASN1.DisplayString;
        BEHAVIOUR
           sysContactBehaviour BEHAVIOUR
           DEFINED AS
              !BEGINPARSE
              REFERENCE
              !!This attribute maps to sysContact with
              object id {system 4} in RFC1213.!!;
              DESCRIPTION
                 !!The textual identification of the
                 contact person for this managed node,
                 together with information
                 on how to contact this person.!!;
              ENDPARSE!;;
        REGISTERED AS
           {1 2 124 360501 14 1 1 3 6 1 2 1 1 4};
```

- Übersetzung von Notifikationen:
  Ereignismeldungen werden in der Internet-Architektur nicht im Informationsmodell, sondern im Kommunikationsmodell definiert. Das heißt, Traps,

die von einem SNMP-Agent zu einem SNMP-Manager geschickt werden, sind kein Teil der MIB, sondern ein Teil des SNMP-Protokolls. In der OSI-Architektur werden dagegen Notifikationen von speziellen Managementobjekten ausgelöst, sie sind damit ein Teil des Informationsmodells.
Aufgrund dieser Unterschiede kann ein SNMP-Trap häufig keinem OSI Managementobjekt eindeutig zugeordnet werden. Daher definiert dieser Internet Draft eine generische Notifikation, auf welche alle SNMP-Traps abgebildet werden. Diese generischen Notifikationen sollen von einer Instanz der „internetSystem" Managementobjekt-Klasse ausgesendet werden (für weitere Details siehe [IIMCIMIBTR], bzw. 4.4.3 oder 5.2.4).

- Erstellen von NAME BINDING Templates:
  Für jede erstellte ISO/ITU Klasse muß zusätzlich ein NAME BINDING Template definiert werden. Dieses NAME BINDING Template legt die Lage einer Instanz dieser Klasse in der Containment-Hierarchie fest. Dabei dient die aus der übergeordneten SNMP-Gruppe entstandene ISO/ITU Klassse als *SUPERIOR OBJECT CLASS*. Es wird also der Internet-Registrierungsbaum auf die OSI-Containment-Hierarchie abgebildet:
  Übersetzte OSI-Klasse „internetSystem" mit der übergeordneten SNMP-MIB-2-Gruppe „mib-2" bzw. die diese repräsentierende OSI-Klasse „mib2":

```
internetSystem-mib2NB  NAME BINDING
        SUBORDINATE OBJECT CLASS
           internetSystem  AND SUBCLASSES;
        NAMED BY SUPERIOR OBJECT CLASS
           mib2;
        WITH ATTRIBUTE internetSystemId;
        BEHAVIOUR
           internetSystem-systemNBBehaviour BEHAVIOUR
           DEFINED AS
           !BEGINPARSE
           INDEX     NULL;
           ENDPARSE!;;
        REGISTERED AS
           {1 2 124 360501 14 4 1 3 6 1 2 1 1 };
```

3.1.2 Der MIB-Compiler

Vom *Telecommunications Laboratory* des *Technical Research Centre of Finland (VTT/TEL)* wurde ein Compiler für die Übersetzung von Internet SNMP MIBs zu OSI GDMO MIBs entwickelt. Dieser „SMIC2GDMO"-Compiler ist eine Erweiterung des von *Dave Perkins* von *SynOptics* entwickelten SNMP MIB Compilers „SMIC". Der Übersetzungsalgorithmus basiert dabei auf den Grundlagen des von

der IIMC veröffentlichten Internet Drafts, wie er im vorigen Abschnitt kurz eingeführt wurde.
Der *SMIC2GDMO*-Compiler erwartet als Eingabe eine SNMPv1-MIB und liefert als Ausgabe die GDMO-Darstellung dieser Eingabe-MIB, siehe Abbildung 3.1.
Dabei muß beachtet werden, daß für jede neue SNMPv1-MIB, die mit diesem Compiler übersetzt werden soll, alle in dieser MIB enthaltenen SNMP-Gruppen in eine oder mehrere Tabellen (zum Beispiel „pszSnmpV1mib_2BuildinGroups") in der Datei „smdump_gdmo.ic" eingetragen werden müssen (siehe [Reilly]). Anschließend muß der gesamte Compiler neu übersetzt und gebunden werden, damit die Änderungen gültig werden.
Weiterhin soll darauf hingewiesen werden, daß die entstandene GDMO-MIB eventuell nachbearbeitet werden muß (siehe dazu [Reilly93]). So erhalten zum Beispiel alle *NAME BINDINGs* der Managementobjekt-Klassen, die aus SNMP-Gruppen entstanden sind, als *SUPERIOR OBJECT CLASS* die OSI-Klasse „system". Dies muß für jede Klasse nachträglich verändert werden, wenn die tatsächliche *SUPERIOR OBJECT CLASS* nicht der Klasse „system" entspricht, da sonst keine Instanzen dieser Klasse kreiert werden können (da die NAME BINDING-Restriktionen verletzt werden). So wurde in dem gerade vorgestellten „internetSystem-mib2NB" *NAME BINDING* die ursprünglich vom Compiler falsch generierte *NAMED BY SUPERIOR OBJECTCLASS* ISO/ITU Klasse „system" mit der richtigen ISO/ITU Klasse „mib2" ausgetauscht.
Wie in Abbildung 3.1 ersichtlich ist, dienen die aus dieser Übersetzung entstandenen GDMO-Templates und ASN.1-Typdefinitionen als Eingabe für den Agenten-Entwicklungsprozesses, wie er in 3.2.3 beschrieben wird. Dort wird diese neue GDMO-MIB mit dem *Managed Object Compiler* auf die Konformität zu dem OSI-Informationsmodell überprüft. Dies erscheint eigentlich als ein überflüssiger Test, denn der *SMIC2GDMO*-Compiler übersetzt definitionsgemäß SNMP-MIBs in OSI-SMI-konforme MIBs. Dieser Schritt wird aber dennoch benötigt, da der *SMIC2GDMO*-Compiler an einigen Stellen der GDMO-Templates Parameter einsetzt, die der *Managed Object Compiler* nicht auflösen und damit der Entwicklungsprozeß nicht fortgesetzt werden kann:
So übersetzt der *SMIC2GDMO*-Compiler zum Beispiel die SNMP-Variable „sysContact", siehe dazu auch den vorherigen Abschnitt, in ein GDMO-Attribut. Das Ergebnis lautet folgendermaßen, wobei nur für diese Betrachtung relevante Teile dargestellt werden:

```
sysContact ATTRIBUTE
        WITH ATTRIBUTE SYNTAX
                IIMCRFC12131354ASN1.DisplayString;
        BEHAVIOUR
                [...]
        REGISTERED AS
           { iimcAutoObjAndAttr 1 3 6 1 2 1 1 4};
```

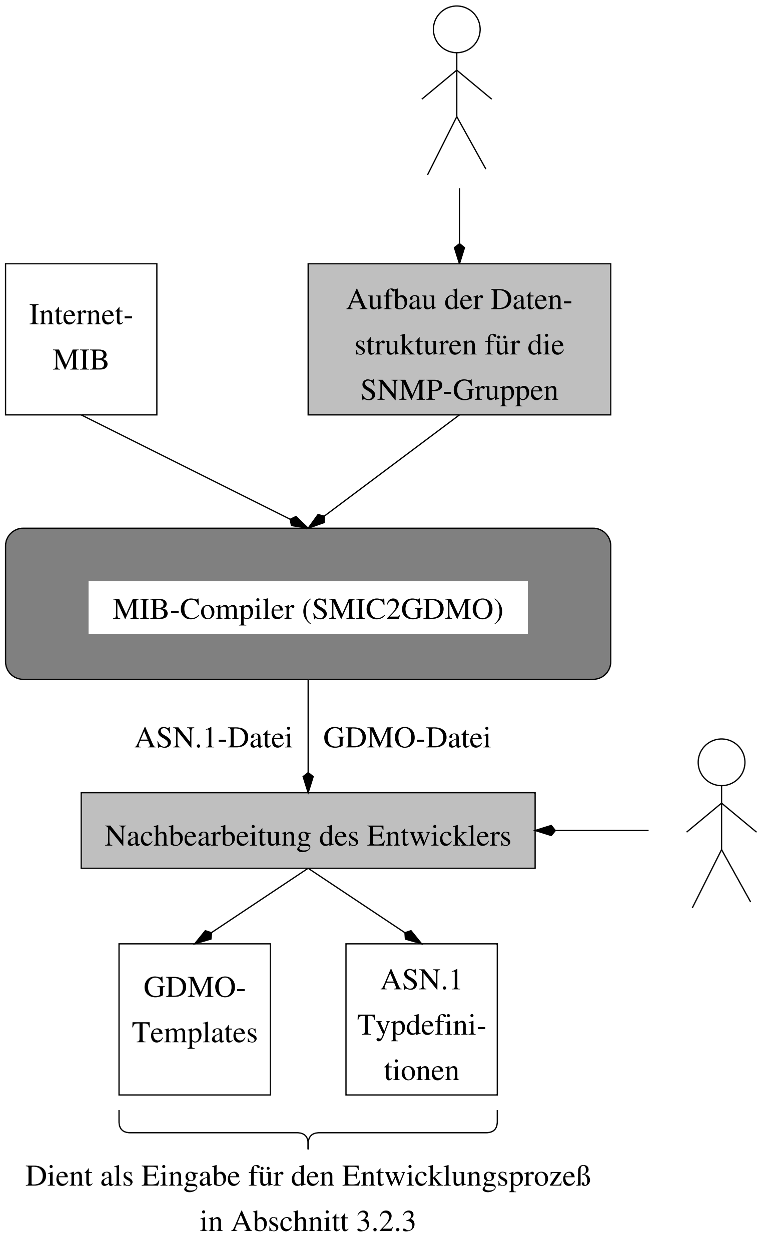


Abbildung 3.1: Der Ablauf einer Übersetzung von einer Internet-MIB zu einer GDMO-MIB.

Dieses GDMO-Template kann der *Managed Object Compiler* nicht fehlerfrei übersetzen, da er den Parameter „iimcAutoObjAndAttr“ nicht auflösen kann. Dieser wird zwar in einer ASN.1-Datei richtig definiert, muß aber nachträglich vom Entwickler ausgebessert werden:

```
sysContact ATTRIBUTE
        WITH ATTRIBUTE SYNTAX
                IIMCRFC12131354ASN1.DisplayString;
        BEHAVIOUR
                [...]
        REGISTERED AS
           { 1 2 124 360501 14 1 1 3 6 1 2 1 1 4};
```

3.2 IBM TMN WorkBench for AIX

Die *IBM TMN WorkBench for AIX* besteht aus einer Gruppe von Werkzeugen und Diensten, die das Entwickeln von TMN-Managementlösungen unterstützen. Für diese Diplomarbeit waren dabei folgende von Bedeutung:

1. **Information Modeling**
   Mit Hilfe der *IBM TMN WorkBench for AIX* ist es möglich, die zu überwachende, reale Ressource nach dem TMN- bzw. OSI-Informationsmodell zu modellieren. Diese Werkzeuge stellen Dienste bereit, mit denen die Realisierung von TMN-konformen Management Information Bases (MIBs) unterstützt wird.

   - **Managed Object Compiler**
     Bei dem *Managed Object Compiler* handelt es sich um eine der zentralen Komponenten der *IBM TMN WorkBench for AIX*. Die nach den OSI- bzw. Internet Informationsmodell gegebenen Definitionen werden geparst und lexikographisch analysiert. Dabei wird ein azyklischer Graph erzeugt, der das Informationsmodell im Speicher repräsentiert. Diese Vorgehensweise stellt einen schnellen und flexiblen Zugriff auf die GDMO- bzw. ASN.1-Informationen für alle Komponenten der *IBM TMN WorkBench for AIX* bereit.

   - **Metadata Database and Metadata Services API**
     Metadata bestehen aus Definitionen von Managementobjekt-Klassen. Diese *Metadata* werden in der *Metadata Database*, einer Shared Libary, verwaltet. Die *Metadata Database* bietet on-line Zugriff auf die gesamte Menge aller Definitionen, auf welchen Applikationen basieren können. Die *Metadata Services API* stellt eine Programmierschnittstelle für den Zugriff auf die *Metadata Database* bereit.

2. **Managed Object Browser and Editor**
   Ein weiteres Werkzeug für das Arbeiten mit großen und damit unübersichtlichen Objektbeschreibungen (GDMO-Templates und ASN.1-Typdefinitionen) ist der *Managed Object Browser and Editor*. Er verarbeitet einerseits das Ergebnis des *Managed Object Compiler* und andererseits die Metadata. Damit erlaubt er eine Darstellung und Bearbeitung von allen Managementobjekt-Klassen mit deren Attributen und Beziehungen. So kann zum Beispiel ein Dokument mit Managementobjekt-Klassendefinitionen aus der *Metadata Database* in den *Managed Object Browser and Editor* geladen werden. Anschließend kann dieses Dokument beliebig verändert werden. Diese Veränderungen werden mit Hilfe des *Managed Object Compiler* auf die Konformität zu dem OSI-Informationsmodell überprüft. Der *Managed Object Compiler* gibt jeden dabei auftretenden Fehler aus. Zuletzt kann das

veränderte, fehlerfreie Dokument als neues, eigenständiges Dokument in der *Metadata Database* abgespeichert werden.

3. **TMN Agent Application Development**
Diese Entwicklungsumbegung erleichtert den Entwurf und die Implementierung von TMN-konformen OSI-Agenten. Sie ist für die Realisierung eines CMIP/SNMP Gateways von entscheidender Bedeutung und soll deswegen im nächsten Abschnitt ausführlich vorgestellt werden.

3.2.1 Ein Toolkit für die Entwicklung von OSI-Agenten

Die in diesem Abschnitt vorgestellte Agenten-Entwicklungsumgebung umfaßt Tools, Schnittstellen und Bibliotheks-Module zur eigenständigen Realisierung eines TMN-konformen OSI-Agenten. Dieses Toolkit ist in eine Agentenarchitektur eingebettet, die festlegt, wie sowohl die Komponenten der Entwicklungsumgebung als auch Benutzer-spezifische Anforderungen in den Agenten integriert werden.
Eine Übersicht über die Struktur dieser Agentenarchitektur gibt Abbildung 3.2 aus [FeHeNi95].
Der Agent besteht aus einer Menge miteinander kommunizierender Komponenten. Diese können in drei große Kategorien eingeteilt werden :

- **Infrastruktur**
Die *Infrastruktur* ermöglicht die Basisfunktionalität des Agenten:
  - Das *Common Management Interface Service Element (CMISE)* stellt die Unterstützung des CMIP-Protokolls bereit.
  - Das *Assotiation Control Service Element (ACSE)* erlaubt Verbindungen zwischen dem Agenten und anderen Agenten bzw. Managern aufzubauen und zu erhalten.
  - Der *Directory Service* lokalisiert andere Agenten bzw. Manager.
- **Allgemeiner Agententeil**
Der allgemeine Agententeil besteht aus Komponenten, die alle Agenten gemeinsam haben, so zum Beispiel für das Erhalten der Containment-Hierarchie, für das Weiterleiten von CMIS-Requests an die jeweiligen Instanzen und das Koordinieren der Antworten von diesen Instanzen oder für das Ausführen von weiteren CMIS-Requests durch Instanzen. Weitere Aufgaben sind das Abspeichern von *Log Records* oder das Weiterleiten oder Bearbeiten von *Event Notifications.*
- **Spezifischer Agententeil**
Dieser Teil wird zum einen vom Agenten-Entwickler generiert, indem er GDMO- und ASN.1-Dokumente erstellt, zum anderen vom *MIBcomposer* bereitgestellt.

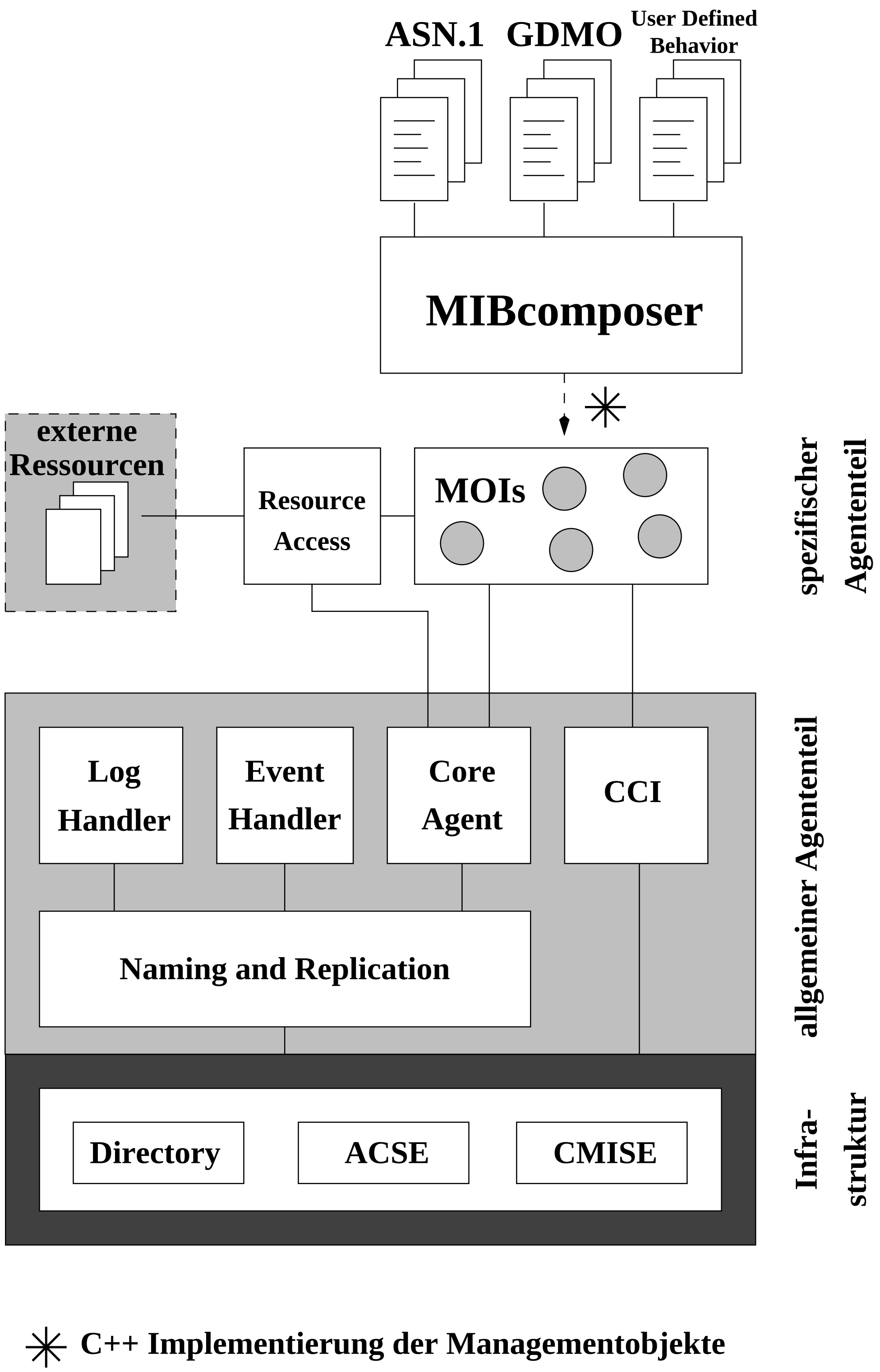


Abbildung 3.2: Die Architektur eines TMN- bzw. OSI-Agenten in der WorkBench

Die Entwicklungsumgebung besteht aus der *Infratruktur*, dem *allgemeinen Agententeil* und dem *MIBcomposer*. Der *MIBcomposer* generiert C++-Implementierungen zum einen von allen Managementobjekt-Klassen, die in der MIB des Agenten vorkommen können (wird in GDMO- und ASN.1-Dokumenten definiert) und zum anderen von Benutzer-spezifischen Verhalten dieser Managementobjekt-Klassen.
In den nächsten Abschnitten werden die einzelnen Komponenten im Detail vorgestellt.

Allgemeiner Agententeil

Der *allgemeine Agententeil* besteht einerseits aus der *Naming and Replication*- und andererseits aus der *Core Agent*-Komponente. Weiterhin existieren noch zwei Funktionsblöcke für das Bearbeiten von *Logs* und *Event Notifications*.

Naming and Replication Die *Naming and Replication*-Komponente speichert unter anderen die Enthaltenseinshierarchie der Instanzen von Managementobjekten. Die Struktur dieser Hierarchie wird durch eine oder mehrere *NAME BINDING*-Beziehung(en) für jede Klasse in den *Guidelines for the Definition of Managed Objects (GDMO)*-Spezifikationen beschrieben. Sobald eine Instanz eines Managementobjekts im Agenten kreiert werden soll, muß nicht nur diese Instanz im Agenten gehalten werden, sondern auch ihre Enthaltenseinsbeziehung. Sie werden vor allem dazu genutzt, um die Instanzen herauszufinden, die in den Scope einer CMIP-Anforderung (m-Set, m-Get, m-Action, m-Delete) fallen. Anschließend werden „Kopien" des originalen Requests an alle diese (durch den Scope ausgewählte) Instanzen weitergeleitet („Replication"). Es fällt also nicht in den Aufgabenbereich eines Agenten-Entwicklers, Scopes zu bearbeiten, dies wird von der *Naming and Replication*-Komponente übernommen.
Die *Naming and Replication*-Komponente empfängt Anforderungen von der Infrastruktur und führt folgende Funktionen aus:

- Create-Request : Überprüfung, ob die angegebene Instanz mit dem angegebenen Namen kreiert werden kann, das heißt, ob der *Distinguished Name (DN)* den *NAME BINDING*-Restriktionen entspricht. Außerdem wird überprüft, ob die Instanz schon existiert, um Duplikate zu vermeiden. Sobald eine Create-Anforderung erfolgreich ausgeführt wurde, wird die neue Instanz in die Containment-Hierarchie aufgenommen.

- Delete-Request : Überprüfung, ob diese Instanz in Bezug auf die *NAME BINDING*-Restriktionen gelöscht werden kann. Außerdem wird überprüft, ob diese Instanz überhaupt existiert, bevor sie gelöscht werden soll. Weiterhin muß vor dem Löschen dieser Instanz getestet werden, ob auch der eventuell existierende *Subtree* in der Containment-Hierarchie gelöscht werden kann.

- Für alle Requests : Einerseits wird das *Scoping* ausgeführt, das heißt, es wird festgestellt, welche Instanz eines Managementobjekts durch den im Scope-Parameter der CMIP-PDU angegebenen Wert bezeichnet werden, und andererseits wird überprüft, ob diese Instanzen existieren. Falls für eine Instanz beides zutrifft, erhält sie eine „Kopie“ der Anforderung („Replication“).

Core Agent Der Core Agent stellt eine Anzahl von Diensten zur Verfügung, die alle Agentenimplementierungen gemeinsam haben. Beispiele für diese Dienste sind: das Verwalten und Weiterleiten der ankommenden Anforderungen von der *Naming and Replication*-Komponente, Instanzen im *spezifischen Agententeil* zu lokalisieren und den Zugriff von Instanzen auf externe Resourcen zu ermöglichen. Der Agent kann *single-* oder *multithreaded* arbeiten. Im *multithreaded*-Modus werden ankommende CMIP-Requests simultan bearbeitet, das heißt, jeder einzelne Request wird in einem eigenen Thread ausgeführt, wobei zusätzlich sichergestellt wird, daß dabei keine Blockaden (Deadlocks) entstehen können. So ist es immer nur einem Request erlaubt, in einer Instanz „aktiv“ zu sein. Sobald ein Request ankommt, wird überprüft, ob die angesprochene Instanz „busy“ ist, das heißt, ob bereits ein anderer Request auf dieser Instanz arbeitet. Falls dies nicht der Fall ist, sperrt der Request diese Instanz und startet mit der Bearbeitung. Anschließend wird die Sperre aufgehoben, so daß anderen Requests die Möglichkeit gegeben wird, ebenfalls diese Instanz zu beanspruchen.
Um diesen *multithreaded*-Modus zu ermöglichen bzw. zu verwalten sind zusätzliche Komponenten erforderlich:

- **Object Table**
  Die *Object Table* enthält für jede im Agenten existierende Instanz einen Eintrag, der den Zustand der Instanz beschreibt : Die Instanz ist gesperrt oder frei verfügbar.

- **Thread Manager**
  Der *Thread Manager* verwaltet die aktiven Threads. Deadlocks werden dadurch verhindert, da der *Thread Manager* zu jedem Thread die Instanzen speichert, welche er gerade sperrt. Schließlich überwacht er alle Threads, um sicherzustellen, daß eine CMIP-Error-Response zum Manager zurückgeschickt werden kann, falls sich ein Thread vorzeitig beendet und damit der Request nicht vollständig und fehlerfrei bearbeitet wurde.

- **Method Calls**
  Diese Methoden erlauben das Setzen von Sperren und das Freigeben von gesetzten Sperren auf Instanzen.

Der *multithreaded*-Modus basiert auf *IBM DCE Threads for AIX V3.1* (Distributed Computing Environment). Da keine Installation dieses Pakets vorhanden war, wurden keine Thread-Konzepte in diese Diplomarbeit integriert.

Callback CMIS/CMIP Interface Das *Callback CMIS/CMIP Interface (CCI)* erlaubt es Instanzen, CMIS/CMIP-Requests in Callbacks auszulösen. Dies ermöglicht die in der OSI-Architektur geforderte Doppelrolle Manager/Agent. Ein Agent tritt in eine Managerrolle, sobald er mit Hilfe dieses Interfaces CMIS/CMIP-Requests abschickt.
CCI ist ein asynchrones Interface und stellt eine Menge an C++-Methoden zur Verfügung, die den möglichen CMIP-Requests entsprechen.

Log und Event Handler Aufgrund der Spezifikation seiner Klasse kann es einer Instanz möglich sein, *Notifikationen* (=asynchrone Mitteilungen) abzuschicken, wenn bestimmte Ereignisse eintreten. So ist es zum Beispiel denkbar, daß eine Instanz die Notifikation *attributeValueChange* generiert, sobald sich der Wert eines Attributes ändert. Als *Event* werden die Meldungen bezeichnet, weche mit Hilfe des CMIP-Protokolls übertragen werden und dabei aus bestimmten Notifikationen entstanden sind. Es werden jedoch nicht alle von Instanzen von Managementobjekten ausgelösten Notifikationen zu Events und damit zum Manager geschickt. Dies könnte zu einem riesigen Kommunikationsaufkommen führen. Um dieses Kommunikationsaufkommen (Netzlast) zu kontrollieren, werden in [ISO10164-5] Mechanismen beschrieben, die es Managern erlauben, nur diejenigen Ereignisse zu beschreiben, die für sie interessant sind und ihnen gemeldet werden sollen. Damit werden nicht alle von den Managementobjekten generierten Notifikationen zu Events, sondern nur diejenigen, die von einem Manager ausgewählt wurden (siehe Abbildung 3.3): Es existiert im Agenten mindestens ein Managementobjekt, welches beim Auftreten bestimmter Ereignisse Notifikationen auslöst (changeAttributeValue, objectCreation, objectDeletion, ...). Ein lokaler Erkennungs- und Verarbeitungsmechanismus, der nicht der Standardisierung unterworfen ist, nimmt die erzeugten Notifikationen entgegen. Er formt danach potentielle Event-Reports. Diesen können neben den Informationen aus den Notifikationen noch weitere Informationen hinzugefügt werden, so zum Beispiel der Zeitpunkt des Ereignisses oder die Objekt-Klasse bzw. die Instanz des Auslösers. Potentielle Event-Reports werden an alle *Event-Forwarding-Discriminators (EFD)*, die im lokalen System existieren, verteilt. Diese EFDs sind wiederum Managementobjekte. Ein *Diskriminator* legt fest, welche *Event-Reports* weitergeleitet werden sollen. Die Zielinstanzen (=Manager) werden dazu in einer Liste gespeichert. Der *Diskriminator* enthält dabei zum einen einen Filter, welcher festlegt, aus welchen potentiellen Event-Reports schließlich Events aufgebaut werden sollen. Andererseits enthält ein *Diskriminator* einen *Scheduler*, der festlegt, in welchem Intervall überhaupt potentielle Event-Reports bearbeitet werden. Da EFDs selbst Managementobjekte sind, enthalten sie auch Attribute und Status und können somit von einem Manager aus gesteuert und überwacht werden. Wie andere Managementobjekte auch, können die EFDs selbst Notifikationen generieren. Diese werden erkannt (von der lokalen Ereignisfeststellung

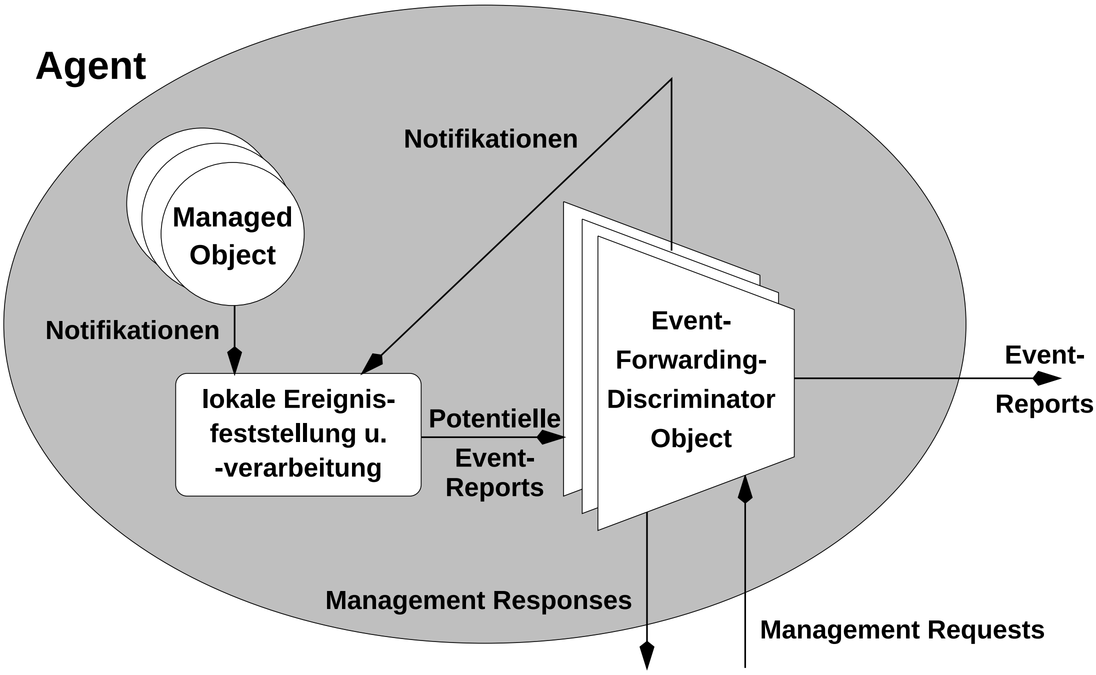


Abbildung 3.3: Aufbau der Event Report Management Funktion

und -verarbeitung) und als potentielle Event-Reports allen EFDs zur Bearbeitung übergeben, damit auch wieder dem EFD, der diese Notifikation ursprünglich erzeugte.
Es können zwei Typen von Ereignismeldungen im Agenten auftreten: Notifikationen und Events. Das Log-Management ist nun dafür verantwortlich, daß diese Ereignismeldungen dauerhaft abgespeichert werden können, siehe Abbildung 3.4.
Ein *Log* [ISO10164-6] ist eine dauerhafte Ablage von Mangementinformationen, die in *Log Records* organisiert werden. Diese Informationen können entweder von empfangenen Events oder von internen Ereignissen stammen. Die internen Ereignisse werden, wie auch bei der *Event-Report-Management Function* [ISO10164-5], von Notifikationen repräsentiert, die jeweils von Managementobjekten ausgelöst werden. Ein lokaler Erkennungs- und Verarbeitungsmechanismus nimmt die Notifikationen entgegen. Er formt anschließend potentielle Log Reports und verteilt diese an alle existierenden *Log*-Instanzen. Alle von den *Log*-Instanzen empfangenen Reports (Event-Reports oder potentielle Log Reports) werden anhand eines Filters überprüft, ob sie abgespeichert werden sollen oder nicht. Bei diesem Filter liegt der gleiche Mechanismus zugrunde, wie er schon bei der *Event-Report-Management Function* beschrieben wurde.

Der *Event Handler* ist eine Implementierung der *eventForwardingDiscriminator*-Klasse. Alle CMIS/P-Anforderungen, die den EventForwardingDiscriminator (EFD) betreffen, werden zu dem Event Handler geleitet und von diesem bearbei-

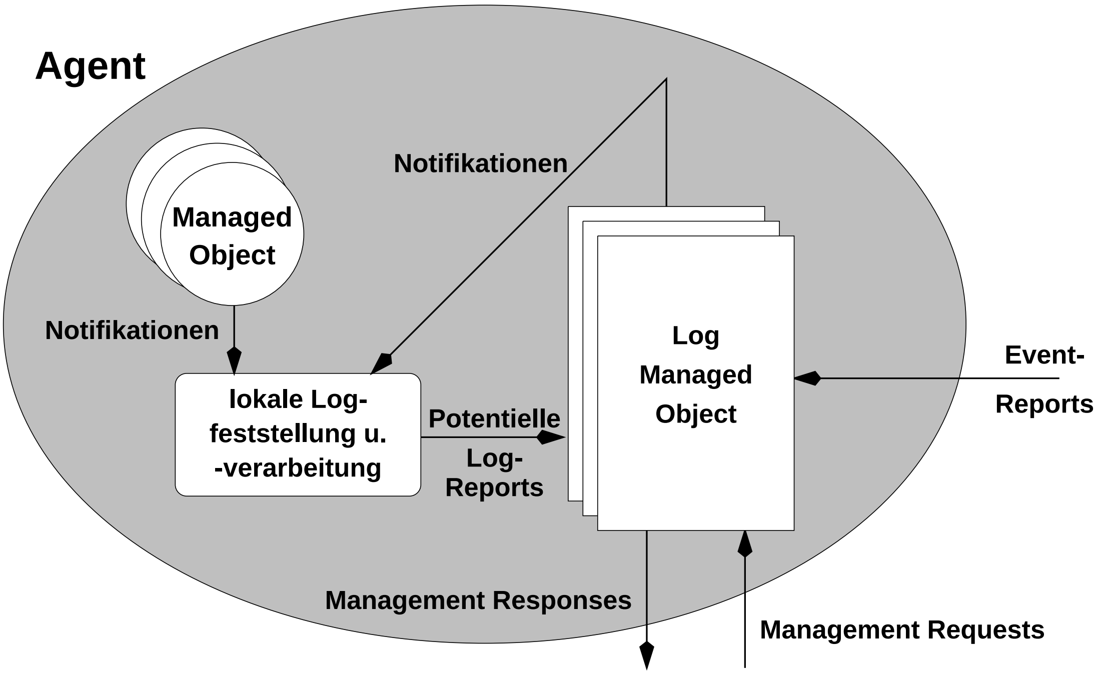


Abbildung 3.4: Elemente der Log Control Function

tet. EFD Instanzen werden statt im *spezifischen Agententeil* in dieser Komponente kreiert. Alle Notifikationen, egal, ob sie von Instanzen von Managementobjekten, vom Log Handler oder vom Event Handler generiert wurden, werden zu dieser Komponente weitergeleitet. Hier wird anhand des EFD-eigenen Filters geprüft, ob die Notifikation als eine CMIP-Event-Notification weitergeleitet werden soll.
Ebenso werden alle CMIS/P-Anforderungen, die die *log*-Klasse betreffen, zum *Log Handler* weitergeleitet. Alle Notifikatioen (von Instanzen von Managementobjekten, von dem Event Handler oder von dem Log Handler) werden zum Log Handler weitergeleitet. Dort wird anhand des Log Handler-eigenen Filters überprüft, ob diese Notifikation gespeichert werden soll oder nicht. Im dem Fall, daß die Notifikation abgespeichert werden soll, wird eine Instanz der Klasse *logRecord* generiert. Diese kann sowohl im Speicher existieren, als auch in eine Datenbank eingetragen werden.

Spezifischer Agententeil

Dieser Teil wird zum einen vom Agenten-Entwickler generiert, indem er GDMO und ASN.1 Dokumente erstellt, zum anderen vom *MIBcomposer*, siehe nächsten Teilabschnitt, bereitgestellt. Es sind diejenigen Komponenten eines Agenten, die ihn von anderen Agenten unterscheiden.

Instanzen von Managementobjekten Der *spezifische Agententeil* enthält Instanzen von C++-Klassen, welche vom *MIBcomposer* generiert wurden. Es können sich dabei um Instanzen handeln, die Managementobjekt-Klassen und deren Attribute repräsentieren, oder aber um Instanzen beliebiger C++-Klassen. Jede Klassen- und Attributimplementierung enthält *Benutzer-Callbacks* (siehe nächsten Teilabschnitt). Diese Callbacks erlauben den Zugriff auf alle Daten innerhalb dieser Instanz, aber sie ermöglichen auch einen Zugriff auf Instanzen anderer Klassen, sogar auf Instanzen anderer Agenten (CCI). Damit können sich Instanzen von Managementobjekt-Klassen wie Manager verhalten.

Resource Access Um einem OSI-Manager auch einen Zugriff auf nicht-OSI-Agenten (externe Ressourcen) zu ermöglichen, ist es notwendig, daß ein OSI-Agent als Übersetzer zwischen den beiden unterschiedlichen Kommunikationsarten fungieren kann. Dieser OSI-Agent stellt somit eine *Q Adapter Function* im TMN-Referenzmodell dar. Die Kommunikation des OSI-Agenten mit einem nicht-OSI-Agenten läuft dabei über das *m-Interface* ab. Die *Resource Access*-Komponente bietet diese Kommunikationsschnittstelle:

- Die Kommunikation zwischen einer Instanz eines Mangementobjekts und einer externen Ressource kann von beiden Seiten initiiert werden.
- Anforderungen und Antworten von der externen Ressource können in der *Resource Access*-Komponente weitergeleitet oder ausgeführt werden.

MIBcomposer

Die Aufgabe des *MIBcomposers* ist es, C++-Implementierungen der Managementobjekte für den Agenten zu generieren:

1. Der *MIBcomposer* generiert den C++-Code für die Managementobjekt-Klassen und die Attribute, die der Agent schließlich verwalten soll. Dieser automatisch generierte Code unterstützt dabei folgende CMIS-Funktionalitäten:

   (a) CMIS/CMIP Filteroperationen

   (b) Allomorphismus, das heißt, eine Instanz einer Managementobjekt-Klasse kann sich genau so verhalten, als wäre sie eine Instanz einer anderen Managementobjekt-Klasse

   (c) Laufzeitüberprüfung für das Einbinden von *conditional packages*

   (d) selbständige Namensgebung für die Instanzen.

   Aufgrund dieser Eigenschaften ist es möglich, einen kompletten und eigenständigen TMN-konformen Agenten zu erstellen. Dies erfolgt ohne irgendwelche Eingaben oder explizite Informationen eines Entwicklers, außer

der Definition, welche Klassen dieser Agent unterstützen soll (entspricht den ASN.1- und GDMO-Dokumenten in Abbildung 3.2). Dieser neue, „sterile“ Agent besitzt noch keine Anbindung an eine zu verwaltende Ressource. Er stellt also eine Containment-Hierarchie zur Verfügung, auf der die vollständige CMIS-Mächtigkeit ausführbar ist, aber bei dem die Klassen/Instanzen und Attribute keine Funktionalität ausführen. Ein derartiger Agent eignet sich zum Beispiel für „Rapid Prototyping“ und damit für eine sukzessive Weiterentwicklung bis schließlich hin zu einem fertigen Produkt.
Sobald nun das Verhalten der Klassen und Attribute vom Entwickler implementiert wird (entspricht dem User defined Behavior in Abbildung 3.2), zum Beispiel durch einen mit einer Ressource kommunizierenden Code, entsteht ein fertiger, voll funktionsfähiger TMN-/OSI-Agent.

2. Diese Kodierung des Verhaltens von Klassen und Attributen erfolgt auf eine einfache und intuitive Weise. So existieren 20 verschiedene *Callbacks*, die als verschiedene Arten von Fragestellungen angesehen werden können. Ein Entwickler muß nun bei der Bearbeitung eines CMIP-Requests eine feste Anzahl und vordefinierte Reihenfolge dieser *Callbacks*/Fragestellungen implementieren/beantworten. So wird zum Beispiel während der Ausführung eines CMIP-GET-Requests zuerst ein *Callback* mit dem Namen *„Precondition Check“* aufgerufen. Dieser überprüft vor dem Zugriff auf den Wert des Attributes eventuelle Vorbedingungen. Anschließend wird über den *„Real Value Access“-Callback* auf den Wert des Attributes zugegriffen. Dabei kann es sich um einen einfachen Variablenzugriff handeln, es kann aber auch ein komplexer Zugriff auf eine darunterliegende Ressource sein. Bei Bedarf können vom Agenten auch weitere CMIP-Requests ausgelöst werden, um den Wert des Attributs zu bestimmen. Zuletzt werden in dem *„Postcondition Check“-Callback* eventuelle Aufgaben nach einem Attributzugriff ausgeführt.
Bei der Implementierung einer Klasse werden automatisch alle zu dieser Klasse gehörenden und die von den Vaterklassen geerbten Attribute dargestellt. So wird in einer komplexen Vererbungshierarchie von einer Vielzahl an Klassen und Attributen immer nur der Ausschnitt gezeigt, der für die gerade aktuelle Bearbeitung wirklich notwendig ist.

3. Durch dieses *Callback*-Konzept werden implementierungsspezifische Details von Managementobjekt-Klassen und Attributen von der Standard-Bearbeitung getrennt. Die jeweiligen *Callbacks* werden in eigenen Dateien verwaltet. Auf diese Weise können, ohne den bisherigen *Callback*-Code zu verändern, neue Klassen und damit Attribute dem Agenten hinzugefügt werden[5].

Sobald der *MIBcomposer* gestartet wird, muß der Benutzer die *„Metadata“* (GDMO- und ASN.1) Dateien bestimmen, die geladen werden sollen. Damit wird

[5]Der Agent muß aber danach neu kompiliert werden.

festgelegt, welche Managementobjekt-Klassen der Agent unterstützen soll.
Weiterhin ist der Name eines Verzeichnisses notwendig („workspace"), in welchem die implementierungsspezifischen Details enthalten sind bzw. in welchem sie abgespeichert werden sollen. Nach dem Parsen der Eingabe-Dokumente kann der Entwickler die *Callbacks* für die Klassen und Attribute implementieren. Anschließend generiert der *MIBcomposer* Konfigurationsdateien (z.B. ein Makefile) und den Code für die Klassen und Attribute, indem er die *Metadata*-Informationen und den Code in dem *workspace*-Direktory geeignet verbindet. Das Resultat ist die vollständige C++-Implementierung aller Managementobjekt-Klassen. Damit ist der Agent konform zu der oben durch die GDMO- und ASN.1-Dateien definierten MIB.

Weitere Eigenschaften

In diesem Abschnitt werden weiterführende Funktionen des Agenten beschrieben:

Persistenz Ein OSI-Agent kann Instanzen von Managementobjekten verwalten, die kritische Daten enthalten und die für eine erfolgreiche Bearbeitung von Requests notwendig sind. Derartige kritische Daten erfordern eine dauerhafte Speicherung, damit sie immer zugänglich sind, selbst nach einem Systemausfall oder einem Zusammenbruch des Agenten. Dieser persistente Datensatz wird dazu in der objektorientierten Datenbank *ObjectStore* gehalten.

3.2.2 Die Prozeßstruktur eines Agenten

Nachdem in dem vorherigen Abschnitt die Agenten-Entwicklungsumgebung und damit die Architektur eines TMN- bzw. OSI-Agenten in der *IBM TMN WorkBench for AIX* vorgestellt wurde, sollen in diesem Unterkapitel die Zuordnungen von den Komponenten der Agentenarchitektur zu real existierenden Prozessen erfolgen, siehe dazu Abbildung 3.5.
Im folgenden sollen die Aufgabenbereiche der einzelnen Prozesse erläutert werden (siehe auch [IBMPrUG95] ab Seite 1):

- **Infratop**
  Der *Infratop*-Prozeß besteht aus den Komponenten *Infrastruktur* und *Naming and Replication*. Er stellt eine Vielzahl der Funktionen bereit, wie sie im OSI Manager/Agent Framework beschrieben werden. Dazu gehören:

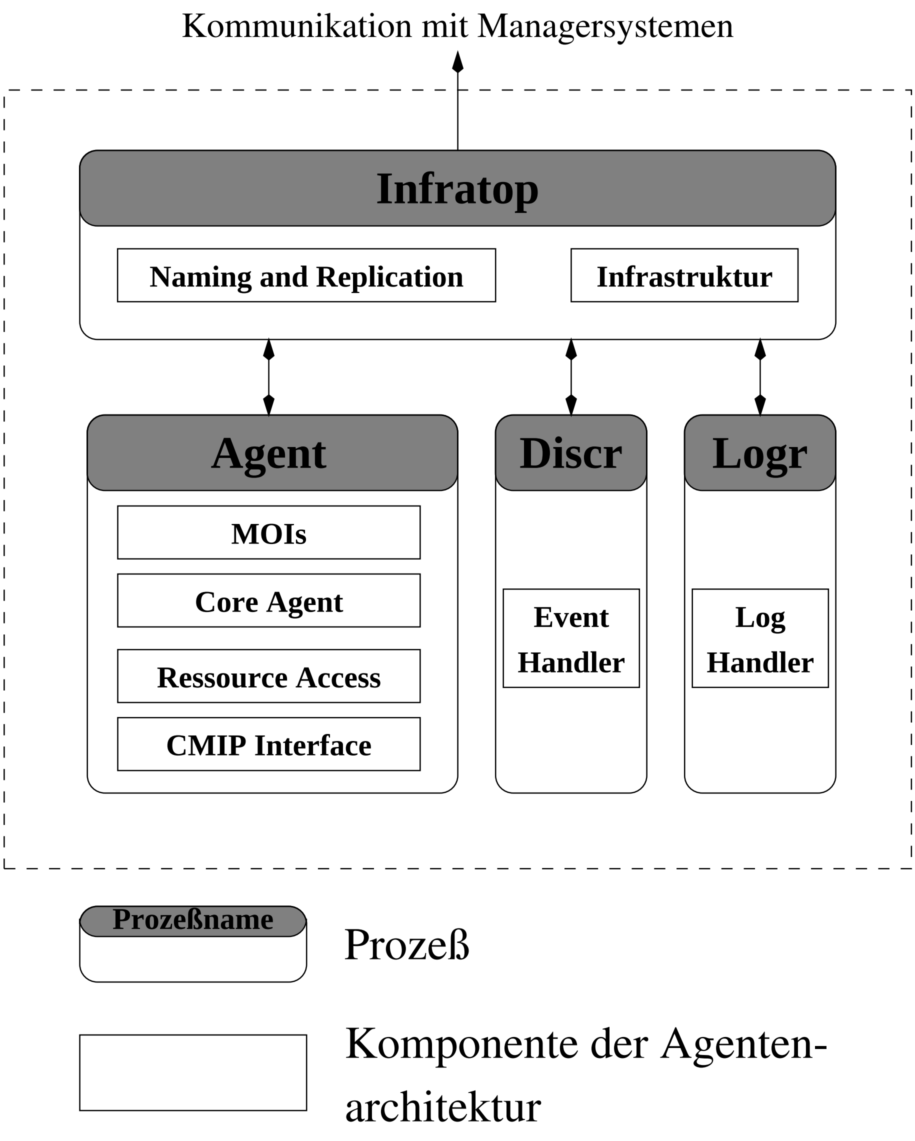


Abbildung 3.5: Die verschiedenen Prozesse eines Agenten

  - Bereitstellen eines Kommunikationsdienstes für den Agenten, das heißt, es werden OSI Schicht 7 Dienst-Instanzen (CMISE, ACSE) zur Verfügung gestellt.
  - Verwalten und Erhalten der Containment-Hierarchie.
  - Automatisches Ausführen von gescopten Requests anhand der Enthalteinshierarchie und Weiterleiten von „Kopien“ der originalen Anforderung an die angesprochenen Instanzen („Replication“).
  - Überprüfen von *NAME BINDING*-Restriktionen bei *CMIP CREATE*- und *DELETE*-Requests.
  - Verbindungen zwischen diesem Agenten und anderen Managern bzw. Agenten aufbauen, erhalten und beenden.

- **Agent**
  Der *Agent* (single- oder multithreaded) setzt sich aus den Komponenten *Core Agent*, *Instanzen von Managementobjekt-Klassen*, *CCI* und *Resource Access* zusammen. Dieser *Agent*-Prozeß stellt die aktuelle Implementierung der Instanzen von Managementobjekten zur Verfügung, die die Systeme und damit die Schnittstellen zu den realen Ressourcen repräsentieren, die überwacht und gesteuert werden sollen. Das *agent*-Binary startet einen singlethreaded, das *agentMT*-Binary einen multithreaded OSI-Agenten.

- **Discr**
  Das *discr*-Programm startet die *Event Handler*-Komponente. Dieser *Event Handler* empfängt Notifikationen und vergleicht diese anschließend mit in dieser Instanz gespeicherten Kriterien. Falls alle Kriterien erfüllt werden, löst der *Event Handler* eine Event-Report-PDU aus.

- **Logr**
  Das *logr*-Programm startet die *Log Handler*-Komponente. Der *Log Handler* ist verantwortlich für das Ausführen folgender Dienste:

  - Das Kreieren bzw. Löschen von *log*- bzw. *logRecord*-Instanzen.
  - Das Bearbeiten von Notifikationen, die von anderen Instanzen im Agenten/Manager nach der OSI-SMF *Log Control* [ISO10164-6] generiert werden.

3.2.3 Der Entwicklungsprozeß eines Agenten

Die Abbildung 3.6 zeigt den Entwicklungsprozeß eines OSI-Agenten mit der *IBM TMN WorkBench for AIX*: Folgende Anmerkungen sollen den Ablauf der Entwicklung kommentieren:

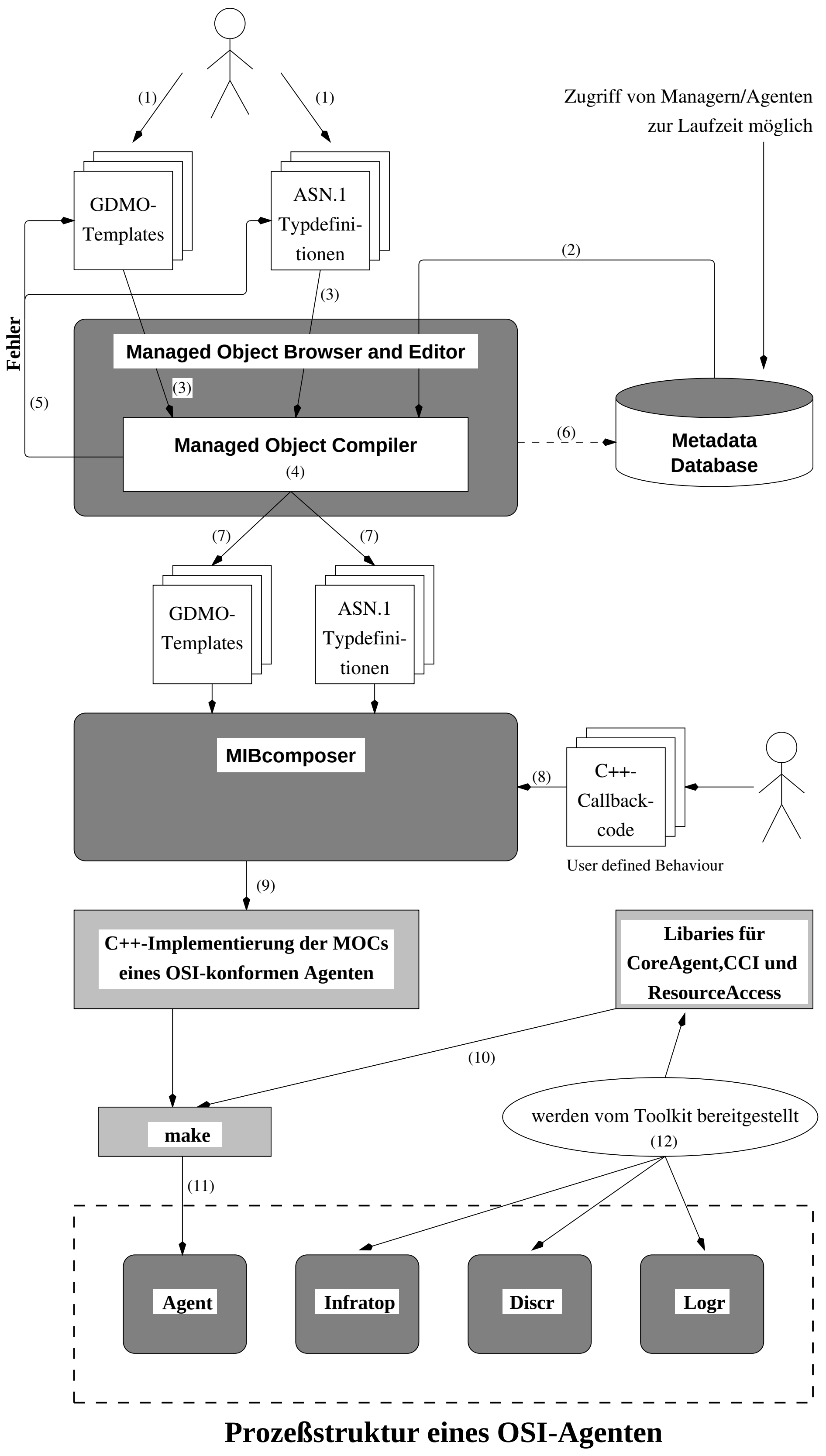


Abbildung 3.6: Der Ablauf des Entwicklungsprozesses eines OSI-Agenten mit der *IBM TMN WorkBench for AIX*

1. Der Agenten-Entwickler entwirft Managementobjekte und definiert diese anschließend mit Hilfe von GDMO-Templates. Die dazu notwendigen ASN.1-Typdefinitionen werden getrennt von den Templates gespeichert.

2. Schon bestehende GDMO-Templates bzw. ASN.1-Typdefinitionen können wiederverwendet werden, indem sie aus der *Metadata Database* gelesen werden.

3. Der *Managed Object Browser and Editor* erlaubt das Arbeiten mit großen und unübersichtlichen Dokumenten (GDMO-, ASN.1-Dateien bzw. Metadata).

4. Diese Dokumente können vom *Managed Object Browser and Editor* aus mit Hilfe des *Managed Object Compilers* auf die Konformität zum OSI-Informationsmodell überprüft werden.

5. Falls dabei ein Fehler auftritt, muß dieser anschließend in den Dokumenten verbessert werden, bevor der Prozeß fortgesetzt werden kann.

6. Sobald ein neues, fehlerfreies Dokument entstanden ist, kann es in die *Metadata Database* eingetragen werden. Damit steht dieses Dokument zur Wiederverwendung in anderen Dokumenten bereit.

7. Die fehlerfreien GDMO-Templates bzw. ASN.1-Typdefinitionen stehen nun als Eingabe für den *MIBcomposer* bereit. Dieser kann daraus die C++-Implementierungen der Managementobjekte aus diesen Dateien erstellen.

8. Ohne weitere Eingaben kann jetzt der *MIBcomposer* bereits den C++-Code der Managementobjekt-Klassen eines OSI-Agenten erstellen. Dieser hat, wie schon oben erwähnt, noch keine Anbindung an eine reale Ressource.
   Der Agenten-Entwickler implementiert das Verhalten von Managementobjekt-Klassen und deren Attribute mittels *Callbacks*.

9. Der *MIBcomposer* generiert aus den GDMO-Templates, den ASN.1-Typdefinitionen und dem Callback-Code (dem Verhalten der Klassen und Attribute) die C++-Implementierung der Managementobjekt-Klassen.

10. Für einen vollständigen OSI-Agenten werden noch die C++-Bibliotheken der anderen Komponenten (Core Agent, CCI und Resource Access) benötigt, um schließlich den *Agent-Prozeß* aufbauen zu können. Diese werden von der Entwicklungsumgebung bereitgestellt (siehe 3.2.1).

11. Im letzten Entwicklungsschritt wird der *Agent*-Prozeß übersetzt und gebunden.

12. Die restlichen benötigten Prozesse werden von der Laufzeitumgebung bereitgestellt. Diese vier Prozesse (Agent, Infratop, Discr und Logr) bilden schließlich zusammen einen OSI-konformen Agenten.

Die Funktionsweise und das Zusammenspiel dieser Prozesse soll im nächsten Abschnitt erfolgen.
Eine Reihe von Werkzeugen erlaubt eine teilweise Automatisierung dieses Entwicklungsprozesses (siehe Abbildung 3.7):

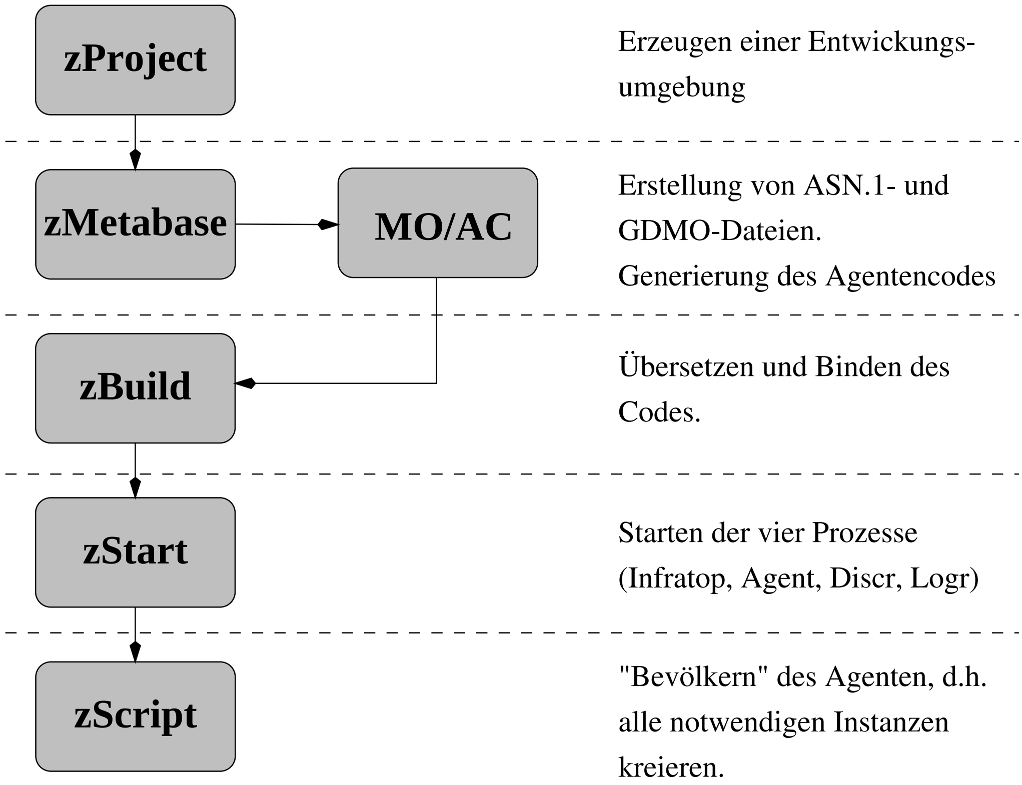


Abbildung 3.7: Die verschiedenen Phasen bei der Entwicklung eines Agenten

3.2.4 Die Funktionsweise des Agenten

Wie schon mehrmals in diesem Kapitel erwähnt, stellen die Komponenten, aus denen sich die Agentenarchitektur zusammensetzt, auf der einen Seite schon eine Vielzahl von Funktionen bereit, auf der anderen Seite muß der Entwickler aber den Code implementieren, der schließlich für den Zugriff auf die zu überwachende und zu steuernde Ressource verantwortlich ist. In diesem Abschnitt soll dargestellt werden, wie die verschiedenen Komponenten miteinander kooperieren, um die Gesamtfunktionalität eines TMN- bzw. OSI-Agenten zu ermöglichen.
Der Start eines Agenten mit dem Befehl *zStart* bewirkt, daß alle für den Agenten

notwendigen Prozesse (Infratop, Agent, Logr und Discr) in den Speicher geladen werden. Die Containment-Hierarchie in der *Naming and Replication*-Komponente, also im *Infratop*, ist noch leer. Ebenso existieren im *Agent*-Prozeß noch keine Instanzen von Managementobjekt-Klassen. Der Agent kann nun automatisch „bevölkert" werden, das heißt, alle zum Betrieb des Agenten notwendigen Instanzen von Managementobjekt-Klassen können kreiert werden.
Das Kreieren einer Instanz einer Managementobjekt-Klasse bewirkt folgendes:

1. Die *Naming and Replication*-Komponente überprüft, ob diese angegebene Instanz mit diesem *Relative Distinguished Name* in Bezug auf die existierenden NAME BINDINGs kreiert werden kann.

2. Sobald Punkt 1 erfüllt ist, wird die Instanz im *spezifischen Agententeil* des Agenten kreiert und sowohl im *Core Agent* (für eine spätere Lokalisierung im *spezifischen Agententeil*) als auch in der Containment-Hierarchie (für die Auflösung von Scopes und die Überprüfung von NAME BINDINGs) im *Infratop* registriert.

Erst nachdem diese neue Instanz in beiden Komponenten (*Core Agent* und *Naming and Replication*) registriert ist, kann sie ihre Managementaufgabe erfüllen, das heißt, eine reale Ressource repräsentieren. Damit kann nun ein OSI-Manager beliebige CMIS-Anforderungen auf dieser Instanz auslösen. Falls in einer CMIS-Anforderung eine Instanz spezifiziert ist, die nicht im *spezifischen Agententeil* existiert und damit nicht im *Core Agent* und in der Containment-Hierarchie registiert ist, wird eine *noSuchInstance*-Fehlermeldung erzeugt.

Das Kreieren von Instanzen kann dabei auf zwei verschiedene Arten ausgelöst werden:

1. Eine darunterliegende (zu repräsentierende) Ressource löst das Kreieren einer Instanz aus (createFromResource), oder

2. die Instanz wird mittels eines CMIS M-CREATE-Requests erstellt.

Der Unterschied zwischen den beiden Möglichkeiten besteht darin, daß die Auslöser verschieden sind: Im ersten Fall ist eine darunterliegende Ressource, also ein Teil des Agenten selbst, im zweiten Fall irgendein OSI-Manager für das Auslösen des Kreierens einer Instanz einer Managementobjekt-Klasse verantwortlich. Da das „Bevölkern" des Agenten automatisch und somit vom Agenten selbst ausgelöst wird, handelt es sich um ein *createFromResource*.
Die Bearbeitung des CMIS-M-CREATE-Requests stellt keine besonderen Anforderungen, außer natürlich dem schon erwähnten Kreieren der neuen Instanz im *spezifischen Agententeil* und dem anschließenden Registrieren im *Core Agent* und in der Containment-Hierarchie im *Infratop*. Zu der Bearbeitung eines *createFromResource* sollen aber noch einige Bemerkungen erfolgen:

Ein *createFromResource* kann definitionsgemäß nur vom Agenten selbst aufgerufen werden. Die Instanzen im *spezifischen Agententeil* des Agenten repräsentieren die zu überwachende, darunterliegende, reale Ressource. Sobald Veränderungen in der realen Ressource stattfinden, müssen auch dementsprechende Veränderungen im *spezifischen Agententeil* des Agenten erfolgen. Folgendes Beispiel soll die Situation verdeutlichen:
Ein Agent soll die Prozeßliste einer UNIX-Workstation überwachen und steuern. Dazu wird jeder existierende Prozeß von einer eigenen Instanz repräsentiert. Sobald ein neuer Prozeß entsteht, muß eine neue Instanz kreiert werden. Ebenso muß für jeden terminierten Prozeß die entsprechende Instanz gelöscht werden.
Der Agent ist also selbst dafür verantwortlich, seinen *spezifischen Agententeil* und damit auch die Containment-Hierarchie im Infratop, durch entsprechendes Kreieren (createFromResource) bzw. Löschen (deleteFromResource) von Instanzen von Managementobjekten konsistent mit der durch diese Instanzen repräsentierten, realen Ressource zu halten. Das heißt, sobald eine durch eine Instanz repräsentierte Ressource terminiert, muß auch die entsprechende Instanz gelöscht werden. Ebenso muß für eine neu entstandene Ressource auch eine, diese repräsentierende, Instanz kreiert werden. Diese Eigenschaft stellt bei der späteren Vorstellung der Architektur des CMIP/SNMP Gateways eine Restriktion dar (siehe Kapitel 5).

Nachdem eine Instanz auf eine der beiden gerade vorgestellten Möglichkeiten im *spezifischen Agententeil* des Agenten kreiert und damit in dem *Core Agent* und in der Containment-Hierarchie registriert wurde, kann ein OSI-Manager mit Hilfe von CMIS-Requests die reale Ressource, die diese Instanz repräsentiert, überwachen und steuern. Die Abbildung 3.8 zeigt ein Beispiel einer Bearbeitung eines gescopten und gefilterten CMIS-M-GET-Requests (1). Dieser CMIS-M-GET-Requests, wird von der im Infratop enthaltenen *Infrastruktur* vom OSI-Manager entgegengenommen und zur *Naming and Replication*-Komponente weitergeleitet (2). Dort werden die in diesem Request angesprochenen Instanzen (Scope) ermittelt (3). Falls die angesprochene *Base Managed Object Instance (BMOC)* nicht in der Containment-Hierarchie existiert, wird eine *noSuchInstance*-Fehlermeldung zum OSI-Manager zurückgeschickt (4). Andernfalls erhält jede angesprochende Instanz eine „Kopie“ (Replication) des originalen Requests. Dazu werden diese Kopien von der *Naming and Replication*-Komponente an den *Core Agent* übergeben (5). Dieser kann, anhand der in der Kopie angesprochenen Instanz, diese im *spezifischen Agententeil* lokalisieren (6) und die Bearbeitung des Requests im *spezifischen Agententeil* auslösen (7). Anschließend werden eventuelle Ergebnisse zu „linked Replies“ zusammengefaßt (8) und über die *Infrastruktur* (9) zum OSI-Manager zurückgeschickt.

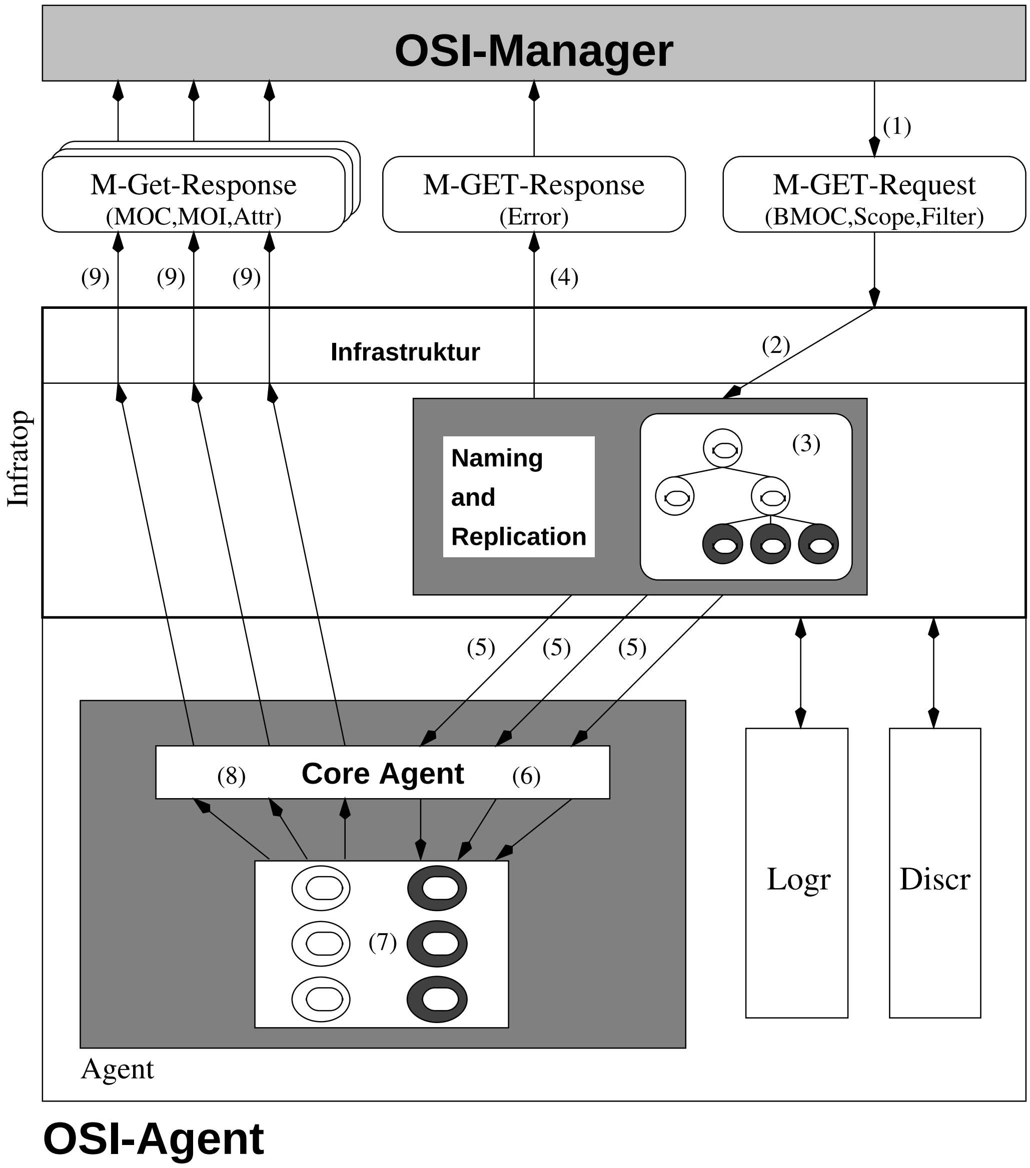


Abbildung 3.8: Der Ablauf einer Bearbeitung eines gescopten und gefilterten CMIS-M-GET-Requests

3.2.5 Zusammenfassung

Die *IBM TMN WorkBench for AIX* mit ihren Tools und Services erleichtert die Entwicklung eines TMN- bzw. OSI-konformen Agenten enorm. Die Erleichterung wird dadurch erreicht, daß das generische oder standard Verhalten eines OSI-Agenten, welches in den *OSI Systems Management* Standards definiert wird, nicht in den Aufgabenbereich eines Agenten-Entwicklers fällt, sondern von den Werkzeugen automatisch implementiert wird. Unter den generischen oder standard Verhalten werden die Problemstellungen verstanden, die unabhängig von der MIB des Agenten in allen Agenten gleichermaßen vorkommen. Dazu gehören (auszugsweise):

- Das Empfangen einer Request-PDU und die Darstellung dieser in einem internen Format.
- Das Auflösen von Scope-Parametern.
- Das Überprüfen von Filter-Attributen.
- Die Initialisierung der Attribute beim Kreieren einer Instanz. Der Attributwert kann durch verschiedene Möglichkeiten bestimmt werden, zum Beispiel durch:
  - einen Parameter in der Create-PDU,
  - einen Initial-Value in der GDMO-Definition,
  - den Wert eines Attributes in einem Referenzobjekt,
  - den Initial-Value in einem Callback.
- Das Generieren von *linked Replies*, falls sich ein Request auf mehrere Instanzen bezieht und damit zu einer Anfrage mehrere Antworten zurückgeliefert werden.

Die Aufgabe des Agenten-Entwicklers reduziert sich dabei einzig auf die Implementierung des Verhaltens von Managementobjekt-Klassen und deren Attributen, das heißt, die Implementierung der Zugriffe von den Instanzen dieser Managementobjekt-Klassen auf die realen Ressourcen, die von diesen repräsentiert werden. Diese Implementierung wird durch das *Callback*-Konzept strukturiert und dadurch übersichtlich und intuitiv. Diese Trennung der implementierungsspezifischen Details vom Standard-Verhalten des Agenten reduziert die Komplexität einer Agenten-Entwicklung erheblich.

Kapitel 4

Das Managementgateway

Aufgrund der Existenz von unterschiedlichen Managementarchitekturen und dem wachsenden Bedürfnis an Interoperabilität zwischen diesen Architekturen werden Mechanismen zur Realisierung der Integration von Managementarchitekturen ineinander benötigt. Das heißt, eine Managementanwendung in irgendeiner Managementarchitektur soll auf jegliche Managementinformation aus dem Gesamtnetz zugreifen können, egal, aus welcher Architektur diese Information stammt. Sobald dieser architekturübergreifende Zugriff transparent gestaltet werden kann, ist ein weiterer, großer Schritt in Richtung integriertes Management erreicht.
Für die Integration einer Managementarchitektur in eine andere eignen sich vor allem Managementgateways, da diese keine Veränderungen in existierenden Manager- bzw. Agenten-Implementierungen voraussetzen und der architekturübergreifende Zugriff transparent ausgeführt werden kann.
Diese Arbeit befaßt sich daher mit der Integration der Internet-Managementarchitektur in die OSI-Managementarchitektur mit dem Ziel eines *CMIP/SNMP Gateways.*

4.1 Aufbau und Anforderungen

Durch den Einsatz von Managementgateways für die Integration von einer Managementarchitektur in eine andere und hier im speziellen des Internet-Managements in die OSI-Architektur, entstehen *Management-Hierarchien*, siehe Abbildung 4.1. Management-Hierarchien dienen vor allen dazu, Management-Funktionalität zu verlagern. Dazu wird ein „intermediate Agent“ konstruiert, der sich gegenüber einem Manager wie ein Agent und gegenüber einem Agent wie ein Manager verhält. Dieser „intermediate Agent“ kann nun Aufgaben erfüllen, zum Beispiel das Sammeln und Aufbereiten von Managementinformationen. Diese Verlagerung der Funktionalität bewirkt eine Reduzierung der Komplexität und Vielfalt der zu verarbeitenden Managementinformationen. Falls die beiden Managementprotokolle, die der „intermediate Agent“ unterstützt, verschieden sind, wie in diesem

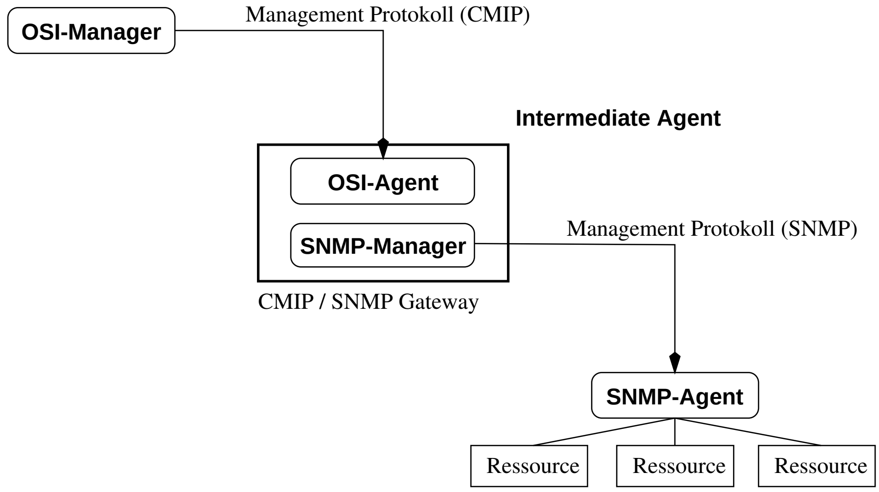


Abbildung 4.1: Management Hierarchie und „intermediate Agent“

Fall, spricht man von einem „Gateway“, hier einem *CMIP/SNMP Gateway*.
Die Aufgabe eines CMIP/SNMP Gateways ist es, es einem OSI-Manager zu ermöglichen, mit der kompletten CMIS-Funktionalität auf SNMP-Resourcen zugreifen zu können. Dies erfordert erstens eine Abbildung der Managementoperationen und zweitens eine Abbildung der auszutauschenden Managementinformationen.
Allgemein müssen für die Integration einer Managementarchitektur in eine andere die Teilmodelle (Informationsmodell, Kommunikationsmodell, Funktionsmodell und Organisationsmodell) aufeinander abgebildet werden.
In diesem Falle, der Integration des Internet-Managements in die OSI-Architektur, ist nur eine Abbildung des Informations- und des Kommunikationsmodells notwendig. Die Organisationsmodelle der beiden Managementarchitekturen unterscheiden sich nicht, es handelt sich dabei um Manager-Agent-Beziehungen, und müssen daher nicht abgebildet werden. Die Funktionsmodelle können nicht aufeinander abgebildet werden, da in der Internet-Architektur kein Funktionsmodell existiert[1]. Die OSI-Architektur besitzt mit den *Systems Management Functions (SMFs)* ein sehr mächtiges Funktionsmodell.
Zusammenfassend zeigt Abbildung 4.2 das Gateway als „intermediate Agent“, welches gegenüber dem OSI-Manager eine OSI-Agentenrolle und gegenüber dem SNMP-Agenten einen SNMP-Managerrolle einnimmt. Weiterhin wird durch die Existenz einerseits einer GDMO-MIB (das heißt, die *Managed Objects* in dieser MIB sind nach OSI-SMI definiert) und andererseits einer Internet-MIB (das heißt, die *Managed Objects* in dieser MIB sind nach der Internet-SMI definiert) bereits die Notwendigkeit einer Informationsmodellabbildung aufgezeigt. Schließlich wird

[1] Die *RMON-MIB* könnte eventuell als Funktionsmodell der Internetarchitektur angesehen werden

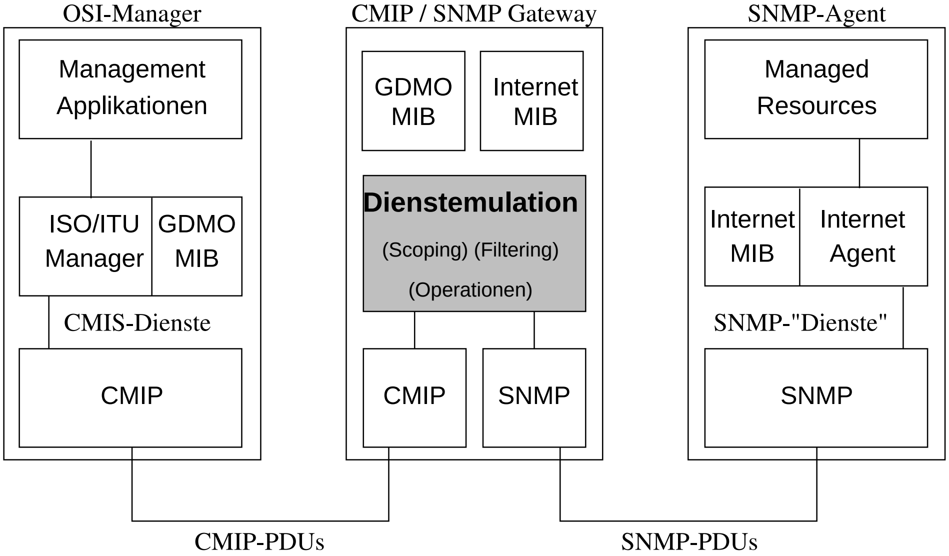


Abbildung 4.2: Der Aufbau und die Einordnung eines CMIP/SNMP Gateways

mit der Komponente „Dienstemulation“ auf die Aufgabe der Kommunikationsmodellabbildung hingewiesen.

4.2 Abbildung der Informationsmodelle

Ein Informationsmodell einer Managementarchitektur stellt die Datenbasis für die Managementinformation und damit die gemeinsame Grundlage für die Verständigung von Manager und Agent bereit. Dieses Modell beschreibt jede zu verwaltende Ressource mit allen benötigten Eigenschaften eindeutig. Eine Managementapplikation in einer Managementarchitektur kann nur Ressourcen managen, die nach dem Informationsmodell dieser Architektur dargestellt sind.
Die OSI-Architektur definiert für das Informationsmodell einen objektorientierten Ansatz. Objektorientiert bedeuted dabei die Verwendung von Datenabstraktion, Vererbung, Containment und Polymorphie. *Managed Objects (MO)* sind die Abstraktion von realen Ressourcen. *Managed Objects* sind Instanzen von *Managed Object Classes (MOCs)*, welche die Eigenschaften einer bestimmten Ressource in einer standardisierten Notation [ISO10165-4] definieren. Eine *Managed Object Class* kann als eine Unterklasse von einer oder mehreren Vaterklasse(n) definiert werden und erbt dadurch alle Eigenschaften der Vaterklasse(n). Diese Vererbungsmöglichkeiten bilden eine Klassenhierarchie, die Vererbungshierarchie. Weiterhin werden *Managed Objects* in der *Management Information Base (MIB)* zusammengefaßt. Die dabei erfolgte Anordnung beschreibt die Containment-Hierarchie (siehe 2.1.2).
Verglichen mit dem OSI-Ansatz, liegt der Internet-Architektur ein viel einfacheres

Informationsmodell zugrunde. Die *Internet-MIB* besteht dabei nur aus Objekten (nicht im objektorientierten Sinn) mit Variablen. Beides, der Objekt-Typ und die Objekt-Instanz (=SNMP-Variable) sind Teil des Registrierungsbaumes, wobei die Variablen die Blätter bilden. Schließlich unterstützt das Internet-Informationsmodell noch die Darstellung von realen Ressourcen durch Tabellen.
Diese großen Unterschiede in den Informationsmodellen der OSI- bzw. Internet-Architektur zeigen, daß die Aufgabe des Gateways nicht nur darin besteht, einfache PDU-Konvertierung (Protocol Data Unit) durchzuführen. Das Gateway soll einem OSI-Manager den Zugriff auf SNMP-Ressourcen ermöglichen, wobei dieser Zugriff aber für den Manager transparent sein soll, das heißt, der Manager greift auf diese Informationen/Ressourcen genau so zu, wie auf Informationen/Ressourcen aus OSI-Agenten. Dazu muß das Gateway eine OSI-Sichtweise auf die SNMP-Ressourcen herstellen. Dies wird durch die Abbildung des Internet- auf das OSI-Informationsmodell ermöglicht. Das Ergebnis dieser Abbildung ist eine Übersetzung der Internet-MIB in GDMO-Notation. Das heißt, alle Internet-MOs werden auf OSI-MOs abgebildet. Diese neuen OSI-MOs haben im Gateway die Aufgabe, die SNMP-Ressourcen zu repräsentieren. Damit wird vom Gateway eine OSI-Sichtweise auf die Internet-Managementinformation bereitgestellt.
In den nächsten beiden Abschnitten werden zwei verschiedene Algorithmen zur Abbildung der Informationsmodelle und damit zur Übersetzung von Internet-MIBs in GDMO MIBs, vorgestellt und miteinander verglichen:

4.2.1 Direkte Übersetzung

Mit der *direkten Übersetzung* wird dabei der von der IIMC veröffentlichte und im Abschnitt 3.1.1 vorgestellte Übersetzungsalgorithmus bezeichnet. Dieser Algorithmus bildet die Internet-Objekte *direkt* auf neue OSI GDMO Klassen ab, unter Einhaltung folgende Regeln:

- SNMP-MIB-Gruppen werden zu Managementobjekt-Klassen übersetzt, zum Beispiel : *system*, *ip*, ...
- SNMP-Tabellenzeilen werden zu Managementobjekt-Klassen übersetzt, zum Beispiel : *ipAddrEntry*.
- Alle anderen SNMP Objekte werden Attribute in den entsprechenden Managementobjekt-Klassen, zum Beispiel wird die SNMP-Variable *sysContact* aus der SNMP-Gruppe *system* ein Attribut der Managementobjekt-Klasse, die aus der SNMP-Gruppe *system* entstanden ist.

4.2.2 Abstrakte Übersetzung

Die *abstrakte Übersetzung* bildet dagegen die Internet-Objekte auf OSI GDMO Klassen ab, die durch geeignete Vererbung aus bereits existierenden GDMO-Klas-

sen entstehen.
Als Grundlage für die Vererbungshierarchie können die *Generic Managed Object Classes (GMOC)* der ISO-Norm 10165-5 für die Modellierung der Internet-Ressourcen benutzt werden. Damit kann das Internet-Management in ein globales OSI-Management eingebunden werden und somit ein Teil des *Global Network Model*, siehe auch [AbClHo93].
Beim Einsatz dieser GMOCs treten aber verschiedene Probleme auf, die daraus resultieren, daß die ISO-Norm 10165-5 nur eine kleine Anzahl von diesen generischen Managementobjekt-Klassen definiert, diese aber gleichzeitig durch ein sehr abstraktes Verhalten charakterisiert werden.
In [Beier93] wird der Versuch gemacht, Teile der Internet MIB-2 in eine OSI-MIB zu übersetzen, wobei die dabei entstandenen Klassen von den GMOCs abgeleitet wurden. Dieses Beispiel zeigt deutlich, daß die *abstrakte Übersetzung* vor allem für Ressourcen, die nicht zu dem OSI-Schichtenmodell konform sind (wie zum Beispiel die Internet-Architektur), schwierig ist und es dabei zu verschiedenen, aber jeweils sinnvollen, Lösungsmöglichkeiten für die Vererbungshierarchie kommen kann. Daher ist es auch nicht möglich, diesen Prozeß zu automatisieren oder gar zu standardisieren.

4.2.3 Diskussion

Der Vorteil der abstrakten Übersetzung ist, daß das Internet-Management vollständig in ein globales OSI-Management eingebunden werden kann. Dies kann mit der direkten Übersetzung nicht erreicht werden, denn dort wird jede Managementinformation, egal, ob diese eventuell durch schon existierende OSI Managmentobjekt-Klassen modelliert werden kann, von neuen, von „top“ abgeleiteten, Klassen repräsentiert. Dadurch wird kein Gebrauch von der objektorientierten Eigenschaft der Vererbung gemacht.
Ein Nachteil der abstrakten Übersetzung ist, daß sie weder automatisiert noch standardisiert werden kann. Es werden, zusätzlich zu der SNMP-MIB, Informationen für die Modellierung der Managementinformation benötigt. Dies wird bei der direkten Übersetzung vermieden: Die Übersetzung erfolgt nach einem vom NM-Forum standardisierten Algorithmus. Ein großer Vorteil der direkten Übersetzung ist zusätzlich, daß sie automatisierbar ist und dafür auch bereits ein Compiler existiert (siehe Abschnitt 3.1.2).
Aus diesen Gründen[2] wurde in der Aufgabenstellung dieser Diplomarbeit festgelegt, daß die Informationsmodellabbildung nach der direkten Übersetzung erfolgen soll.

[2]wegen der Automatisierbarkeit und der Standardisierung des Algorithmus

4.3 Abbildung der Kommunikationsmodelle

Nachdem im vorigen Abschnitt beschrieben wurde, wie die Managementinformation bei der Integration des Internet-Managements in die OSI-Architektur abgebildet werden soll, folgt in diesem Kapitel eine Diskussion, wie diese Managementinformation ausgetauscht werden kann. Dazu müssen die Kommunikationsmodelle aufeinander abgebildet werden. Die Aufgabe eines Gateways, es einem OSI-Manager zu ermöglichen, SNMP-Agenten zu managen, kann erst erfüllt werden, wenn *beide* Abbildungen spezifiziert und ausgeführt wurden.
Im Internet-Kommunikationsmodell wird das SNMP-Protokoll definiert. Es existieren dabei die PDUs *GetRequest, GetNextRequest, GetBulk und SetRequest* für die Kommunikation eines SNMP-Managers mit einem SNMP-Agenten. Der Agent empfängt diese PDUs, bearbeitet sie und schickt eine GetResponse-PDU zurück. Weiterhin kann der Agent Ereignisse mittels Traps unaufgefordert dem Manager mitteilen. Da SNMP auf dem Internet-Protokoll *UDP* (datagramm-orientiert) basiert, handelt es sich um eine unbestätigte Kommunikation.
Dagegen basiert das Kommunikationsmodell für das schichtenübergreifende Management der OSI-Architektur auf dem CMIP-Protokoll. Einem OSI-Manager stehen für die Kommunikation mit einem OSI-Agenten folgende PDUs zur Verfügung: m-Get, m-Set, m-Create, m-Delete, m-Action und m-GetCancel. Eine Instanz einer Managementobjekt-Klasse in einer MIB eines OSI-Agenten kann Ereignisse mit Hilfe von m-EventReport PDUs an den OSI-Manager schicken.
Die Kommunikation kann dabei allgemein sowohl bestätigt, als auch unbestätigt erfolgen. Einige dieser PDUs erlauben zusätzlich das Übertragen von Scopes und Filtern (m-Get, m-Set, m-Action, m-Delete). Auch kann in diesen PDUs eine Synchronisationsbedingung übermittelt werden.
Für den Austausch von Managementinformation zwischen diesen beiden Architekturen bedarf es einer Protokollkonvertierung, die vom Gateway durchzuführen ist. Diese Übersetzung beruht auf folgenden Schritten:

- Die vom OSI-Manager abgeschickten PDUs müssen empfangen, entschlüsselt und verarbeitet werden.

- Diese Verarbeitung der CMIP-Requests kann einerseits dadurch erfolgen, daß im Gateway selbst Aktionen ausgeführt bzw. Informationen gelesen werden. Andererseits kann ein CMIP-Request auch SNMP-PDUs auslösen, um vom SNMP-Agenten Informationen zu erhalten bzw. Aktionen im Agenten auszulösen. Es sind sicherlich auch CMIP-Requests denkbar, zu deren Bearbeitung beides notwendig ist.

- Bei der Bearbeitung müssen eventuell vorhandene Scopes, Filter oder Synchronisationsbedingungen beachtet und ausgeführt werden.

- Nach der Bearbeitung eines CMIP-Requests werden die Ergebnisse in Response-PDUs verpackt und zum Manager zurückgeschickt.

- Die vom SNMP-Agenten ausgelösten Ereignismeldungen (Traps) müssen vom Gateway in OSI-Notifikationen übersetzt und zu den entsprechenden EFDs (Event Forwarding Discriminator) weitergeleitet werden. Diese übernehmen schließlich den Transport zu den jeweiligen OSI-Managern.

Diese Bearbeitungsschritte lassen eine weitere Hierarchie erkennen, die in [AbClHo93] „Management-Informationshierarchie“ genannt wird. In dieser Management-Informationshierarchie wird die Managementinformation in den Blättern der Hierarchie gesammelt und zu einem gewissen Grad an die höheren Hierarchiestufen weitergeleitet. Auf diese, hier existierende Management-Hierarchie mit dem OSI-Manager, dem Gateway und dem SNMP-Agent können zwei verschiedene Management-Informationshierarchien gefunden werden, je nachdem, wie das Gateway die Managementinformationen verwaltet:

- Zustandslose (stateless) Komponenten speichern keine Managementinformationen. Es werden stattdessen alle von einer höheren Informationshierarchiestufe angeforderten Informationen direkt aus einer darunterliegenden Informationshierarchiestufe abgefragt. Für ein stateless CMIP/SNMP Gateway bedeuted dies, daß jede ankommende CMIP-PDU auf eine oder mehrere SNMP-PDU(s) abgebildet wird. Die SNMP-Response(s) wird (werden) anschließend mittels einer oder mehrerer CMIP-Response-PDU(s) zurückgeschickt. Ebenso werden SNMP-Traps auf EventReport von speziellen Instanzen von Managementobjekten abgebildet. Das Gateway speichert *keine* Managementinformationen aus den MIBs der SNMP-Agenten.

- Zustandsbehaftete (stateful) Komponenten speichern die Informationen einer unteren Informationshierarchie. Für ein stateful CMIP/SNMP Gateway bedeutet dies, daß die Internet-MIBs der SNMP-Agenten komplett oder nur teilweise in einer lokalen Gateway-MIB repliziert werden müssen. Die ankommenden CMIP-PDUs werden nun direkt auf der lokalen Gateway-MIB bearbeitet. Die Informationen in dieser replizierten Gateway-MIB sollen möglichst aktuell sein, wobei temporäre Inkonsistenzen aber unvermeidbar sind. SNMP-Operationen werden nur zur Konsistenzsicherung der lokalen Gateway-MIB benötigt.

Die Vor- und Nachteile der beiden Lösungen in Bezug auf die Performance sind offensichtlich : Je mehr statisch die Managementinformation ist, um so besser ist das stateful Gateway geeignet, denn die Konsistenzsicherung bedarf keiner oder nur weniger Kosten. Ist dagegen die Information dynamisch, wird ein stateless Gateway vorzuziehen sein.
Die Vorteile eines stateful Gateways liegen unter anderem darin, daß Scoping und Filtering direkt in der lokalen MIB ausgeführt werden können und nicht simuliert werden müssen, so wie bei einem stateless Gateway. Dort werden alle ankommenden CMIP-Request direkt in SNMP-PDUs umgewandelt, unter Berücksichtigung

und Auflösung von Scopes und Filtern (siehe 4.4.1). Der Preis für diese Vereinfachung im stateful Gateway (siehe 4.4.2) wird aber mit hohen Kosten bezahlt, denn es werden komplexe Algorithmen für das Replizieren der SNMP-MIBs notwendig. Dazu muß ein Caching-Mechanismus implementiert werden, der sicherstellt, daß die Informationen in der lokalen Gateway-MIB möglichst konsistent mit den realen Ressourcen in den SNMP-Agenten sind, die sie ja repräsentieren.
Ein optimales, idealisiertes Gateway würde ein Kompromiß eingehen, der die Vorteile jeder Lösung integriert und die Nachteile vermeidet: für dynamische Managementinformation würde es sich wie ein stateless Gateway, für statische Managementinformation würde es sich wie ein stateful Gateway verhalten. Dazu ist es aber notwendig, daß das Gateway Wissen über das Verhalten der Managementinformation besitzt. Dieses Wissen ist aber nicht in der MIB gespeichert, muß also auf andere Weise beschafft werden[3].

4.4 Arten von Gateways

Am Anfang diese Kapitels wurden der Aufbau eines und die Anforderungen an ein CMIP/SNMP Gateway beschrieben. Das Gateway muß einerseits, um die zu managende SNMP-Ressource in einer OSI-Sichtweise darzustellen, eine Informationsmodellabbildung durchführen. Andererseits müssen die CMIS-Anforderungen eines OSI-Managers auf die SNMP-Ressourcen projiziert werden. Dies erfolgt durch die Kommunikationsmodellabbildung.
Für jede dieser Abbildungen wurden zwei Alternativen aufgeführt. Da für die Übersetzung der Internet-MIBs zu OSI-MIBs aus ober erwähnten Gründen der Algorithmus der IIMC verwendet werden soll, werden im folgenden zwei verschiedene CMIP/SNMP Gateways vorgestellt, die sich hinsichtlich der Kommunikationsmodellabbildung unterscheiden.
So wird zuerst das theoretische Konzept, das einem stateless Gateway zugrunde liegt, vorgestellt. Anschließend soll die Kommunikationsmodellabbildung anhand eines stateful-Ansatzes diskutiert werden.

4.4.1 Das stateless Gateway

Die IIMC veröffentlichte in einem Internet Draft [IIMCPROXY] die notwendigen Konzepte für die Realisierung eines stateless Gateways. Die Informationsmodellabbildung beruht dabei auf dem ebenfalls von der IIMC veröffentlichten und in 3.1.1 vorgestellten Übersetzungsalgorithmus. Dieses, von der IIMC spezifizierte, Gateway entspricht somit genau den Anforderungen, wie sie im vorigen Teil dieses Kapitels aufgeführt wurden und soll deswegen hier genauer vorgestellt werden:
Dieses Gateway, von der IIMC mit *ISO/Internet Proxy* bezeichnet, soll die

[3]Ein Lösungsansatz wird in Abschnitt 4.4.2 vorgestellt.

SNMP-Ressourcen von einem oder mehreren SNMP-Agenten für das Management mit einem OSI-Manager zur Verfügung stellen. Dazu werden zuerst die MIBs der SNMP-Agenten mit dem oben erwähnten Algorithmus in GDMO-Notation umgewandelt. Die MIB des Gateways wird danach aus zwei Kategorien von Managed Objects aufgebaut:
Zum einen aus „remote Objects" und zum anderen aus „local Objects". Bei den „remote Objects" handelt es sich um Instanzen der Klassen, die aus der Übersetzung der Internet-MIBs in GDMO MIBs entstanden sind. Sie repräsentieren damit die zu überwachenden und steuernden Internet-Ressourcen im Gateway. Über Operationen bzw. Aktionen auf diesen „remote Objects" wird auf die SNMP-Agenten direkt zugegriffen. Das heißt, durch einen Zugriff auf diese „remote Objects" wird eine Kommunikationsmodellabbildung und damit eine Protokollkonvertierung von CMIP nach SNMP durchgeführt.
Bei den „local Objects" handelt es sich um Instanzen von Klassen, die das Gateway repräsentieren und damit für das Management des Gateways selbst verantwortlich sind. So können in diesen Managed Objects die SNMP-Agenten, mit dazu notwendigen Konfigurationsdaten wie IP-Adresse oder Community Strings, gespeichert werden, die gerade durch die „remote Objects" in der MIB des Gateways repräsentiert werden. Weiterhin sind mit diesen „local Objects" Instanzen denkbar, die Schwellwerte überwachen, Statistiken erstellen, bei irgendwelchen Ereignissen, zum Beispiel bei einer Wertänderung eines Attributs, Notifikationen auslösen usw.
Die Hauptaufgabe des Gateways besteht aber darin, CMIS-Anforderungen eines OSI-Managers auf den „remote Objects" auf geeignete SNMP-PDUs abzubilden. Der Proxy stellt definitionsgemäß in der Management-Informationshierarchie eine stateless Komponente dar, das heißt, alle Anforderungen einer höheren Hierarchiestufe (=OSI-Manager) werden direkt auf eine untere Hierarchiestufe (=SNMP-Agent) abgebildet. Die folgenden Abschnitte befassen sich deswegen mit der konzeptionellen Durchführung der Abbildung von CMIS-Diensten auf SNMP-PDUs im Gateway.
Die Abbildung läßt sich dabei in zwei unabhängige Aufgabenbereiche einteilen (siehe auch Abbildung 4.3):

1. Name Mapping :
   Das Gateway muß eine Abbildung ausführen, um den OSI-Namen der Managementinformation, die in dem CMIS-Request spezifiziert wird, in den korrespondierenden Internet-Namen (OID) umzuwandeln.

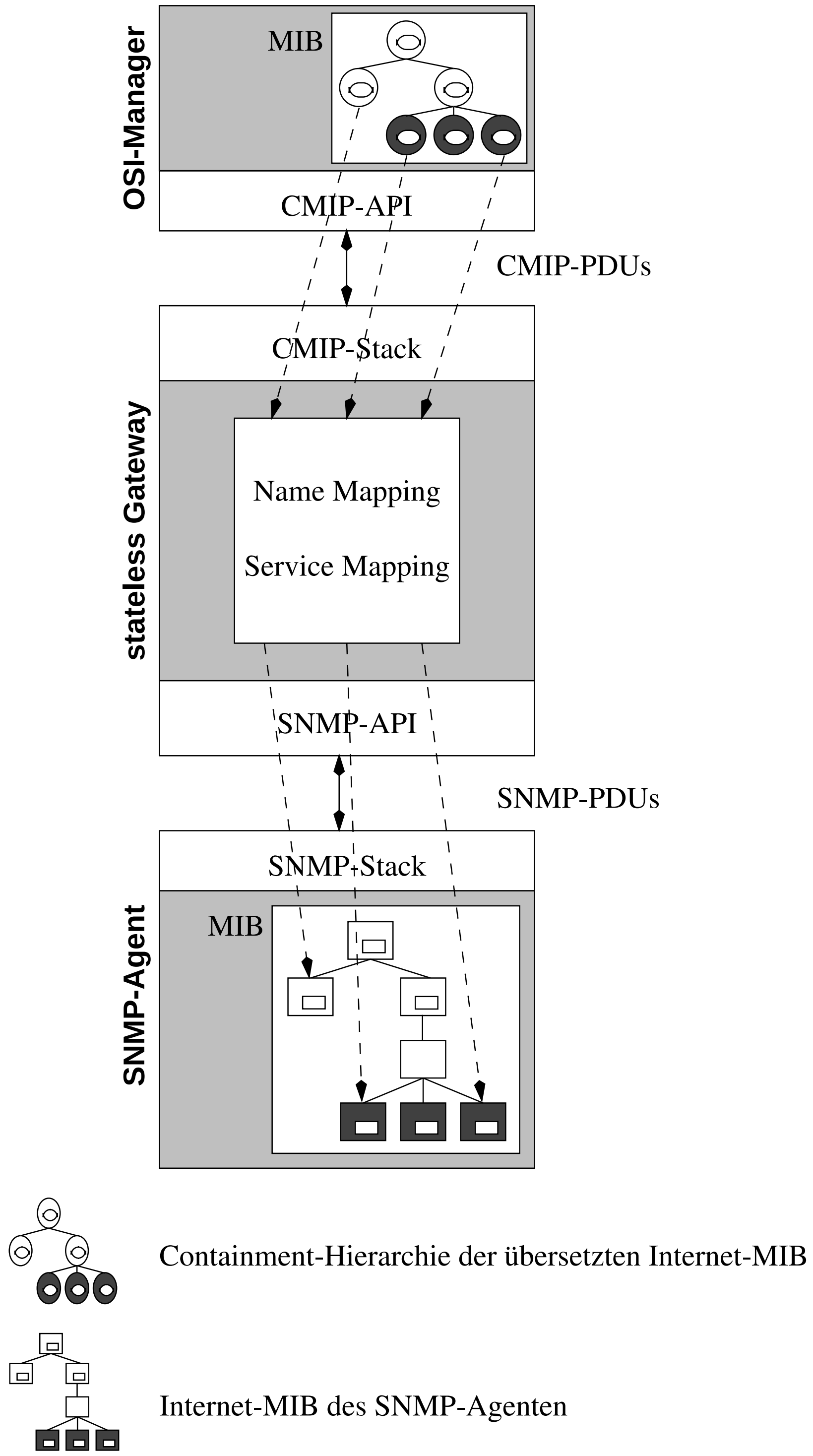


Abbildung 4.3: Ein stateless Gateway

2. Service Mapping :
   Um die Interoperabilität zwischen den beiden Managementarchitekturen zu erreichen, ist es notwendig, die Dienste bzw. Funktionalität, die eine Architektur zur Verfügung stellt, in die jeweils andere zu übersetzen. Diese Abbildung wird durch das „Service Mapping“ möglich. Im einfachsten Fall kann zum Beispiel ein CMIS-M-GET-Request auf eine SNMP-GetRequest-PDU abgebildet werden.

Diese beiden Problemstellungen sollen im folgenden detaillierter diskutiert werden:

Name Mapping

Sobald ein OSI-Manager auf ein Internetobjekt zugreifen will, identifiziert er diese Ressource nach dem Namensschema, welches auf der Containment-Hierarchie basiert. Die Aufgabe des Gateways besteht darin, den originalen Internet Objekt Identifikator dieser Ressource wiederherzustellen, welcher aus dem Internet-Registrierungbaum abgeleitet ist.
In Abschnitt 3.1.1 wurde für die Registrierung und Namensgebung der OSI Managementobjekte, die Internet-Ressourcen repräsentieren, ein Algorithmus angegeben. Beim „Name Mapping“ besteht die Aufgabe, genau diesen Algorithmus umzukehren und aus der Registrierung und Bezeichnung der OSI Managed Objects den originalen Namen der Internet-Ressource zu gewinnen. Folgendes Beispiel soll den Sachverhalt verdeutlichen:
Es wird angenommen, daß in einem CMIS-M-GET-Request auf ein Attribut eines Managementobjekts zugegriffen werden soll, das einen SNMP-Zähler repräsentiert. Dieser Zähler soll in diesem Beispiel die SNMP-Variable „tcpActiveOpens“ sein. Die OID dieser MIB-2-Variablen ist dabei „1.3.6.1.2.6.5“.
Im OSI-Modell wird dieser Zähler durch ein Attribut in der Managementobjekt-Klasse „{iimcRFC12131354}:tcp“ repräsentiert, welche aus der Übersetzung der SNMP-MIB-2-Gruppe „tcp“ entstanden ist. Wie im Abschnitt 3.1.1 beschrieben, wird einerseits die Klasse „{iimcRFC12131354}:tcp“ unter der OID „{iimcAutoObjAndAttr} .1.3.6.1.2.6“ und andererseits das Attribute „tcpActiveOpens“ unter der OID „{iimcAutoObjAndAttr} .1.3.6.1.2.6.5“ registriert.
Die Aufgabe des Gateways ist es nun, die OID der Internet-Objektinstanz in der Internet-MIB herauszufinden. Da der Name einer Objektinstanz aus zwei Teilen besteht, zum ersten aus dem Objekt-Typ und zum zweiten aus dem Suffix, welches den Wert 0 für nicht Spalten-Objekte und den Wert des Indizes für Spalten-Objekte annimmt, kann der Vorgang getrennt werden : zum einen in die Bestimmung der OID des Objekt-Typs und zum anderen in die Bestimmung des Suffix. Ersteres kann dadurch gelöst werden, daß von der OSI-OID der Klasse oder des Attributs, welches die Internet-Ressource repräsentiert, der erste Teil („{iimcAutoObjAndAttr}“) weggelassen wird. Das Suffix wird durch den RDN

(Relative Distinguished Name) der zu der Internet-Ressource korrespondierenden Instanz eines Managementobjekts bestimmt.
So wird zum Beispiel die OID der Internet-MIB-2-Variablen *tcpActiveOpens* jetzt dadurch bestimmt, daß von der OID des Attributes der erste Teil „{iimcAutoObjAndAttr}“ weggelassen wird und als Suffix der RDN der Instanz der Klasse „{iimcRFC12131354}:tcp“ (der Wert ist 0) angehängt wird.
Die Abbildung 4.4 zeigt den zweigeteilten Prozeß:

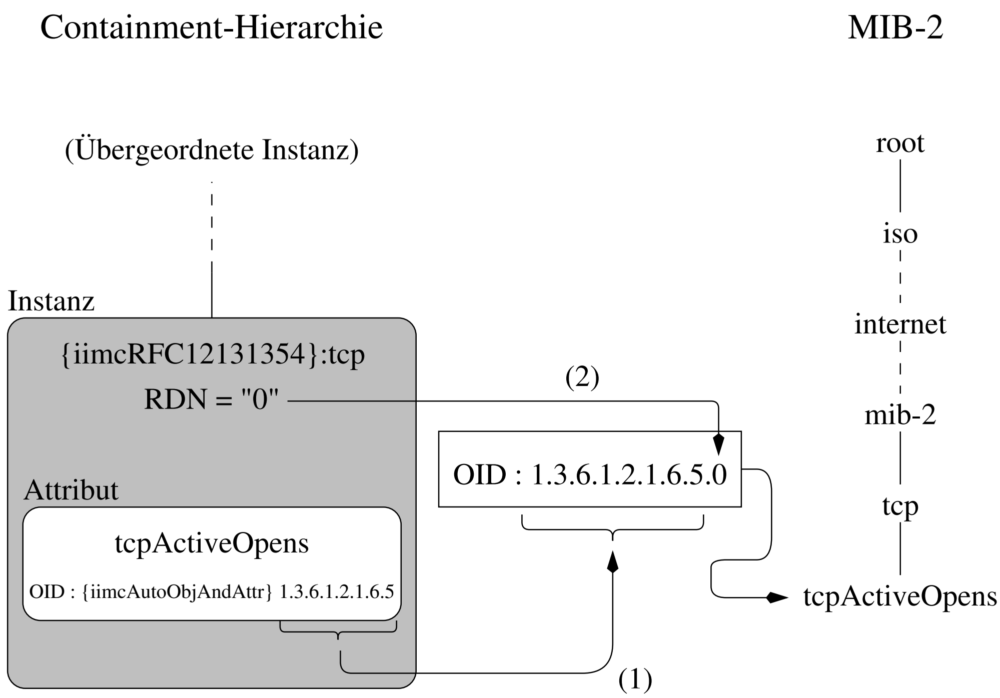


Abbildung 4.4: Die Problematik des „Name Mapping“

Service Mapping

In diesem Abschnitt soll die Abbildung der CMIS-Basisdienste auf SNMP-PDUs erfolgen. Der Algorithmus dafür aus dem Internet Draft [IIMCPROXY] wird in die Objektauswahl und die Bestimmung von Instanzen getrennt und soll hier kurz vorgestellt werden.

Objektauswahl: Scoping und Filtering Die Objektauswahl ist dafür zuständig, die Menge von Instanzen von Managementobjekten zu bestimmen, die in einem CMIS-Request durch den Scope und/oder den Filter angegeben werden. Scoping bezeichnet dabei einen Teilbaum in der Containment-Hierarchie. Ein Filter schränkt den im Scope definierten Teilbaum weiter ein, indem nur die Instanzen von Managementobjekten ausgewählt werden, bei welchen die Attribut-Werte

dem Filter entsprechen. Auf der somit erhaltenen Menge von Instanzen von Managementobjekten wird schließlich der CMIS-Request ausgeführt.
Falls kein Filter spezifizert ist, wird der CMIS-Request an allen, durch den Scope ausgewählten, Managementobjekten ausgeführt. Falls kein Scope spezifiziert ist, wird der CMIS-Request nur an der *Base Managed Object Instance* durchgeführt.
Da es sich bei dem Gateway um eine stateless Komponente handelt, hat es kein Wissen darüber, welche Objektinstanzen gerade aktuell im Internet-Agenten existieren. Daher müssen zuerst alle aktuellen Instanzen bestimmt werden, bevor festgelegt werden kann, welche Instanzen durch den Scope angesprochen werden. Eine Vereinfachung des Algorithmus zur Bestimmung der Instanzen von Managementobjekten im Scope ergibt sich dadurch, daß zuerst nur die Managementobjekt-Klassen festgestellt werden, deren Instanzen eventuell in den Scope fallen könnten. Danach kann festgestellt werden, welche Instanzen dieser Managementobjekt-Klassen gerade aktuell im Gateway existieren. Zur Bestimmung dieser Managementobjekt-Klassen wird der Containment-Baum, beginnend bei der im CMIS-Request gegeben *Base Managed Object Class*, anhand der möglichen NAME BINDINGs, vollständig durchlaufen. Die NAME BINDINGs geben dabei an, zu welcher *superior Managed Object Class* eine *subordinate Managed Object Class* in der Containment-Hierarchie existieren darf (siehe 5.2.1).

Bestimmung der Instanzen, Dienstemulation Nachdem nun alle im Scope vorhandenen Managementobjekt-Klassen bestimmt sind, wird überprüft, ob diese Managementobjekt-Klassen die im Filter angesprochenen Attribute besitzen. Ist dies nicht der Fall, brauchen diese Managementobjekt-Klassen bei der weiteren Bearbeitung nicht betrachtet zu werden.
Nun müssen die gerade aktuellen Instanzen dieser Managementobjekt-Klassen, die noch in Frage kommen, herausgefunden werden. Dabei sind zwei Möglichkeiten zu unterscheiden:

1. Für Managementobjekt-Klassen, die eine SNMP-Gruppe repräsentieren, wird eine SNMP-GetRequest-PDU auf eine beliebige Variable in dieser Gruppe ausgelöst, um festzustellen, ob eine Instanz dieser Klasse existiert. Dabei erfolgt das „Name Mapping“, um die OID der Internet-Ressource zu bestimmen.

2. Für Managementobjekt-Klassen, die SNMP-Tabellenzeilen repräsentieren, werden mehrere SNMP-GetNextRequest-PDUs benötigt, um alle gerade aktuellen Tabellenzeilen zu bestimmen. Die Internet-OID der Tabelle kann dabei aus der OSI-Registrierungs-OID der Managementobjekt-Klasse abgeleitet werden, indem das Suffix „{iimcAutoObjandAttr}“ weggelassen wird.

Für jede existierende Instanz muß jetzt überprüft werden, ob ein Filter existiert. Falls in der CMIP-PDU ein Filter spezifiziert ist und dessen Auswertung FALSE ergibt, wird die Bearbeitung dieser Instanz abgebrochen. Falls kein Filter in der

CMIP-PDU existiert oder die Auswertung eines spezifizierten Filters TRUE ergibt, wird der CMIS-Dienst auf dieser Instanz ausgeführt. Dies erfolgt nach folgenden Regeln:

- Falls es sich bei dem CMIS-Dienst um eine Leseoperation (M-GET) handelt, werden korrespondierenden SNMP-GetRequest-PDUs auf dem Internet-Agenten ausgeführt. Anschließend werden die Ergebnisse zu einer CMIS-M-GET-Response zusammengefaßt und zum Manager zurückgeschickt.

- Handelt es sich bei dem CMIS-Dienst um eine Schreiboperation (M-SET), werden entsprechende SNMP-SetRequest-PDUs erzeugt. Es werden hier ebenfalls die entsprechenden Ergebnisse des SNMP-Agenten zusammengefaßt und diesmal in einer CMIS-M-SET-Response zum Manager zurückgeschickt.

- SNMP bietet keine Operationen für das Kreieren und Löschen von Objekten. Die Objekte werden direkt vom Agent oder einem Subagenten bereitgestellt. Es kann dabei jedoch mit Tabelleneinträgen eine Ausnahme gemacht werden. Hier kann durch das Setzen von bestimmten Spalten ein neuer Tabelleneintrag erzeugt werden. Daher werden CMIS-M-CREATE-Anforderungen in SNMP-SetRequest-PDUs umgewandelt.

- Entsprechend verhält es sich mit CMIS-M-DELETE-Requests. Durch das Setzen eines Wertes einer bestimmten Tabellenspalte mit einer SNMP-SetRequest-PDU wird die entsprechende Tabellenzeile gelöscht.

Synchronisation Das Gateway kann nur eine „best effort“ Synchronisation unterstützen. Denn für eine „atomic“ Synchronisation wäre ein Transaktionskonzept notwendig. Dazu sind noch tiefergehende Untersuchungen notwendig und damit nicht Teil der Aufgabenstellung dieser Diplomarbeit.

Behandlung von Notifikationen Die Behandlung von Notifikationen (Traps) durch das Gateway unterscheidet sich im stateless bzw. stateful Ansatz nicht. Sie wird daher in einem eigenen, später folgendem Abschnitt (siehe 4.4.3) diskutiert.

Zusammenfassung

Dieses theoretische Konzept eines stateless Ansatzes für die Realisierung eines CMIP/SNMP Gateway wird der Hauptaufgabe gerecht, das Internet-Management in die OSI-Architektur zu integrieren.
Allerdings müssen dazu einige Bemerkungen angeführt werden:

- Da ein stateless Gateway jede ankommende CMIP-Anforderung direkt in eine oder mehrere SNMP-PDU(s) konvertiert, können, je nach Typ der Managementinformation, entscheidende Vor- bzw. Nachteile entstehen: Der Vorteil ist, daß sehr dynamische Information immer mit dem aktuellen Wert den OSI-Managern zur Verfügung gestellt werden kann. Dies ist auch für statische Informationen richtig, hier wird allerdings bei jedem Zugriff meist unnötig viel Kommunikation vom Gateway mit den SNMP-Agenten ausgelöst, obwohl sich der Wert der Information wahrscheinlich seit dem letzten Zugriff nicht veränderte. Weiterhin ist durch das Auflösen von Scopes und Filtern sehr viel Informationsaustausch zwischen Gateway und SNMP-Agenten notwendig, um die Instanz eines Managementobjekts herauszufinden, welche schließlich durch den CMIS-Dienst bearbeitet werden sollen. Dies kann zu enormen Performanceproblemen und ungeheuerer Netzlast führen. Dazu soll folgende, in der Praxis wahrscheinlich eher selten vorkommende, aber mögliche, CMIS-Anforderung betrachtet werden: Ein CMIS-M-GET vom Root-Objekt der Containment-Hierarchie mit dem Scope „wholeSubtree“ und der Attribut-Liste „allAttributes“. Das heißt, es sollen alle möglichen Instanzen von Managementobjekten mit all ihren Attributen des gesamten OSI-Containment-Baums zurückgegeben werden. Der Leser möge sich selbst vorstellen, wieviele SNMP-PDUs dazu notwendig sind, falls das Gateway eine große Anzahl an SNMP-Agenten verwaltet und jeder SNMP-Agent wiederum eine große Anzahl an SNMP-Gruppen mit vielen Variablen enthält.

- Weiterhin soll noch darauf hingewiesen werden, daß es mit einem stateless Gateway nicht möglich ist, Veränderungen von Managementinformationen zu erkennen. Somit ist es auch nicht möglich, die generischen Notifikationen *changeAttributeValue*, *objectCreation* bzw. *objectDeletion* zu realisieren.

- Zuletzt darf auch nicht unerwähnt bleiben, daß dieser stateless Ansatz, im Vergleich zu der stateful Realisierung, sehr viel einfacher zu implementieren ist, da er auf komplexe Strategien und Mechanismen[4] verzichten kann.

4.4.2 Das stateful Gateway

Die Kommunikationsmodellabbildung bei der Integration des Internet-Managements in die OSI-Architektur kann auch durch ein stateful Gateway erfolgen. Dabei wird nicht jeder CMIS-Request auf eine oder mehrere SNMP-PDU(s) abgebildet, sondern direkt auf einer lokalen Gateway-MIB ausgeführt, siehe Abbildung 4.5. Dazu müssen ein Teil oder die gesamte Information der MIBs der Internet-Agenten in der lokalen Gateway-MIB gespeichert werden.

[4] Für die Replikation der Internet-MIB

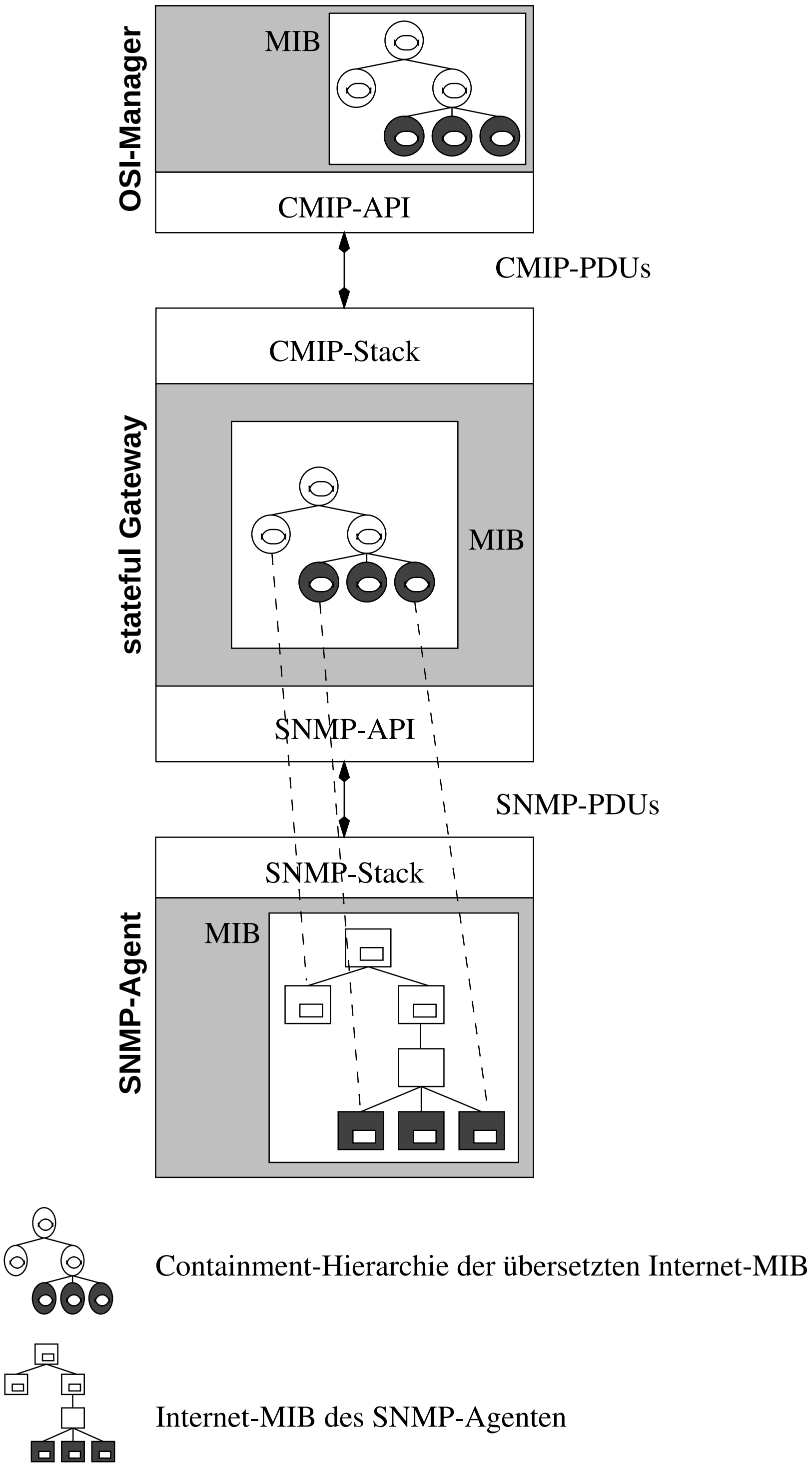


Abbildung 4.5: Ein stateful Gateway

Das heißt aber auch, daß die Managementinformation in dieser replizierten Gateway-MIB möglichst aktuell sein muß, also ein konsistenter Zustand zwischen der replizierten MIB und den originalen MIBs in den SNMP-Agenten, und damit zu den realen Ressourcen, bestehen.
Zwei zu lösende Problematiken ergeben sich aus dieser Situation:

1. Wer ist für die Konsistenzhaltung der Gateway-MIB verantwortlich ?

2. Zu welchem Zeitpunkt erfolgt eine Replikation einer bestimmten Ressource ?

Diese Problemstellungen sollen in den nächsten beiden Abschnitten vertieft werden.
Zuvor soll aber noch eine Bemerkung hinsichtlich des Cachings gemacht werden: Einerseits besteht die Replikation der Internet-MIBs in der lokalen Gateway-MIB aus der Auffrischung von Attributwerten. Das heißt, Attribute werden mittels SNMP-GetRequest-PDUs aktualisiert.
Andererseits können aber auch neue Tabellenzeilen in der Internet-MIB entstehen, bzw. Tabellenzeilen gelöscht werden. Dies bedeutet, daß die Containment-Hierarchie der replizierten Gateway-MIB verändert werden muß, also dem Kreieren bzw. Löschen von Instanzen von Managementobjekten. Das Erkennen von neuen Tabellenzeilen bzw. von veralteten Instanzen von Managementobjekten ist entweder die Aufgabe der in der Containment-Hierarchie übergeordneten Instanz[5] oder einer zentralen Komponente, siehe unten, und kann mittels eines SNMP-Walks[6] auf die jeweilige SNMP-Tabelle ereicht werden, siehe dazu 5.2.2.
Für die Replikation der Internet-MIBs in einer lokalen Gateway-MIB ist es notwendig, das OSI- auf das Internet-Namensschema abzubilden: Für jedes zu replizierende OSI-Managementobjekt in der lokalen Gateway-MIB muß für die Konsistenzsicherung der Name der korrespondierende Internet-Resource bestimmt werden. Diese Problematik tritt ebenso beim stateless Gateway auf (Name Mapping) und daher können die dort beschriebenen Algorithmen übernommen werden.
Abschließend soll noch erwähnt werden, daß CMIS-M-SET-, M-CREATE- und M-DELETE-Requests direkt auf SNMP-SetRequests abgebildet werden müssen.

Komponenten für die Replikation der Internet-MIB

Es gibt zwei verschiedene Möglichkeiten, wer für die Replikation der Internet-MIB in der lokalen Gateway-MIB verantwortlich ist:

[5] Also eine Instanz der ISO-Klasse, die diejenige SNMP-Gruppe repräsentiert, die die SNMP-Tabelle enthält (So ist eine Instanz der Klasse „tcp“ für das Replizieren der SNMP-Tabelle „tcpConnTable“ und damit für das Kreieren bzw. Löschen von Instanzen der Klasse „tcpConnEntry“, verantwortlich).

[6] Dabei handelt es sich um eine spezielle Abfolge von SNMP GetNextRequest-PDUs, die jeweils genau eine Tabelle durchlaufen.

1. Jede Instanz eines Managementobjekts bzw. jedes Attribut in der Gateway-MIB ist selbst für die Konsistenzsicherung mit der realen Ressource, die sie repräsentiert, verantwortlich. Das heißt, es muß für jede Instanz eines Managementobjekts bzw. jedes Attribut eine Strategie implementiert werden (siehe nächsten Abschnitt). Dazu ist pro Implementierung ein eigener Timer notwenig, der nach jedem Ablauf die Replikation der realen Ressource mittels SNMP-PDUs anstößt. Den Timeout bestimmt dabei die verwendete Strategie.

2. Es existiert eine zentrale Komponente, ein eigener Prozeß oder Task, an den sich die Instanzen von Managementobjekten bzw. Attribute anmelden bzw. abmelden können. Dabei erfolgt der „update“ eines angemeldeten Objekts jeweils nach einem Intervall, das von der verwendeten Strategie abhängt. Dazu wird ein Timer implementiert, der für jedes angemeldete Objekt überprüft, ob das Intervall abgelaufen ist und damit, ob es mittels SNMP-PDUs repliziert werden muß oder nicht.

Strategien für den Zeitpunkt der Replikation

Es wurde bisher mehrmals von einer Strategie gesprochen, die die Größe des Intervalls und damit den Zeitpunkt bestimmt, nachdem die reale Ressource mittels SNMP-PDUs in der lokalen Gateway-MIB repliziert werden soll. Wie schon öfters in dieser Diplomarbeit erwähnt, ist dieser Zeitpunkt von der Art der Managementinformation abhängig. So sollte eine ideale Strategie statische Managementinformation eher selten und dynamische Managementinformation dagegen sehr häufig abfragen, dabei aber möglichst Inkonsistenzen vermeiden:

1. Eine sehr schlechte, aber einfach zu implementierende Strategie ist, daß jedes Objekt nach einem festen, für alle gleichen, Intervall abgefragt wird (zum Beispiel 5 Sekunden). Daß diese Strategie wenig sinnvoll ist, braucht nicht mehr erwähnt zu werden, denn damit ist die dynamische Managementinformation in der Gateway-MIB sehr wahrscheinlich veraltet und statische Managementinformation wird unnötig oft aufgefrischt.

2. Eine bessere, wenn auch nicht optimale Lösung beschreibt folgende Strategie[7]: Für jedes Managementobjekt wird das Pollingintervall exponentiell an die Art seiner Managementinformation angepaßt. Das heißt, falls sich der Wert des Objekts verändert, wird das Intervall halbiert, andernfalls verdoppelt. So wird zu jedem Objekt, das repliziert werden soll, ein 32 Bit Wert gespeichert. In diesem 32 Bit Wert darf nur eine 1 vorkommen, der Rest besteht aus Nullen. Je nachdem, in welchem Bit diese eine 1 gerade vorkommt, wird ein Intervall festgelegt:

[7] nach [Ke96]

Bit 0	entspricht Intervallgröße von einer Sekunde
Bit 1	entspricht Intervallgröße von 2 Sekunden
Bit 2	entspricht Intervallgröße von 4 Sekunden
Bit n	entspricht Intervallgröße von 2^n Sekunden

Der Default-Wert kann beliebig gesetzt werden, zum Beispiel auf 8 Sekunden. Das heißt, das Objekt wird nun alle 8 Sekunden aktualisiert. Ändert sich der Objekt-Wert nach 8 Sekunden, so soll dieser 32 Bit Wert nach rechts geshiftet werden, womit das Intervall auf 4 Sekunden reduziert wird. Ebenso soll, falls sich nach 4 Sekunden keine Änderung im Objekt-Wert ergibt, der 32 Bit Wert nach links geschoben werden, das Intervall damit auf seine ursprüngliche Größe gehoben werden, usw.
Damit paßt sich das Intervall der Art der Managementinformation des Objekts an: Für statische Information bewegt sich die 1 nach links und damit wird das Aktualisierungintervall größer. Für dynamische Managementinformation pendelt dagegen die 1 im rechten Teil des 32 Bit Wertes, je nachdem, wie dynamisch das Verhalten des Objekts ist.
Diese Strategie bringt aber auch Gefahren mit sich, da plötzliche Veränderungen schlecht erkannt werden, dazu folgendes Beispiel: Die IP-Adresse (statische Managementinformation) wird geändert. Da sie seit langem nicht mehr verändert wurde, ist die 1 in dem 32 Bit Wert beispielsweise auf Position 20, dem entsprechen 2^{20} Sekunden = 12,1 Tagen. So kann es maximal 12 Tage dauern, bis die nächste Aktualisierung durchgeführt und damit die Änderung im Gateway vorgenommen wird. Bis dahin ist die Gateway-MIB, bezogen auf diese IP-Adresse, inkonsistent mit der Internet-MIB.

Behandlung von Notifikationen

Wie schon erwähnt, unterscheiden sich die Bearbeitungen von Notifikationen im stateful bzw. stateless Ansatz nicht. Sie erfolgt im nächsten Abschnitt.

Zusammenfassung

Dieser Abschnitt hat gezeigt, daß das Hauptproblem bei dem stateful Ansatz die Konsistenzsicherung der replizierten Gateway-MIB ist. Wie Anfangs des Kapitels schon erläutert, ist eine vollständige Konsistenz der Gateway-MIB gegenüber der realen Ressource in den SNMP-Agenten nicht möglich.
Damit stellt die eben diskutierte Strategie, in der die reale Ressource aufgrund ihres zeitlichen Verhaltens und damit aufgrund ihres Typs von Managementinformation, unterschiedlichen Intervall-Klassen zugeordnet wird, einen Kompromiß dar. Die Praxis wird zeigen, wie akzeptabel diese wirklich ist.

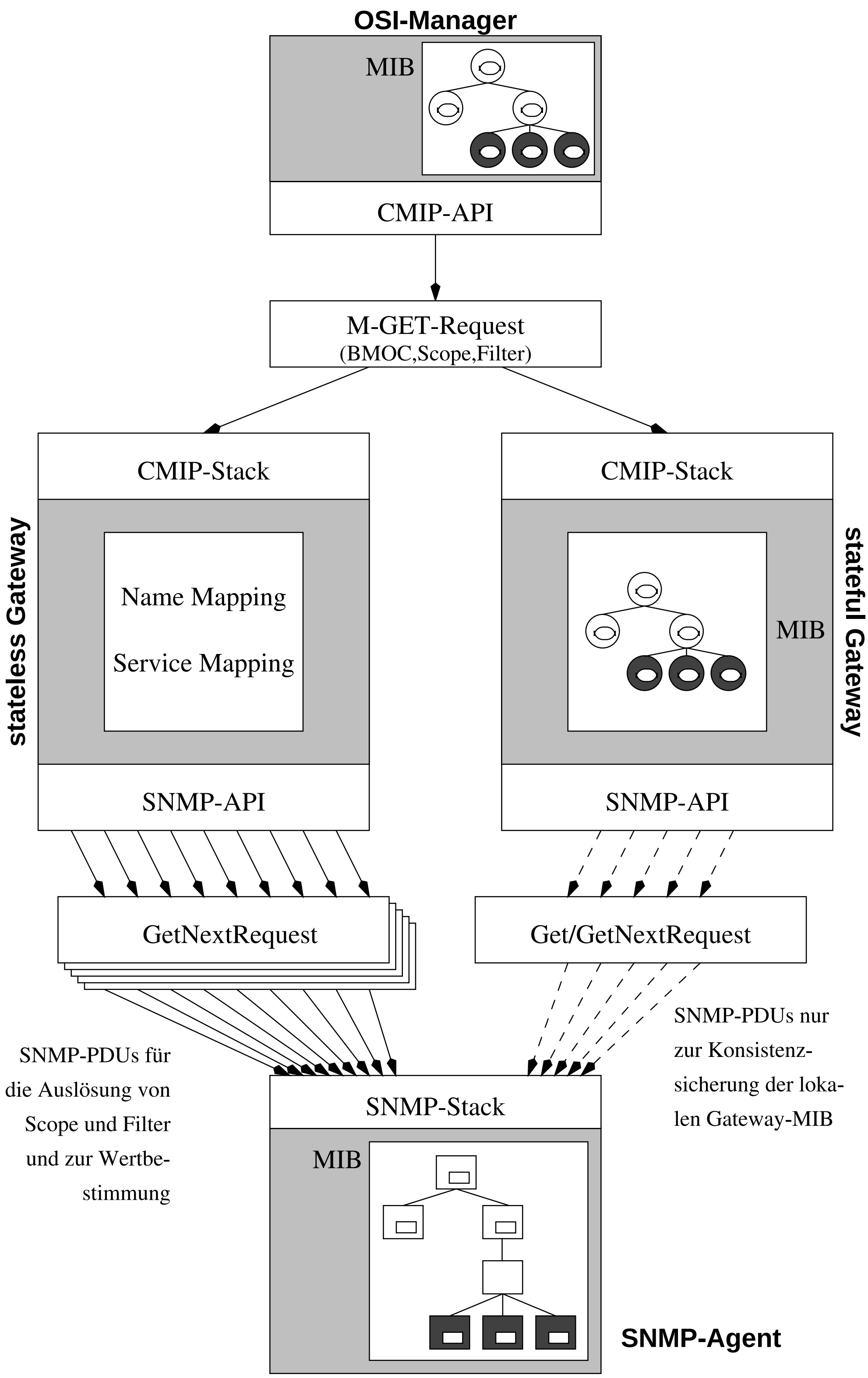


Abbildung 4.6: Die unterschiedliche Bearbeitung von CMIP-Anforderungen der beiden Gateways

Ein großer Vorteil des stateful Gateways liegt darin, daß damit bei der Behandlung von gescopten und gefilterten CMIP-Requests sehr hohe Performace, im Vergleich zum stateless Ansatz, erreicht werden können.
Das Beispiel in Abbildung 4.6 zeigt die Bearbeitung eines gescopten und gefilterten CMIS-M-GET-Requests. Das stateless Gateway auf der linken Seite benötigt zum einen für die Auflösung des Scopes und des Filters eine gewisse Anzahl an SNMP-GetNextRequest-PDUs. Zum anderen fordert die Bestimmung der SNMP-Variableninhalte weitere SNMP-GetRequest- bzw. GetNextRequest-PDUs. Das stateful Gateway speichert die komplette MIB des SNMP-Agenten in seiner lokalen Gateway-MIB. Dadurch werden beim Auflösen des Scopes und des Filters keine expliziten SNMP-Requests benötigt, da alle Informationen im Gateway vorhanden sind. Ebenso können die SNMP-Variableninhalte sofort aus den Attributen der Managementobjekte in der lokalen MIB gelesen werden. Dies fordert ebenfalls keine SNMP-Requests. Es werden aber sehr wohl SNMP-Requests für die Konsistenzsicherung dieser lokalen Gateway-MIB notwendig. Es werden beim stateful Gateway somit die SNMP-Requests auf einen gewissen Zeitraum verteilt. Diese Verteilung findet beim stateless Gateway nicht statt, es werden nur dann SNMP-Anforderungen ausgelöst, wenn sie auch wirklich notwendig sind. Damit werden im Falle eines gescopten und gefilterten M-GET-Requests eine sehr große Anzahl an SNMP-Anforderungen benötigt.

Weiterhin kann durch einen effizienten Einsatz von SNMP GetNextRequests- (SNMPv1) und GetBulkRequest-PDUs (SNMPv2) die Kommunikation von Gateway und SNMP-Agenten minimiert werden.
Zuletzt soll noch darauf hingewiesen werden, daß die generischen Notifikationen wie *changeAttributeValue*, *objectCreation* bzw. *objectDeletion* mit dem stateful Ansatz relativ einfach zu realisieren sind[8].

4.4.3 Die Behandlung von SNMP-Traps

Die eben vorgestellten stateless und stateful Gateways ermöglichen einem OSI-Manager das Management von SNMP-Agenten. Sie erlauben es aber einem SNMP-Agenten noch nicht, mit Hilfe von Traps/Notifikationen Ereignisse einem OSI-Manager mitzuteilen. Der folgende Abschnitt widmet sich deshalb dieser Funktionalität des Gateways.
Internet Traps/Notifikationen und Inform-Requests werden von der Internet SMI keiner speziellen Gruppe oder keinem speziellen Objekt zugeordnet, sie sind deshalb kein Teil des Informationsmodells, sondern ein Teil des Kommunikationsmodells. In der ISO/ITU SMI ist jedoch für eine abzuschickende Notifikation eine bestimmte Objektinstanz erforderlich. Daraus ergibt sich die Fragestellung, wel-

[8]In der *IBM TMN WorkBench for AIX* werden diese generischen Notifikationen standardmaßig für alle Klassen/Attribute bereitgestellt

che OSI Managemenobjekt-Klasse die Internet Traps/Notifikationen weiterleiten soll.
Dazu wird in [IIMCIMIBTR] eine generische Notifikation, „internetAlarm“, definiert, auf welche alle Internet Traps/Notifikationen abgebildet werden. Diese Notifikation muß noch einer Klasse zugeordnet werden. Einerseits bietet sich dazu die aus der Internet-MIB-2-Gruppe *system* abgeleitete Klasse *internetSystem* an. Andererseits wäre auch die in [IIMCPROXY] definierte Klasse *cmipSnmpProxy*-Klasse denkbar. Es wird aber auch von der Existenz von Instanzen dieser Klassen abhängen. So soll, wenn es keine Instanz der einen Klasse gibt, eine Instanz der anderen Klasse diese Aufgabe übernehmen[9]. Ein weiterer wichtiger Punkt ist, wie die Notifikationen behandelt werden, die für generische Klassen, damit auch für die aus den Internet-MIBs entstandenen Managementobjekt-Klassen, definiert sind. Beispiele dafür sind *objectCreation*- und *attributeValueChange*-Notifikationen. Da zu diesen Ereignissen keine korrespondierenden SNMP-Traps existieren, werden sie auch nicht vom SNMP-basierten Management registriert. Sollen sie dennoch unterstützt werden, müssen sie im Gateway simuliert werden. Dies könnte durch ein periodisches Abfragen von Internet-MIB Werten erfolgen, wobei aber die Werte zum Vergleich abgespeichert werden müssen. Dazu muß das Gateway stateful sein, deshalb sind solche Notifikationen nicht mit dem stateless Ansatz zu realisieren.

4.5 Zusammenfassung

Zu Beginn dieses Kapitels wurde festgestellt, daß für die Integration des Internet-Managements in die OSI-Architektur sowohl eine Informationsmodell- als auch eine Kommunikationsmodellabbildung notwendig ist. Für beide Abbildungen wurden zwei Möglichkeiten angegeben.
Dabei soll die Informationsmodellabbildung, aus schon bekannten Gründen, nach der direkten Übersetzung erfolgen.
Für die Kommunikationsmodellabbildung wurden der stateless bzw. der stateful Ansatz vorgestellt. Beide stellen einem OSI-Manager eine OSI-Sichtweise auf die SNMP-Ressourcen zur Verfügung, wobei ein stateful Gateway durch die Möglichkeit, die generischen Notifikationen zu unterstützen, ein wenig mehr Mächtigkeit aufweist. Dies wird aber mit möglichen, strategiebedingten Konsistenzverlusten bezahlt.
Fazit:
Ein optimales, idealisiertes Gateway würde ein Kompromiß eingehen, der die Vorteile jeder Lösung integriert und die Nachteile vermeidet: für dynamische Managementinformation würde es sich wie ein stateless Gateway, für statische Managementinformation würde es sich wie ein stateful Gateway verhalten.

[9] Siehe dazu die Implementierungsbeschreibung in 6.1.2.

Kapitel 5

Die Architektur des CMIP/SNMP Gateways

In diesem Kapitel soll die Architektur eines CMIP/SNMP Gateways erstellt werden. Dabei besteht die Aufgabe darin, die in Kapitel 4 vorgestellte Top-Down Sichtweise mit der in Kapitel 3 beschriebenen Bottom-Up Sichtweise zu verbinden. Das heißt, die in Kapitel 4 vorgestellten Informations- und Kommunikationsmodellabbildungen sollen auf die Agentenarchitektur der *IBM TMN WorkBench for AIX* aus Kapitel 3 projiziert werden. Die daraus resultierende Architektur soll im anschließenden Kapitel 6 implementiert werden.

5.1 Anforderungen

Die folgenden, allgemeinen Anforderungen soll die Gatewayarchitektur erfüllen:

- Das Gateway soll einem OSI-Manager eine OSI-Sichtweise auf SNMP-Ressourcen zur Verfügung stellen.
- CMIS-Anforderungen werden entweder im Gateway selbst bearbeitet oder auf SNMP-PDUs projiziert.
- Das Gateway soll den Zugriff auf die MIB-2 und die Systemagenten-MIB bereitstellen. Dazu werden diese beiden Internet-MIBs in GDMO-konforme MIBs übersetzt, wie es in den Abschnitten 3.1.1 und 4.2.1 („direkte Übersetzung") beschrieben wurde.
- Es soll zuerst versucht werden, die Kommunikationsmodellabbildung auf der Grundlage einer stateless Komponente zu realisieren, da dieser Ansatz eine einfachere Implementierung verspricht. Aufgrund daraus resultierender Erfahrungen sollen später stateful Ansätze integriert werden.
- Die SNMP-Traps sollen OSI-Event Notifikationen zugeordnet werden.

Weiterführende, aber nicht realisierte, Anforderungen an das CMIP/SNMP Gateway:

- Die automatische Erfassung aller SNMP-Agenten in einem Teilnetz durch das Gateway.
- Parallele Bearbeitung von CMIS-Anforderungen und damit die Realisierung eines Synchronisationskonzeptes zur Deadlock-Vermeidung.
- Unterstützung von generischen OSI-Events, zum Beispiel *changeAttributeValue* oder *objectCreation*.

5.2 Die Einbettung des Gateways in die Agentenarchitektur der *IBM TMN WorkBench for AIX*

Das Ziel dieses Abschnittes ist die Architektur eines CMIP/SNMP Gateways. Die Aufgabe dabei besteht darin, zu prüfen, inwieweit dieses Ziel (Top-Down Sichtweise aus Kapitel 4) mit den vorhandenen Werkzeugen (Bottom-Up Sichtweise aus Kapitel 3) realisierbar ist.
Mit der Überlegung, daß ein stateless Gateway eine einfachere Implementierung verspricht, sollte die erste Architektur des Gateways auf der Grundlage einer stateless Komponente beruhen. Da sehr schnell gezeigt werden kann (siehe unten), daß ein reines stateless Gateway nicht auf die Agentenarchitektur der *IBM TMN WorkBench for AIX* abgebildet werden kann, wird schließlich die Architektur eines CMIP/SNMP Gateways vorgestellt, welches sowohl stateless als auch stateful Ansätze integriert.

Es soll zuerst der Versuch gemacht werden, ein stateless Gateway in die Agentenarchitektur der *IBM TMN WorkBench for AIX* einzubetten (siehe auch Abbildung 5.1). Dazu wird es nötig sein, *Name Mapping* und *Service Mapping* (siehe 4.4.1) in diese Agentenarchitektur einzubinden. Das heißt, für einen am *Infratop* ankommenden CMIS-Request, der ein oder mehrere „remote Object(s)“ enthält, muß für diese(s) das *Name Mapping* und *Service Mapping* durchgeführt werden. Es müssen zuerst die Instanzen der Managementobjekte bestimmt werden, die im Scope und Filter ausgewählt werden, um anschließend diesen CMIS-Request auf diesen selektierten Instanzen auszuführen.

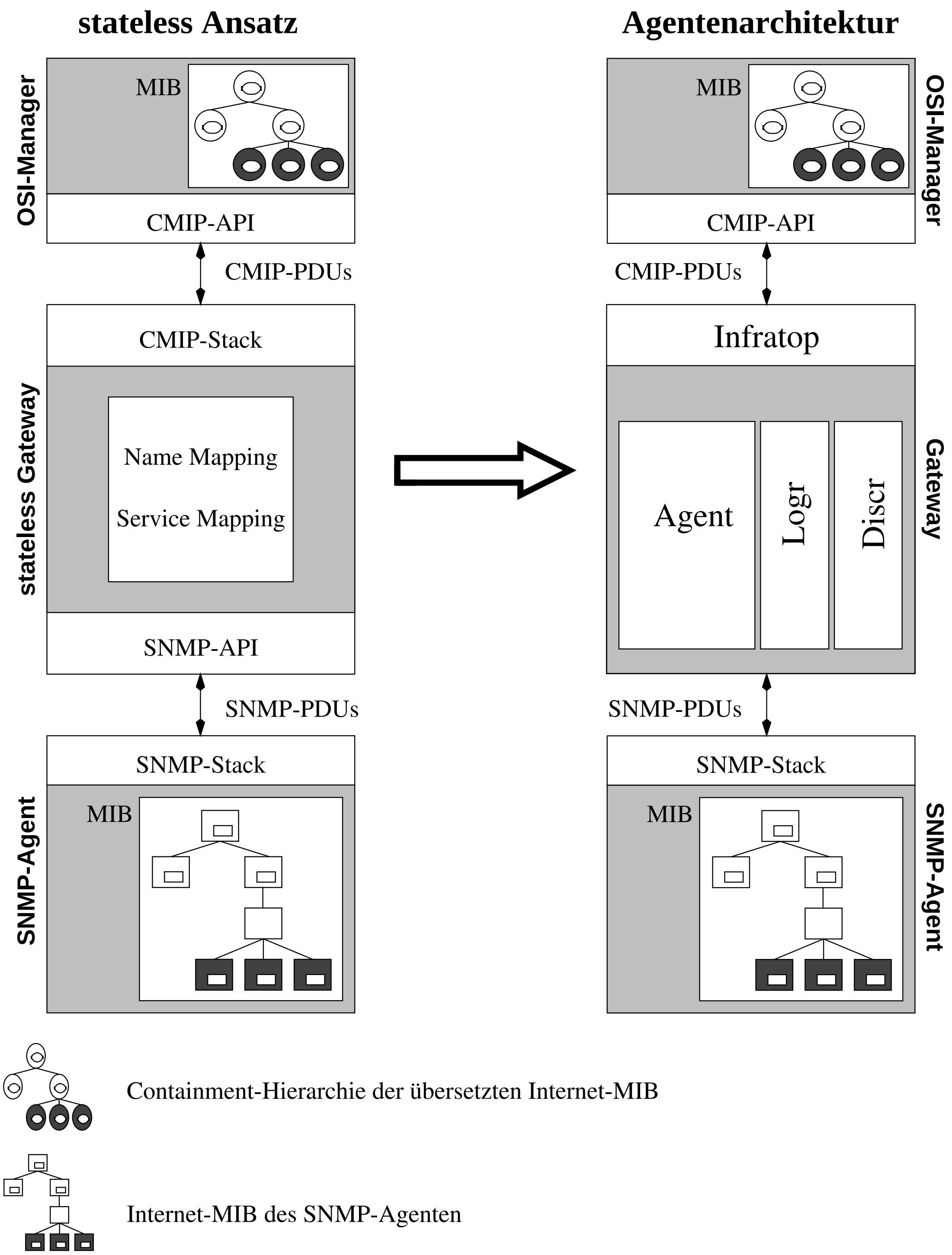


Abbildung 5.1: Die Einbettung eines stateless Gateways in die Agentenarchitektur

Das Ausführen von Scopes und die Replikation des Requests übernimmt aber in der IBM-Agentenarchitektur schon die *Naming and Replication*-Komponente. Diese speichert und verwaltet dazu alle existierenden Instanzen von Managementobjekten in der Containment-Hierarchie. Das bedeutet aber auch, daß jede Instanz eines Managementobjekts im OSI-Agenten zuerst kreiert werden muß, bevor auf sie, mit Hilfe eines CMIS-Requests, zugegriffen werden kann, siehe 3.2.4. Speziell für Klassen, die aus der Übersetzung der Internet-MIBs in GDMO-MIBs entstanden sind und damit SNMP-Ressourcen repräsentieren, heißt das:
Es müssen zuerst Instanzen dieser Klassen kreiert und damit (unter anderen) die Containment-Hierarchie im *Infratop* aufgebaut werden, bevor auf diese „remote Objects" zugegriffen werden kann. Diese „remote Objects" repräsentieren definitionsgemäß SNMP-Gruppen und SNMP-Tabellenzeilen (siehe „direkte Übersetzung", 4.2.1).
Es müssen für alle existierenden SNMP-Gruppen entsprechende Instanzen kreiert werden, die diese im Gateway repräsentieren.
Die Existenz von Tabellenzeilen und damit die Möglichkeit, eine entsprechende Instanz zu kreieren, kann nur dadurch bestimmt werden, daß mittels einer oder mehrerer SNMP-PDU(s) der Index dieser Tabellenzeile herausgefunden wird. Dieser Index wird definitionsgemäß zu dem *Relative Distinguished Name* der Instanz. Damit wird aber Managementinformation aus der MIB des SNMP-Agenten im Gateway gespeichert. Da eine Tabellenzeile auch von dem SNMP-Agenten gelöscht bzw. eine neue erstellt werden kann, muß das Gateway zusätzlich diese Tabelle regelmäßig auf neue bzw. veraltete Zeilen hin überprüfen (siehe 5.2.2).
Dies steht aber im Widerspruch zu dem stateless Ansatz, welcher ja besagt, daß keine Managementinformation aus dem SNMP-Agent gespeichert werden soll/darf, damit auch nicht die Existenz von Instanzen von Managementobjekten, die SNMP-Tabellenzeilen repräsentieren. Es soll jede Anforderung von einer höheren Hierarchiestufe (=OSI-Manager) auf die darunterliegende Hierarchiestufe und damit auf den SNMP-Agenten abgebildet werden. Aus diesen Gründen folgt, daß ein reines stateless Gateway wegen der Repräsentation der SNMP-Tabellenzeilen durch Instanzen von Managementobjekten und damit dem Speichern von Managementinformation aus der MIB des SNMP-Agenten, nicht auf die Agentenarchitektur der *IBM TMN WorkBench for AIX* abgebildet werden kann.

Fazit: Das Kreieren von Instanzen, die SNMP-Ressourcen repräsentieren, steht im Widerspruch zum stateless Ansatz. Speziell wird bei der Repräsentation von SNMP-Tabellenzeilen in der Gateway-MIB durch Instanzen Managementinformation gespeichert (der Index der jeweiligen Zeile wird zum Relative Distinguished Name). Die Besonderheit von SNMP-Gruppen, daß sie in einem SNMP-Agenten für die gesamte Lebensdauer existieren, ermöglicht es, die Informationen für das Kreieren von Instanzen der Klassen, die diese SNMP-Gruppen repräsentieren, direkt aus der MIB-Definiton (und diese ist in der *Metadata Database* gespeichert) zu lesen. Dadurch kann es vermieden werden, Managementinforma-

tion vom SNMP-Agenten abzufragen und im Gateway zwischenzuspeichern, siehe 5.2.1.
Der Zugriff auf die Attribute in den Instanzen, die die SNMP-Variablen repräsentieren, kann aber schließlich nach dem stateless Ansatz erfolgen, das heißt, bei einem lesenden Zugriff soll der Wert mittels einer SNMP-GetRequest-PDU bzw. bei einem schreibenden Zugriff soll der Wert mittels einer SNMP-SetRequest-PDU, direkt von bzw. in dem SNMP-Agenten gelesen bzw. geschrieben werden.
Die Agentenarchitektur der *IBM TMN WorkBench for AIX* verlangt, daß eine vollständige Containment-Hierarchie existiert, das heißt, jede im SNMP-Agenten gerade vorhandene Ressource muß aktuell in der Gateway-MIB durch Instanzen und deren Attribute repräsentiert werden.
Um nicht von einem reinen stateless Gateway zu einem reinen stateful Gateway überzugehen, liegt eine Gatewayarchitektur nahe, die sowohl stateless als auch stateful Ansätze integriert: All jene SNMP-Ressourcen, welche aufgrund der Einschränkungen der *IBM TMN WorkBench for AIX* nicht direkt auf SNMP-PDUs abgebildet werden können (dem entsprechen SNMP-Gruppen und SNMP-Tabellenzeilen), weil sie durch Instanzen im Gateway repräsentiert werden müssen (und somit die Containment-Hierarchie aufbauen), sollen nach dem stateful Ansatz implementiert werden. Alle anderen SNMP-Ressourcen (dem entsprechen alle SNMP-Variablen, sie werden durch Attribute in den OSI-Klassen dargestellt) sollen nach dem stateless Ansatz realisiert werden, das heißt, der Wert eines OSI-Attributs soll direkt aus dem SNMP-Agenten gelesen werden. Diese Gatewayarchitektur stellt Abbildung 5.2 dar.
Die Abbildung zeigt, daß im Gateway Instanzen für SNMP-Gruppen und SNMP-Tabellenzeilen existieren. Die SNMP-Tabelle wird regelmäßig überprüft (1), ob die in der MIB durch Instanzen repräsentierten Tabellenzeilen aktuell sind bzw. ob neue Tabellenzeilen entstanden sind und damit neue Instanzen kreiert werden müssen (2). Die Attributwerte werden bei jedem Zugriff direkt mittels SNMP-GetRequest-PDUs aus den entsprechenden SNMP-Variablen im SNMP-Agenten gelesen (3).
Lösungsmöglichkeiten für die Repräsentation von SNMP-Gruppen bzw. -Tabellenzeilen im Gateway wurden bereits in der Begründung, daß ein reines stateless Gateway nicht in die Agentenarchitektur eingebettet werden kann, angesprochen. Dies soll aber in den nächsten beiden Abschnitten detailliert vertieft werden:

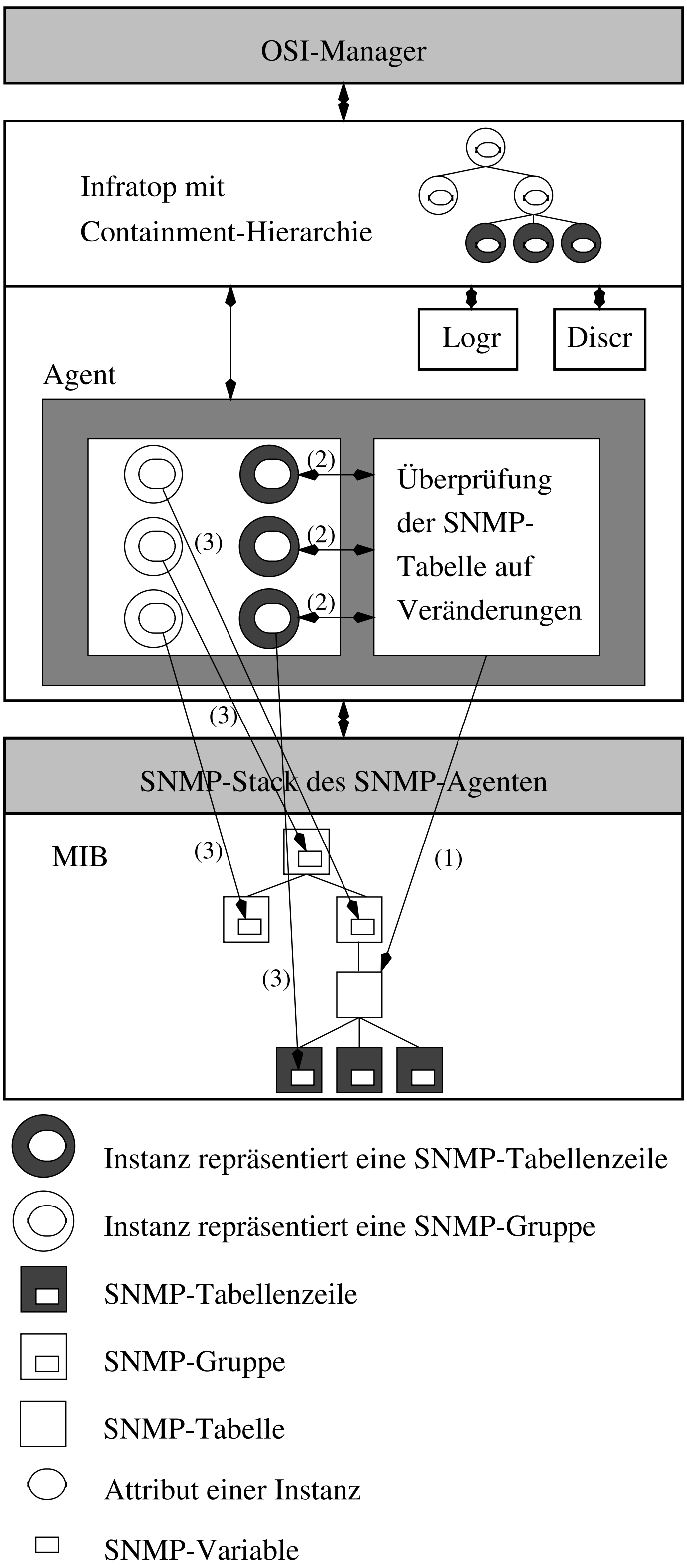


Abbildung 5.2: Die vorläufige Architektur des Gateways

5.2.1 Die Repräsentation von SNMP-Gruppen im Gateway

Die SNMP-Gruppen werden in den Internet-MIBs definiert. Sie können nicht kreiert oder gelöscht werden und existieren damit für die gesamte Lebensdauer der jeweiligen SNMP-Agenten.
Die SNMP-Gruppen werden, durch die Übersetzung von Internet-MIBs in GDMO-MIBs, auf OSI-Klassen abgebildet. Dabei werden auch NAME BINDINGs erstellt, die die Lage der Instanzen dieser Klassen in der Containment-Hierarchie festlegen. Aufgrund der fehlenden Enthaltenseinshierarchie im Internet-Informationmodell, wird bei der „direkten Übersetzung", siehe 4.2.1, der Registrierungsbaum der SNMP-Gruppen auf diese NAME BINDINGS und damit auf die OSI-Containment-Hierarchie abgebildet. Anders formuliert heißt das, die NAME BINDINGs der OSI-Klassen, die aus der Übersetzung der SNMP-Gruppen entstanden sind, stellen die Lage dieser SNMP-Gruppen im Internet-Registrierungsbaum dar. Da die SNMP-Gruppen dauerhaft existent sind, ist die Gültigkeit dieser NAME BINDINGs nicht eingeschränkt. Damit können diese NAME BINDINGs direkt nach der Übersetzung in die *Metadata Database* eingetragen werden, stehen somit dem Gateway und sämtlichen OSI-Managern ständig zur Verfügung und müssen nicht explizit aus den jeweiligen SNMP-Agenten gelesen werden.
Sobald nun ein neuer SNMP-Agent mit Hilfe des Gateways einem OSI-Manager bereitgestellt werden soll, müssen zuerst die Instanzen dieser Klassen, welche die SNMP-Gruppen repräsentieren, kreiert werden (siehe 3.2.4). Dies kann nun rekursiv mit Hilfe der NAME BINDINGs erfolgen: Am Beispiel des Registrierungsbaumes der SNMP-Gruppen der MIB-2 wird ersichtlich, daß die Gruppe „mib-2" allen anderen MIB-2 Gruppen übergeordnet ist, siehe Abbildung 5.3. Bei der Übersetzung dieser MIB-2 in eine GDMO-MIB entstehen einerseits für die SNMP-Gruppe „mib-2" ein NAME BINDING mit der SUPERIOR OBJECT CLASS „system" und andererseits für alle anderen SNMP-Gruppen dieser MIB-2 jeweils ein NAME BINDING mit der SUPERIOR OBJECT CLASS „mib2". Als SUBORDINATE OBJECT CLASS erhalten alle NAME BINDINGs den Namen der jeweils aus der Gruppe entstandene OSI-Klasse, siehe Abbildung 5.4.
Nun können, beginnend bei der Klasse „system", alle NAME BINDINGs aus der *Meatdata Database* bestimmt werden, welche als SUPERIOR OBJECT CLASS diese Klasse „system" besitzen. Dafür existiert in unserem Beispiel nur ein NAME BINDING: dieses enthält als SUBORDINATE OBJECT CLASS die Klasse „mib2", welche aus der Übersetzung der MIB-2 Gruppe „mib-2" entstanden ist. Es kann nun für diese Klasse eine Instanz kreiert werden. Anschließend werden alle NAME BINDINGs mit dieser Klasse „mib2" als SUPERIOR OBJECT CLASS bestimmt. Diese damit erhaltenen NAME BINDINGs enthalten wiederum jeweils als SUBORDINATE OBJECT CLASS alle Klassen, die die restlichen MIB-2 Gruppen repräsentieren, für die nun entsprechend eine Instanz kreiert werden kann.
An diesem Beispiel wird der angewandte, rekursive Algorithmus für den Aufbau

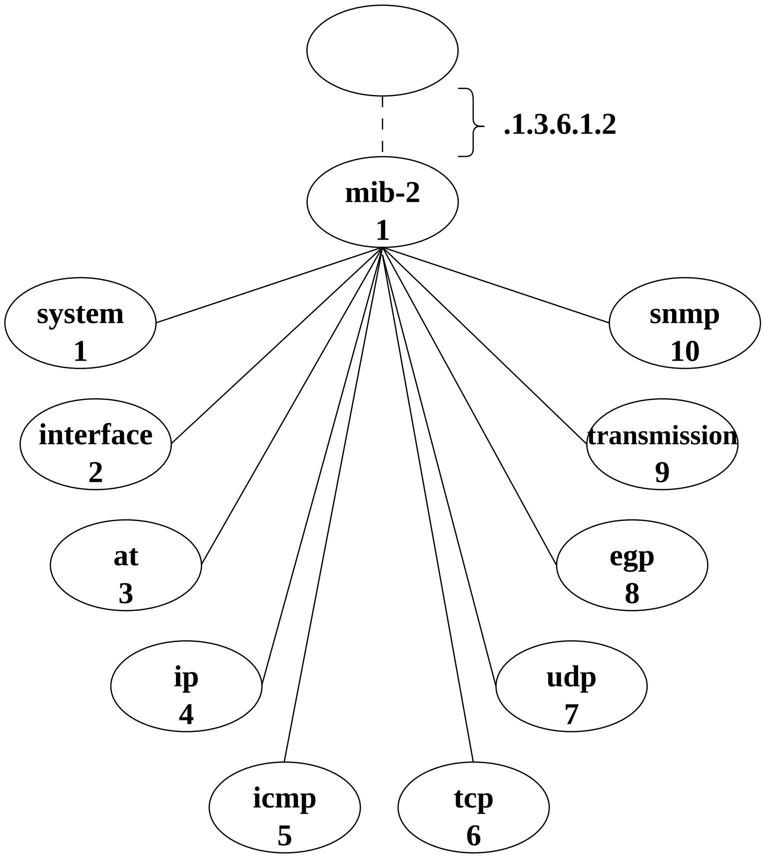


Abbildung 5.3: Die Gruppen der MIB-2

der Containment-Hierarchie für die Klassen, die die SNMP-Gruppen repräsentieren, im Gateway deutlich:
Für eine gegebene Managementobjekt-Klasse bestimme aus der *Metadata Database* alle NAME BINDINGs mit dieser Klasse als SUPERIOR OBJECT CLASS. Kreiere für jede in diesen NAME BINDINGs enthaltenen SUBORDINATE OBJECT CLASSes eine Instanz und rufe jeweils mit dieser SUBORDINATE OBJECT CLASS die Rekursion erneut auf.
Der Algorithmus beginnt dabei mit der OSI-Klasse „system“, da diese als Wurzel in der Enthaltenseinshierarchie dient und daher im IIMC-Übersetzungsalgorithmus (siehe 3.1.1) als „Root“-Klasse für die NAME BINDINGs definiert wird [IIMCIMIBTR].

Ergebnis: Für jeden neu zu managenden SNMP-Agenten kann das Gateway anhand der in der *Metadata Database* gespeicherten NAME BINDINGs sofort seine lokale Gateway-MIB um die Containment-Hierarchie der Instanzen der Klassen, die SNMP-Gruppen repräsentieren, erweitern, ohne auf Informationen aus der MIB dieses SNMP-Agenten zugreifen zu müssen.

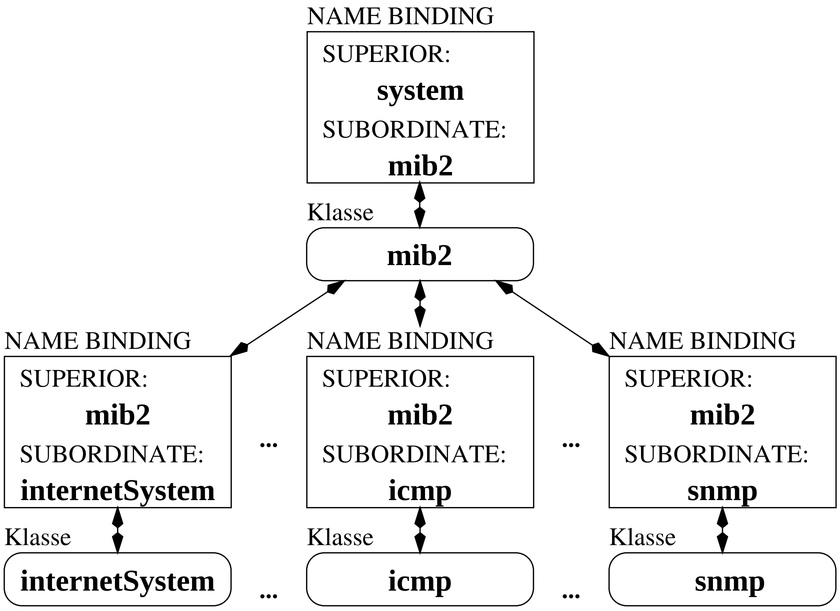


Abbildung 5.4: Einige Klassen und NAME BINDINGs aus der Übersetzung der MIB-2

Auf die Attribute in diesen Instanzen kann nun beliebig zugegriffen werden, wobei die Werte direkt aus den entsprechenden SNMP-Agenten gelesen werden.

5.2.2 Die Repräsentation von SNMP-Tabellenzeilen im Gateway

Die bisherige Architektur des Gateways erlaubt den Aufbau der Containment-Hierarchie für die Instanzen, die SNMP-Gruppen repräsentieren. Der Zugriff auf die Attribute dieser Instanzen von Managementobjekten erfolgt, wie schon oben erläutert, nach dem stateless Prinzip.
Für die vollständige OSI-Sichtweise auf SNMP-Ressourcen fehlen noch die Instanzen von Managementobjekt-Klassen, welche SNMP-Tabellenzeilen repräsentieren. Sobald diese Instanzen kreiert werden, werden die MIBs der SNMP-Agenten komplett in die lokale Gateway-MIB integriert und so für das Management eines OSI-Managers bereitgestellt.
Bei der „direkten Übersetzung“ wird, wie gerade erwähnt, jede SNMP-Tabellenzeile einer Instanz einer Managementobjekt-Klasse zugeordnet. Dabei erhält die Instanz als *Relative Distinguished Name* den Index der SNMP-Tabellenzeile. Dieser Index kann nur anhand von einer oder mehreren SNMP-GetRequest-PDU(s) bestimmt werden, das heißt, es muß Managementinformation aus der MIB des SNMP-Agenten im Gateway gespeichert werden.
Weiterhin wird eine Veränderung in einer SNMP-Tabelle, also dem Entstehen bzw. Terminieren von Tabellenzeilen, im allgemeinen nicht von einem SNMP-Agent ei-

nem SNMP-Manager gemeldet. Das heißt, das CMIP/SNMP Gateway, welches diese SNMP-Tabelle durch Instanzen von Managemenobjekt-Klassen (die eben die Tabellenzeilen repräsentieren), für das Management durch einen OSI-Manager bereitstellen soll, erhält keine Mitteilung über Veränderungen in der SNMP-Tabelle. Es ist also die Aufgabe des Gateways, dafür zu sorgen, daß für jede zu managende SNMP-Tabelle die Instanzen in der lokalen Gateway-MIB konsistent zu den realen Ressourcen (Tabellenzeilen) sind, siehe auch 3.2.4. Das heißt, für jede Tabellenzeile im SNMP-Agent muß genau eine, diese repräsentierende Instanz eines Managementobjekts in der Gateway-MIB existieren. Sobald eine neue Tabellenzeile erstellt bzw. gelöscht wird, muß auch die entsprechende Instanz im Gateway kreiert bzw. gelöscht werden. Dies ist aber wieder abhängig vom Typ der Tabellenzeile: dynamische Tabellen müssen stetig, statische Tabellen dagegen weniger häufig, auf Veränderungen hin überprüft werden. Das Gateway hat also zwei Problematiken zu lösen, siehe auch 4.4.2:

1. Jede SNMP-Tabelle muß auf Veränderungen hin überprüft werden, um dementsprechend Instanzen kreieren bzw. löschen zu können.

2. Es muß für jede SNMP-Tabelle die Art dieser Managementinformation bestimmt werden, um entscheiden zu können wie häufig bzw. nach welchen Intervallen diese SNMP-Tabelle überprüft werden soll.

Im Abschnitt 4.4.2 werden 2 Möglichkeiten angegeben, von wem die erste Problematik, die Replikation der SNMP-Tabelle in der Gateway-MIB, ausgeführt werden kann. Der erste Fall, daß jede Instanz selbst für die Konsistenzsicherung mit der realen Ressource verantwortlich ist, kann hier aus folgenden Gründen nicht realisiert werden:
Es wäre für jede Instanz notwendig, einen eigenen Timer zu implementieren, der nach einem bestimmten Intervall die Replikation ausführt. Da aber jede Instanz in der *IBM TMN WorkBench for AIX* mittels *Callbacks* kodiert wird und Callbacks nur aufgerufen werden, sobald bestimmte Situationen eintreffen, existiert keine geeignete Möglichkeit, einen Timer regelmäßig zu überprüfen. Denkbar wäre auch ein eigener Thread, in welchen der Timer abläuft. Dies hat aber den Nachteil, daß für eine große Anzahl an SNMP-Tabellen die Anzahl der maximal möglichen Threads überschritten werden kann und somit nicht alle SNMP-Tabellen überwach- und steuerbar wären. Bei dieser Diplomarbeit war aber noch ein anderer Grund ausschlaggebend dafür, daß dieser Fall nicht implementiert wurde: Es gab keine Installation von DCE und damit keine Möglichkeit, Threads zu realisieren.
Weiterhin ist es software-technisch sinnvoll, komplexe und häufig auftretende Module in eigenen Komponenten zu realisieren.
Daher wurde in diesem Gateway die 2. Möglichkeit in Betracht gezogen: Es wird eine zentrale Komponente (mit dem Namen „Global Polling“) eingeführt, an welcher die SNMP-Tabellen angemeldet werden.

Die Aufgabe der *Global Polling*-Komponente ist es, für jede registrierte SNMP-Tabelle einerseits zu überprüfen, ob neue Tabellenzeilen in dieser Tabelle entstanden sind und dementsprechend neue Instanzen in der Gateway-MIB zu kreieren. Andererseits muß diese Komponente feststellen, ob Tabellenzeilen gelöscht wurden, um die entsprechenden Instanzen in der Gateway-MIB zu löschen. Zusammengefaßt ist das *Global Polling* für die Konsistenzsicherung der Instanzen in der Gateway-MIB mit der realen Ressource (Tabellenzeilen) in den SNMP-Agenten verantwortlich.
Um zu entscheiden, nach welchen Intervallen, also zu welchen Zeitpunkten, die jeweilige Tabelle auf Veränderungen hin überprüft werden soll, wird die 2. Strategie aus 4.4.2 herangezogen:
Für jede angemeldete SNMP-Tabelle existiert ein 32 Bit Wert. Jede Bit-Position stellt dabei ein Intervall dar, mit der Länge $2^{Bit-Position}$ Sekunden. Falls eine Überprüfung, nach einem so definierten Zeitpunkt, eine Veränderung in einer Tabelle feststellt, wird dieser 32 Bit Wert um eine Position nach rechts geschoben, womit die Länge des Intervalls halbiert wird. Entsprechend wird die Länge des Intervalls verdoppelt, falls keine Veränderung bei einer Überprüfung einer Tabelle festgestellt werden kann. Somit pendelt dieser 32 Bit Wert und damit die Intervallgröße um annähernd dem zeitlichen Verhalten der jeweiligen Tabelle.
Eine SNMP-Tabelle hat ähnliche Eigenschaften wie eine SNMP-Gruppe: Sie existiert für die gesamte Lebensdauer des SNMP-Agenten. Der Unterschied zwischen einer SNMP-Gruppe und einer SNMP-Tabelle im Gateway ist die Repräsentation. SNMP-Gruppen werden von Instanzen von Managementobjekt-Klassen repräsentiert. Eine Komponente für eine SNMP-Tabelle erscheint im Gateway nicht. Es existieren lediglich Managementobjekt-Klassen und NAME BINDINGs für die Tabellenzeilen dieser SNMP-Tabellen. Damit stellt sich die Frage, wie und wann eine SNMP-Tabelle an der *Global Polling*-Komponente angemeldet wird.
Aufgrund der eben erwähnten Eigenschaft, daß die Lebensdauer einer SNMP-Tabelle gleich der des Agenten ist, kann die Tabelle zum gleichen Zeitpunkt an der *Global Polling*-Komponente angemeldet werden, zu der auch die SNMP-Gruppen instanziiert werden. Dies entspricht also dem Zeitpunkt, an dem der SNMP-Agent für das Management durch einen OSI-Manager bereitgestellt werden soll, siehe Abbilding 5.5. In dem, im vorherigen Abschnitt vorgestellten, rekursiven Algorithmus zur Instanziierung von Managementobjekt-Klassen von SNMP-Gruppen werden alle NAME BINDINGs aus der *Metadata Database* gelesen. Es werden aber nur jene NAME BINDINGs verwendet, die sich auf die Klassen beziehen, welche SNMP-Gruppen repräsentieren. Dieser Algorithmus kann nun so erweitert werden, daß für alle gelesene NAME BINDINGs, die sich auf die Klassen beziehen, welche SNMP-Tabellenzeilen repräsentieren, die dazugehörige SNMP-Tabelle an der *Global Polling*-Komponente angemeldet wird. Dabei ergibt sich die OID einer SNMP-Tabelle aus der OID einer in dieser Tabelle enthaltenen Tabellenzeile durch entsprechendes Abschneiden eines bestimmten Suffix.

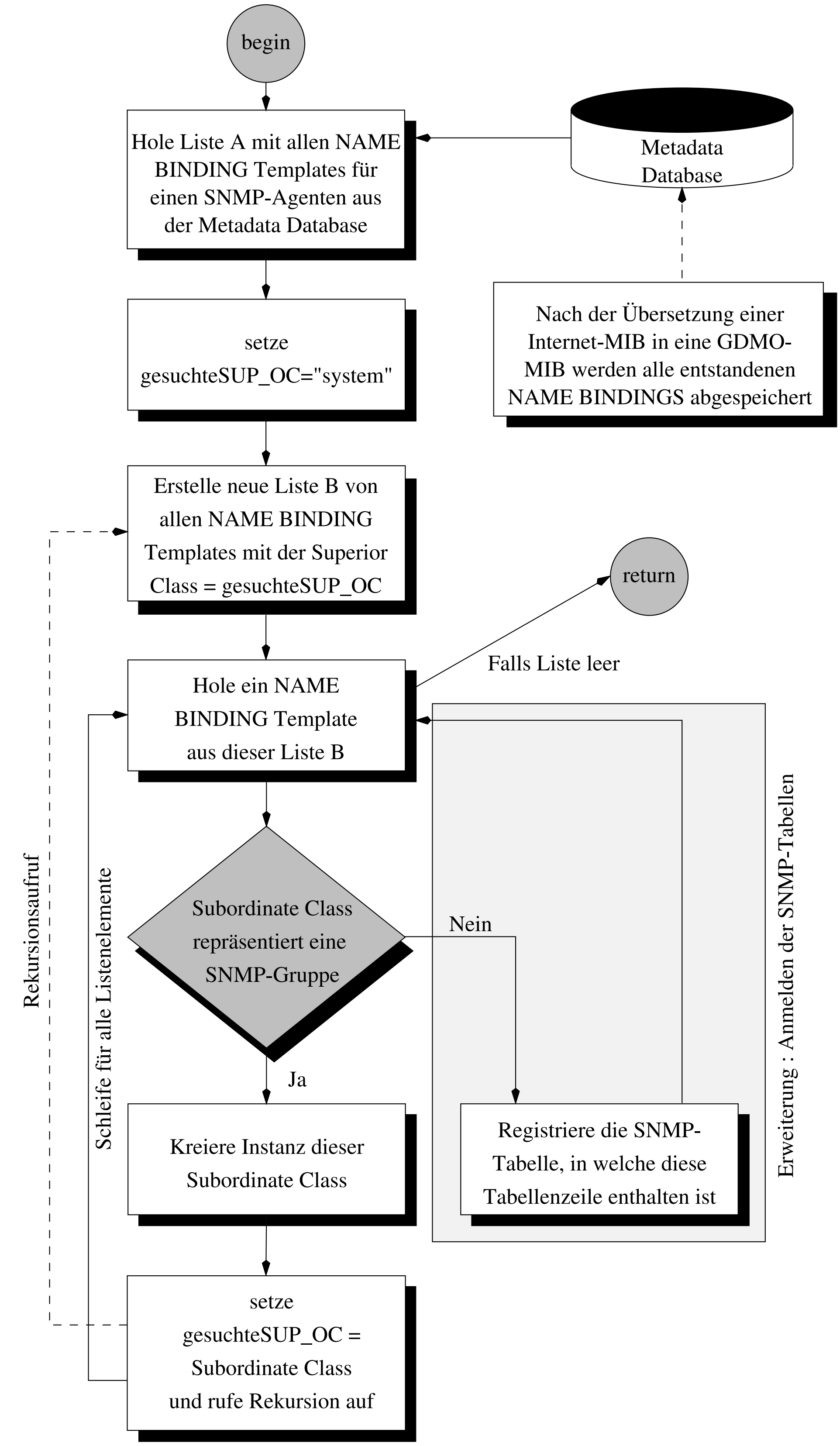


Abbildung 5.5: Der Algorithmus zum Aufbau der Containmenthierarchie

In SNMP-Tabellen kann es möglich sein, mit Hilfe von SNMP-SetRequest-PDUs und damit von einem SNMP-Manager aus, Tabellenzeilen zu Erstellen bzw. zu Löschen (als ungültig zu deklarieren). Indem das Gateway das Kreieren bzw. das Löschen von Instanzen auf diese SNMP-SetRequest-PDUs abbildet, wird es einem OSI-Manager gestattet, Tabellenzeilen in einem SNMP-Agent entsprechend zu erstellen bzw. zu löschen.

Die Abbildung 5.6 zeigt die bisherige Architektur und damit die Funktionsweise des CMIP/SNMP Gateways:

1. In der *Metadata Database* werden alle aus der Übersetzung der Internet-MIBs in OSI-MIBs entstandenen NAME BINDINGs eingetragen.

2. Sobald ein neuer SNMP-Agent für das OSI-Management bereitgestellt werden soll, werden nach 5.2.1 Instanzen für alle SNMP-Gruppen kreiert.

3. In dem in 5.2.1 vorgestellten Algorithmus werden alle in den *Metadata Database* vorhandenen NAME BINDINGs gelesen, es werden aber nur die benutzt, welche sich auf Klassen beziehen, die SNMP-Gruppen repräsentieren. Zusätzlich soll dieser Algorithmus noch für die NAME BINDINGs, welche sich auf die Klassen beziehen, die aus SNMP-Tabellenzeilen entstanden sind, die entsprechende SNMP-Tabelle in der *Global Polling*-Komponente angemeldet werden.

4. Damit, durch diese Anmeldung, werden die Instanzen in der Gateway-MIB, die die SNMP-Tabellenzeilen repräsentieren, kreiert.

5. Die *Global Polling*-Komponente überprüft schließlich regelmäßig, ob Veränderungen in der SNMP-Tabelle auftreten und damit, ob entsprechend Instanzen gelöscht bzw. kreiert werden müssen.

6. Der Zugriff auf Attribute in Instanzen wird direkt auf SNMP-PDUs übersetzt und damit der Wert aus der MIB des jeweiligen SNMP-Agenten gelesen bzw. gesetzt.

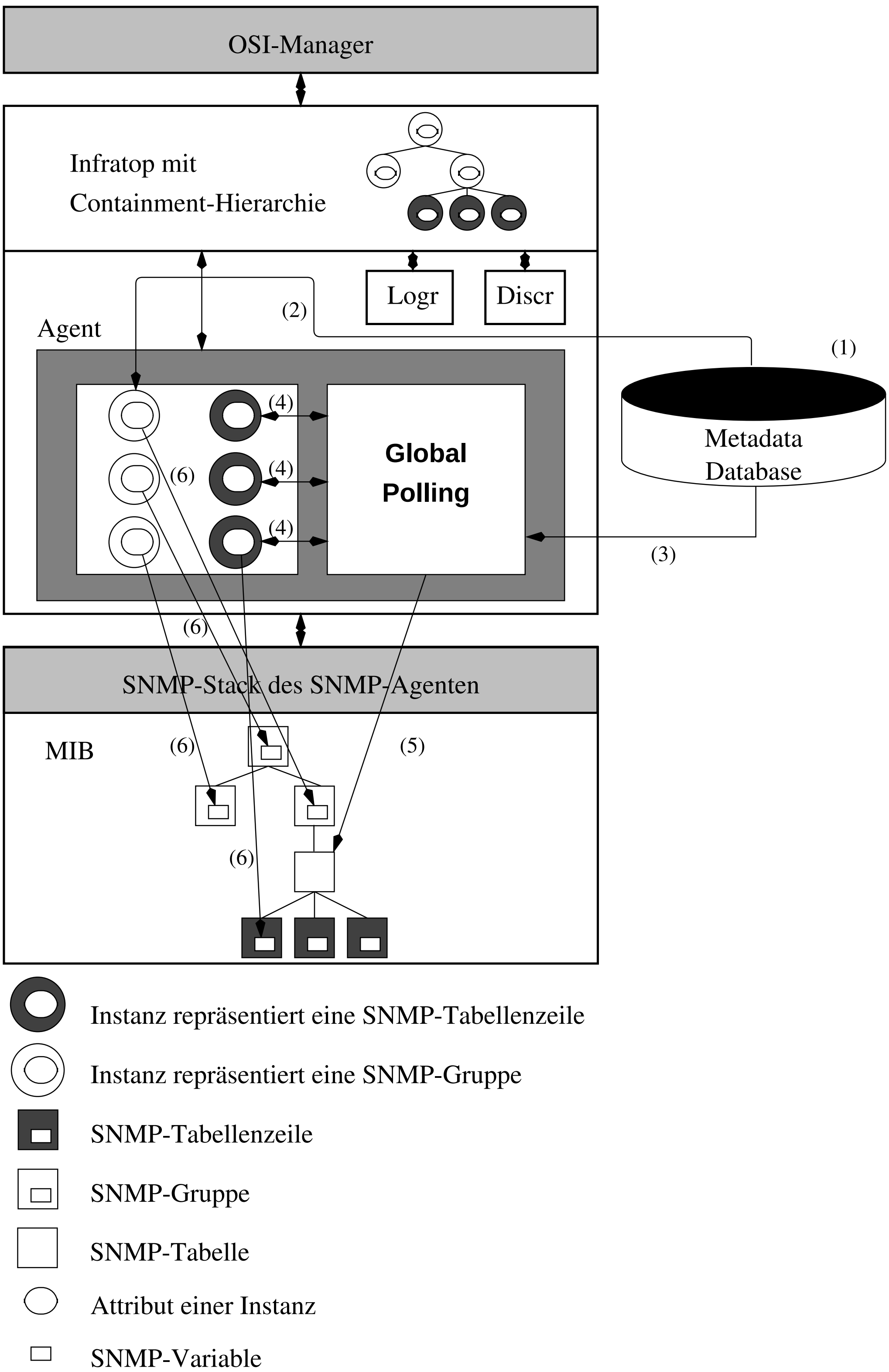


Abbildung 5.6: Die bisherige Gatewayarchitektur

5.2.3 Die Bereitstellung der SNMP-Manager-Funktionalität

Bei der bisherigen Gatewayarchitektur wurde noch nicht festgelegt, wie die Kommunikation des Gateways mit den SNMP-Agenten abläuft. Es wurde lediglich angemerkt, daß die OSI-Attribute ihren Wert direkt mittels SNMP-PDUs aus der MIB der jeweiligen SNMP-Agenten erhalten bzw. die *Global Polling*-Komponente die SNMP-Tabellen auf Veränderungen überprüfen kann.
Für die Einbindung der SNMP-Manager-Funktionalität ist es daher naheliegend, jedem Attribut bzw. der *Global Polling*-Komponente die Möglichkeit zu geben, selbstständig SNMP-PDUs zu verschicken bzw. die Antworten zu empfangen. Damit wäre in jedem Attribut bzw. in der *Global Polling*-Komponente ein eigener SNMP-Manager integriert. Das heißt aber auch, da jedes Attribut genau eine SNMP-Variable repräsentiert, daß für jede in den SNMP-Agenten vorhandenen Variablen ein SNMP-Manager im Gateway existiert. Damit ist aber ein sinnvoller Einsatz von SNMP-GetNextRequests- bzw. -GetBulkRequest-PDUs ausgeschlossen. Weiterhin wäre die Integration einer neuen SNMP-Programmierschnittstelle, der „Update“ einer vorhandenen Programmierschnittstelle oder das Einführen einer neuen SNMP Version nur durch das Verändern aller Attribute bzw. der *Global Polling*-Komponente möglich. Schließlich wäre es denkbar, die SNMP-Kommunikation des Gateways mit den SNMP-Agenten selbst für das Management bereitzustellen. Dazu müßten globale Datenstrukturen eingeführt werden, die von jedem SNMP-Manager im Gateway aktualisiert werden. Diese Datenstrukturen könnten Statistiken wie die Anzahl aller empfangenen, fehlerhaften SNMP-PDUs enthalten.
Die Diskussion dieses Ansatzes hat gezeigt, daß es sinnvoll ist, die SNMP-Programmierschnittstelle in eine eigene, globale Komponente im Gateway einzubinden: der *SNMP-API*. Sie besteht aus einer Instanz der C++-Klasse `SNMP` und kapselt in dieser die verwendete SNMP-Programmierschnittstelle. Dabei werden folgende Methoden der Klasse `SNMP` auf die Funktionen der Programmierschnittstelle abgebildet (siehe auch 6.1.1):

- `get`: Erzeugt je nach Bedarf SNMP-GetRequest-, SNMP-GetNextRequest- oder SNMP-GetBulkRequest-PDUs.

- `set`: Erzeugt eine SNMP-SetRequest-PDU.

- `specialWalk`: Gibt eine Liste mit allen Indizes der Tabellenzeilen einer SNMP-Tabelle zurück, die gerade aktuell im SNMP-Agenten existieren. Diese Methode wird von der *Global Polling*-Komponente benötigt, um feststellen zu können, ob Veränderungen in einer SNMP-Tabelle aufgetreten sind. Dementsprechend können Instanzen, die diese Tabellenzeilen im *spezifischen Agententeil* repräsentieren, kreiert bzw. gelöscht werden.

Die OSI-Attribute bzw. die *Global Polling*-Komponente kann nun durch entsprechendes Aufrufen dieser Methoden SNMP-PDUs auslösen und somit mit einem SNMP-Agenten kommunizieren.
Durch die Kapselung der verwendeten SNMP-Programmierschnittstelle ist es möglich, diese einfach auszutauschen und dabei nur an einer Stelle Veränderungen vornehmen zu müssen. So kann zum Beispiel sehr einfach eine neue SNMP-Version in das Gateway integriert werden.
Die Komponente *SNMP-API* bildet zusammen mit dem *SNMP-TrapController* (siehe 5.2.4) den SNMP-Manager des Gateways. Der SNMP-Manager kann nun seinerseits durch eine Instanz der OSI-Klasse „snmpComEntity“ gemanaged werden. Dabei sind Funktionalitäten denkbar, wie:

- die Anzahl der versendeten GetRequest-PDUs
- die Anzahl der versendeten SetRequest-PDUs
- die Anzahl der empfangenen GetResponse-PDUs
- die Anzahl der empfangenen Traps
- die Anzahl von fehlerhaften PDUs
- der Erstellen von Statistiken
- usw.

Hier könnte man sich die komplette Funktionalität der SNMP MIB-2 Gruppe „snmp“ vorstellen, aber durch die Mächtigkeit von CMIS auch weiterführende und komplexere Statistiken.

5.2.4 Die Behandlung von SNMP-Traps

Als letzte Aufgabe, um damit die Gatewayarchitektur zu vervollständigen, fehlt noch die Eigenschaft, SNMP Traps bzw. InformRequest-PDUs auf OSI-Notifikationen (siehe 3.2.1) abzubilden.
Diese Funktionalität ist aus dem Blickwinkel des TMN-Referenzmodells heraus die Aufgabe der *Q Adaptor Function (QAF)*. Es wird eine Schnittstelle (*m-Interface*) zwischen einer *Operations Systems Function (OSF)* und einer nicht-OSI-Komponente bereitgestellt. In diesem Fall handelt es sich bei dem *m-Interface* um eine Schnittstelle zwischen CMIP und SNMP, wobei die Kommunikation bei der Bearbeitung von Traps bzw. InformRequest-PDUs nur in eine Richtung abläuft, vom SNMP-Agenten zu dem Gateway (QAF). Das Gateway bildet die empfangenen Traps bzw. InformRequest-PDUs auf die in 3.1.1 vorgestellte generische Notifikation ab und löst deren Weiterleitung als Notifikation

einer Instanz eines Managementobjekts aus. Welche Instanz dies im speziellen sein soll, wird in 6.1.2 beschrieben.
Das *m-Interface* und damit die Möglichkeit, Traps bzw. InformRequest-PDUs zu empfangen kann mit der in 3.2.1 vorgestellten *Resource Access*-Komponente realisiert werden. Diese wird dazu in einer neuen Gatewaykomponente mit dem Namen *TrapController* integriert. Diese stellt einen relativ eigenständigen Bereich im Gateway dar, da die Kommunikation nur, von Traps bzw. InformRequest-PDUs ausgelöst, in Richtung *spezifischer Agententeil* abläuft. Der *TrapController* bildet zusammen mit der *SNMP-API* den SNMP-Manager des Gateways und kann mit einer Instanz der Klasse „snmpComEntity" selbst überwacht und gesteuert werden.

Die Abbildung 5.7 zeigt den vollständigen Überblick über die Architektur des CMIP/SNMP Gateways, dessen Implementierung in Kapitel 6 folgt. Die Zahlen in den Klammern entsprechen folgenden Bemerkungen:

1. In der *Metadata Database* werden alle aus der Übersetzung der Internet-MIBs in OSI-MIBs entstandenen NAME BINDINGs eingetragen.

2. Sobald ein neuer SNMP-Agent für das OSI-Management bereitgestellt werden soll, werden nach 5.2.1 die Instanzen für alle SNMP-Gruppen kreiert

3. und die SNMP-Tabellen in der *Global Polling*-Komponente angemeldet (siehe Algorithmus in Abbildung 5.5).

4. Damit, durch diese Anmeldung, werden die Instanzen in der Gateway-MIB, die die SNMP-Tabellenzeilen repräsentieren, kreiert.

5. Die *Global Polling*-Komponente überprüft schließlich regelmäßig, ob Veränderungen in der SNMP-Tabelle auftreten und damit, ob entsprechend Instanzen gelöscht bzw. kreiert werden müssen.

6. Der Zugriff auf die Attribute in den Instanzen wird auf Methodenaufrufe in der Komponente *SNMP-API* abgebildet. Diese Methodenaufrufe lösen schließlich die benötigten SNMP-PDUs aus.

7. Der relativ eigenständige *TrapController* hört den Port 162 ständig ab und kann somit Traps bzw. InformRequest-PDUs empfangen.

8. Diese werden bestimmten Instanzen von Mangementobjekten zugeordnet und auf OSI-Notifikationen abgebildet.

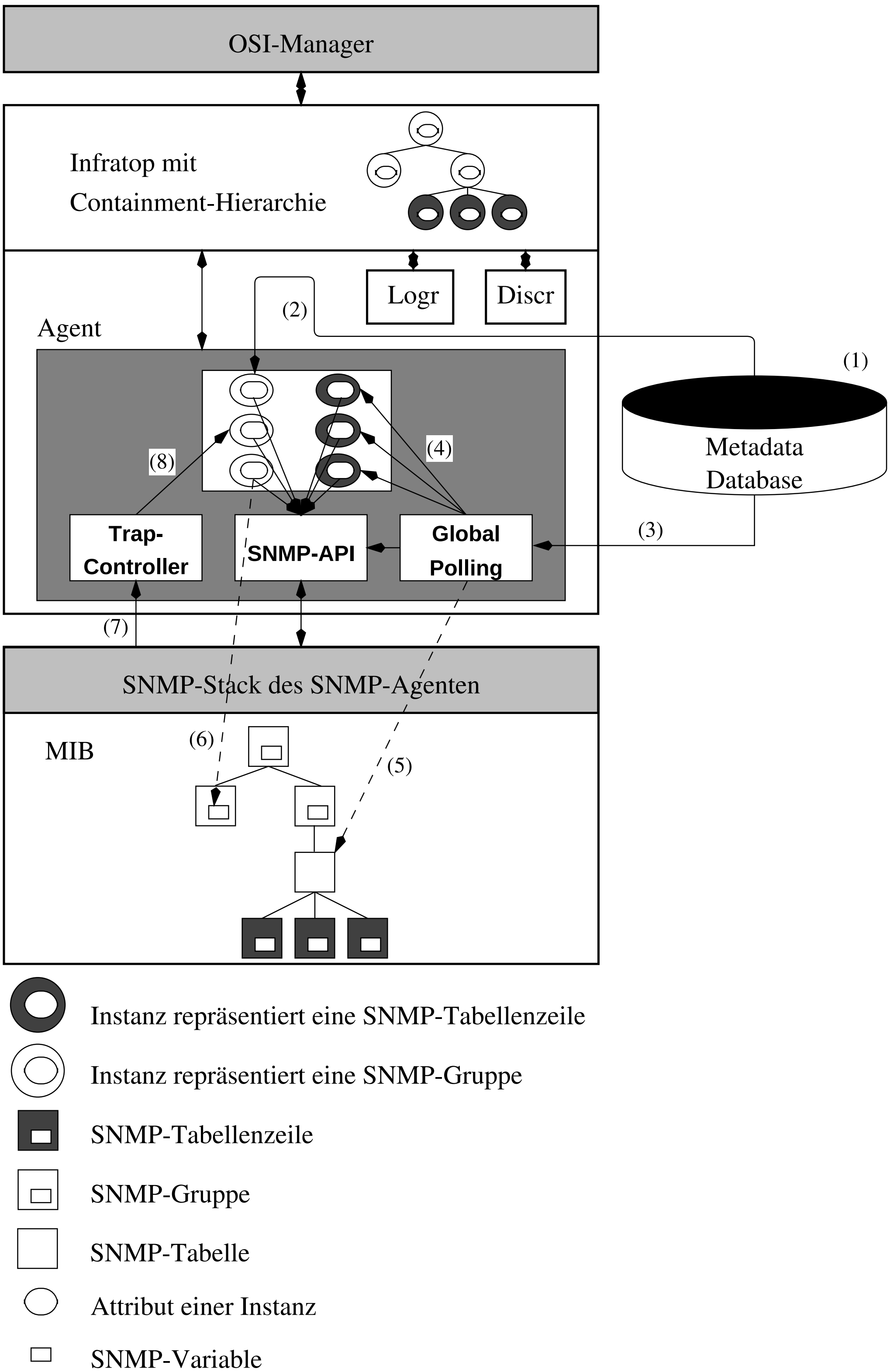


Abbildung 5.7: Die vollständige Architektur des CMIP/SNMP Gateways

5.2.5 „Local Objects“ für das Management des Gateways

Für das Management des Gateways selbst steht die Klasse „cmipSnmpProxy“[1] zur Verfügung. Beim Starten des Gateways soll eine Instanz dieser Klasse automatisch kreiert werden. Das Kreieren einer Instanz dieser Klasse soll dabei das weitere Kreieren einer Instanz der schon oben erwähnten Klasse „snmpComEntity“ bewirken. Damit ist das Management des im Gateway enthaltenen SNMP-Managers ebenfalls möglich. Die Containment-Hierarchie besteht nach dem Start des Gateways aus folgenden Instanzen, siehe Abbildung 5.8.

Nun können, explizit durch einen OSI-Manager oder durch einen automatischen

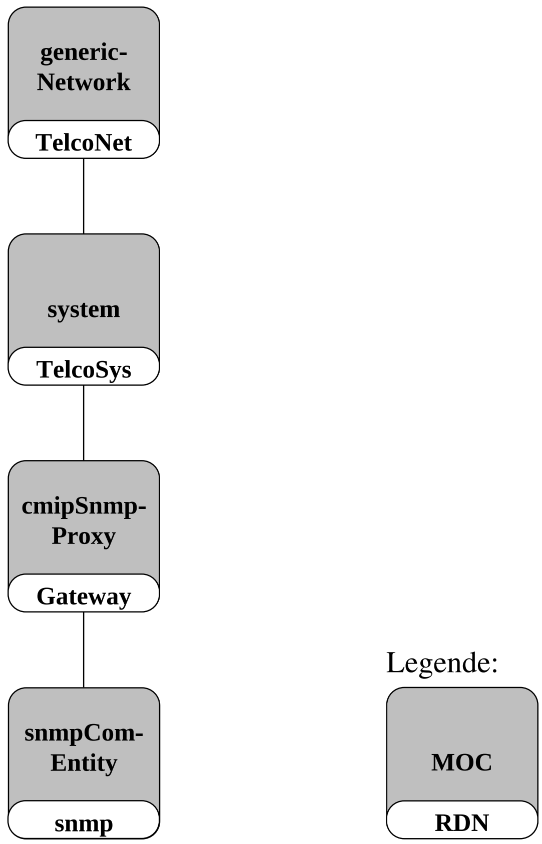


Abbildung 5.8: Die Containment-Hierarchie nach dem Start des Gateways

Explorationsprozeß, die Instanzen, die für das Management von SNMP-Agenten notwendig sind, kreiert werden. Dazu muß für jeden zu managenden SNMP-Agent eine Instanz der Klasse „cmipSnmpProxyAgent“ kreiert werden. Diese Klasse enthält Attribute, die den SNMP-Agenten charakterisieren:

[1]Die GDMO-Templates der OSI-Klassen („local Objects“) „cmipSnmpProxy“, „cmipSnmpProxyAgent“ und „remoteSystem“ wurden mit teilweise kleineren Modifikationen aus dem Internet Draft [IIMCPROXY] übernommen.

- IP-Adresse
- Community-Strings
- die von diesem Agenten unterstützten Internet-MIBs

Das Kreieren einer Instanz dieser Klasse soll automatisch das Kreieren einer Instanz der Klasse „remoteSystem“ bewirkten. Sie dient als *Root-Objekt* für die Containment-Hierarchie der jeweiligen SNMP-Agenten. So werden anschließend einerseits unter dieser Instanz alle Instanzen von Managementobjekten kreiert, die die SNMP-Gruppen aus der MIB dieses SNMP-Agenten repräsentieren, siehe 5.2.1. Andererseits sollen alle SNMP-Tabellen in diesem SNMP-Agenten an der *Global Polling*-Komponente angemeldet werden. Die gesamte Containment-Hierarchie sieht zum Beispiel nach dem Kreieren einer Instanz der Klasse „cmipSnmpProxyAgent“, die den SNMP-Agenten auf dem Rechner „ibmhegering1“ repräsentiert, folgendermaßen aus (wobei dieser Agent die *LRZ Systemagenten-MIB* unterstützt, siehe Abbildung 5.9): Diese Abbildung zeigt deutlich, daß die Containment-Hierarchie der Instanzen, die die MIB des SNMP-Agenten repräsentieren („remote Objects“), mit der Struktur dieser MIB in dem Internet-Registierungsbaums übereinstimmt (vergleiche Abbildung 2.12).
Das Beispiel zeigt aber auch die vorhandene Redundanz: Ein SNMP-Agent wird sowohl von einer Instanz der Klasse „remoteSystem“ als auch von einer Instanz der Klasse „cmipsnmpProxyAgent“ repräsentiert. Dies ist durchaus so gewollt, da bei einer großen Anzahl an zu verwaltenden SNMP-Agenten die Menge der Instanzen unübersichtlich wird. So kann, indem nur der Teilbaum mit der Wurzel „cmipsnmpProxy“ betrachtet wird, sofort festgestellt werden, welche und wieviele SNMP-Agenten gerade vom Gateway überwacht werden. Die Instanzen der Klasse „remoteSystem“ dienen dabei dazu, einen Zugang zu den MIBs der SNMP-Agenten herzustellen. Wie schon erwähnt, stellen diese Instanzen damit Root-Objekte für die Containment-Hierarchien der Instanzen der jeweiligen SNMP-Agenten („remote Objects“) dar.
Weiterhin ist es sinnvoll, die Community-Strings eines SNMP-Agenten nur an einer zentralen Stelle zu verwalten, um auf Änderungen flexibel reagieren zu können. So werden diese, wie schon oben erwähnt, als Attribut in der Instanz der Klasse „cmipsnmpProxyAgent“ gehalten. Daraus resultiert, daß jedes „remote Object“ zuerst den benötigten Community-String aus dieser Instanz lesen muß, bevor sie SNMP-PDUs zu dem jeweiligen SNMP-Agenten auslösen können.

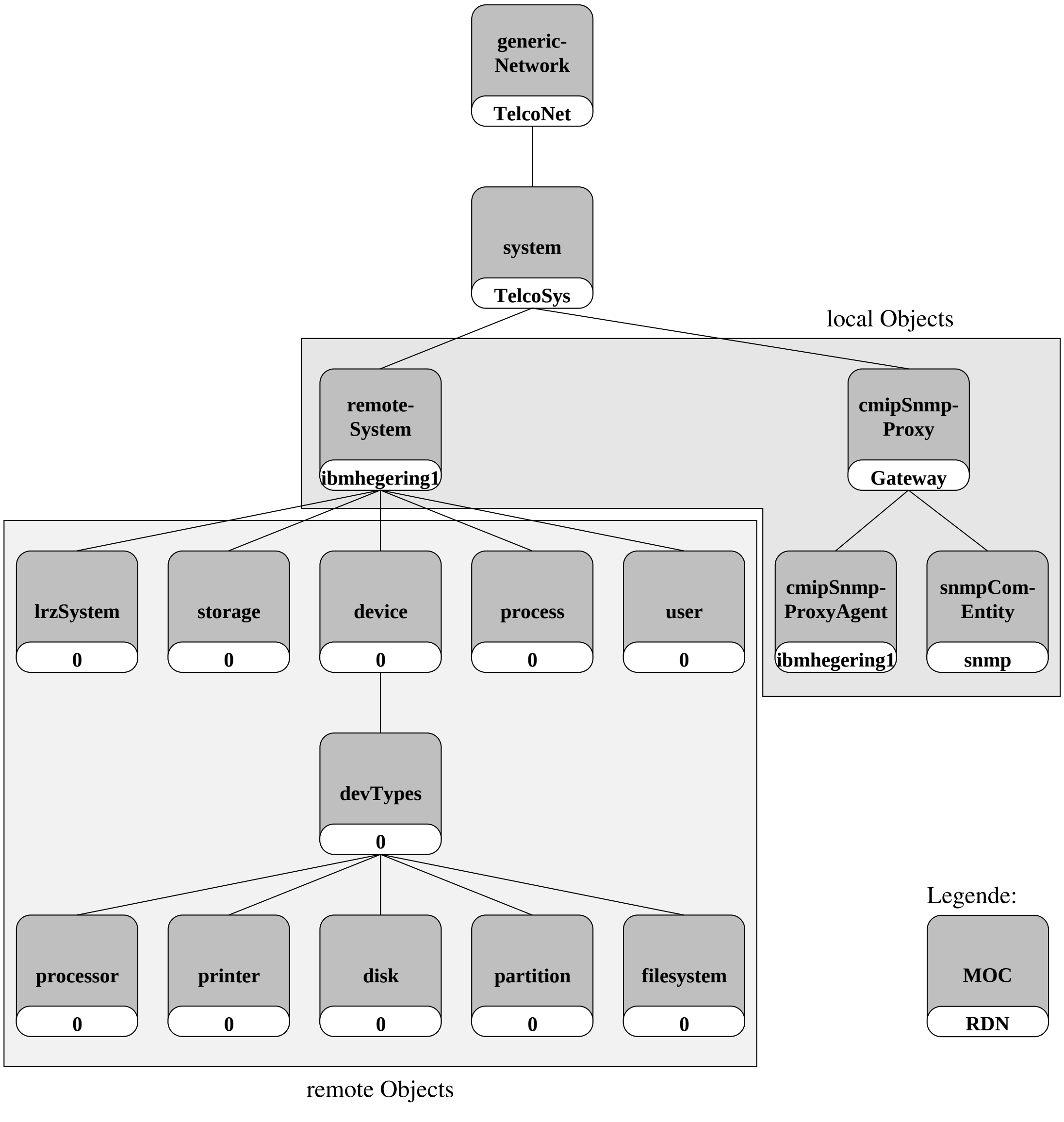


Abbildung 5.9: Die Containment-Hierarchie des Gateways mit einem SNMP-Agenten auf dem Host „ibmhegering1“

5.2.6 Einschränkung von Inkonsistenzen

Durch die Integration des stateful Ansatzes in das Gateway können in der Gateway-MIB Inkonsistenzen auftreten. Eine Einschränkung dieser Inkonsistenzen wird eventuell dadurch erreicht, daß für jede SNMP-Tabelle eine Aktion definiert wird, welche explizit die Aktualisierung dieser Tabelle in der Gateway-MIB veranlaßt.
Die *Global Polling*-Komponente repliziert die angemeldeten SNMP-Tabellen nach einer festgelegten Strategie. Falls ein OSI-Manager nicht sicher feststellen kann, ob die Gateway-MIB, oder Teile davon, konsistent mit den MIBs in den SNMP-Agenten ist, kann er mit Hilfe dieser Aktionen explizit eine Replikation auslösen und damit einen konsistenten Zustand herstellen.
Diese Aktion wird in eine CMIS M-ACTION eingebettet und bestimmten Klassen zugeordnet: Es wird für jede OSI-Klasse, die eine SNMP-Gruppe repräsentiert und aufgrund der NAME BINDING-Hierarchie eine Klasse enthält, die eine SNMP-Tabellenzeile darstellt, die Aktion „aktualisiereTabelle“definiert.
Diese Aktion kann nun direkt für jede SNMP-Tabelle einzeln aufgerufen werden. Mit Hilfe der CMIS-Mächtigkeit ist es aber auch möglich, die Aktion vom Root-Objekt der Containment-Hierarchie des Gateways und dem Scope „Whole Subtree“ auszulösen. Damit werden alle im Gateway verwaltenten SNMP-Tabellen repliziert und somit ein globaler konsistenter Zustand der Gateway-MIB zu den MIBs in den SNMP-Agenten geschaffen.
Diese Aktion widerspricht der am Anfang in Kapitel 4 geforderten Transparenz. Denn sobald ein OSI-Manager Kenntnis von dieser Aktion hat, unterscheidet er zwischen den Managementobjekten, die SNMP-Ressourcen repräsentieren und den anderen Managementobjekten.

5.3 Zusammenfassung und Diskussion

Dieses Kapitel stellt eine Architektur eines CMIP/SNMP Gateways vor, die einem OSI-Manager eine OSI-Sichtweise auf SNMP-Ressourcen zur Verfügung stellt. Ein reines stateless Gateway kann aufgrund der Einschränkungen, die sich beim Einsatz der *IBM TMN WorkBench for AIX* ergeben, nicht realisiert werden: Die Komponenten der WorkBench fordern, daß die gesamten zu überwachenden und steuernden Ressourcen komplett im Agenten durch Instanzen von Managementobjekten und durch deren Attribute repräsentiert werden. Um dies für SNMP-Ressourcen anwenden zu können, müssen Informationen aus der MIB des SNMP-Agenten (der Index der Tabellenzeilen aller SNMP-Tabellen) im Gateway zwischengespeichert werden (als RDN der Instanzen).
Da ein Übergang zu einem reinen stateful Gateway einer idealen Lösung, ebenfalls wie ein reiner stateless Ansatz, nicht nahe kommt, werden in dieser Architektur sowohl stateless als auch stateful Prinzipien integriert: All jene SNMP-Ressour-

cen, welche aufgrund der Einschränkungen der *IBM TMN WorkBench for AIX* nicht direkt auf SNMP-PDUs abgebildet werden können (dem entsprechen SNMP-Gruppen und -Tabellenzeilen), weil sie durch Instanzen im Gateway repräsentiert werden müssen und somit die Containment-Hierarchie aufbauen, sollen nach dem stateful Ansatz implementiert werden. Alle anderen SNMP-Ressourcen (dem entsprechen alle SNMP-Variablen, sie werden durch Attribute in den OSI-Klassen dargestellt) sollen nach dem stateless Ansatz realisiert werden, das heißt, der Wert eines OSI-Attributs soll direkt aus dem SNMP-Agenten gelesen werden. Damit entspricht diese Architektur nicht dem in 4.5 vorgestellten idealisierten Gateway, kommt aber in einigen Ansätzen diesem recht nahe: So werden SNMP-Tabellenzeilen aufgrund ihres zeitlichen Verhaltens und damit aufgrund ihrer Art von Managementinformation in der Gateway-MIB repliziert. Ebenfalls wird der Teil der Containment-Hierarchie, der den Registrierungsbaum der SNMP-Gruppen im Gateway repräsentiert, nur einmal aufgebaut und entspricht damit dem Gesichtspunkt, statische Managementinformation nur einmal oder nur selten zu replizieren.
Der Attributzugriff widerspricht aber den Vorstellungen einer idealen Lösung: Dieser wird nach einem reinen stateless Ansatz behandelt, das heißt, alle Attributzugriffe werden direkt auf SNMP-PDUs abgebildet.

Die aus dieser Architektur resultierende Performance läßt sich folgendermaßen abschätzen:
Für die Auflösung von Scopes werden keine expliziten Zugriffe auf die SNMP-Agenten benötigt, da die Containment-Hierarchie im Gateway verwaltet und von der *Global Polling*-Komponente konsistent mit der realen SNMP-Ressource gehalten wird. In dieser Beziehung verhält sich die Architektur wie ein reines stateful Gateway. Damit ergibt sich auch der Performance-Vorteil (siehe auch 4.6) gegenüber den stateless Gateway, welches ja alle Informationen zur Scopeauflösung direkt aus den SNMP-Agenten beschaffen muß. Bei der Auflösung von Filtern verhält sich die Architektur dagegen wie ein stateless Gateway. Alle Werte der Attribute, die im Filter spezifiziert werden, müssen für alle im Scope selektierten Instanzen explizit aus den SNMP-Agenten gelesen werden, bevor sie zur Auflösung des Filters herangezogen werden können. Damit treten für diese Architektur für die Filterauflösung auch die Performance-Nachteile auf, wie sie bei einem stateless Gateway typisch sind. Ebenso verhält es sich mit Attributzugriffen, da diese auch nach dem stateless Ansatz erfolgen. Schließlich darf nicht vergessen werden, daß durch die Replikation der SNMP-Tabellenzeilen in der Gateway-MIB durch die *Global Polling*-Komponente eventuell Inkonsistenzen (siehe 4.4.2) oder auch hoher Ressourcenverbrauch auftreten können: So wird zur Konsistenzsicherung einer SNMP-Tabelle in der Gateway-MIB, also dem Erkennen von neuen bzw. veralteten Tabellenzeilen, immer die gesamte SNMP-Tabelle benötigt. Diese muß jedesmal nach dem Ablauf des Polling-Intervalls von der *Global Polling*-Komponente angefordert werden. Dies kann bei großen Tabellen zu einer Vielzahl an

SNMP-PDUs führen und damit zu langen Wartezeiten, in denen das Gateway nicht für andere Aufgaben zur Verfügung steht.
Eine denkbare Weiterentwicklung dieser Architektur ist der Übergang zu einem reinen stateful Gateway. Dazu müßte noch der Attributzugriff dahingehend verändert werden, daß sich die Attribute ebenfalls, wie SNMP-Tabellenzeilen, an der *Global Polling*-Komponente anmelden und von dieser durch den Einsatz einer bestimmten Strategie repliziert werden. Zukünftige Performance-Messungen müssen zeigen, ob dieser Schritt sinnvoll ist, da damit auch der Nachteil des stateful Ansatzes noch schwerwiegender auftritt, daß die hundertprozentige Konsistenzsicherung der SNMP-Agenten in der lokalen Gateway-MIB nicht erreicht werden kann (siehe 4.4.2).
Im nächsten Kapitel soll die Implementierung dieser CMIP/SNMP Gatewayarchitektur beschrieben werden.

Kapitel 6

Die Implementierung des CMIP/SNMP Gateways

Nachdem im vorherigen Kapitel die Architektur des CMIP/SNMP Gateways erarbeitet wurde, soll diese nun implementiert werden. Die Implementierung erfolgt in C++, aus dem einfachen Grund, da eine Agentenentwicklung mit der *IBM TMN WorkBench for AIX* auf dieser Programmiersprache basiert. Es bietet sich für dieses Kapitel folgende Gliederung an, entsprechend gegenläufig zu den Abschnitten in Kapitel 5.2 (wobei die Implementierung der „local Objects" nicht behandelt wird):
So soll zuerst die SNMP-Manager-Funktionalität in das Gateway integriert werden. Anschließend wird die Realisierung der *Global Polling*-Komponente beschrieben. Darauf aufbauend werden die C++-Klassen, welche den OSI-Klassen und Attributen in den GDMO-Templates zugrundeliegen, implementiert.
Die Bedründung für diese Gliederung liegt darin, daß die SNMP-Manager-Funktionalität eine Voraussetzung für die Implementierung sowohl der *Global Polling*-Komponente als auch der gerade erwähnten C++-Klassen darstellt und somit zu Beginn vorgestellt wird. Ebenso beruhen die Realisierungen dieser C++-Klassen auf einer Implementierung der *Global Polling*-Komponente; somit erfolgt eine Beschreibung der Polling-Funktionalität an zweiter Stelle.
Da eine CMIS M-ACTION stets einer OSI-Klasse zugeordnet werden muß, wird die Implementierungsbeschreibung der Aktion „aktualisiereTabelle" in Abschnitt 6.3 eingefügt.
Zuletzt wird ein Werkzeug vorgestellt, daß orthogonal zu der bisherigen Implementierung steht und die Aufgabe erfüllt, den C++-Code für die OSI-Klassen und Attribute aus Abschnitt 6.3 automatisch zu generieren.

6.1 Der SNMP-Manager

In Abschnitt 5.2.3 wurde bereits erwähnt, daß sich der SNMP-Manager des Gateways aus den Komponenten *SNMP-API* und *SNMP-TrapController* zusammensetzt.

6.1.1 Die *SNMP-API*-Komponente

Die Komponente *SNMP-API* hat die Aufgabe, die Kommunikation des Gateways mit den SNMP-Agenten bereitzustellen. Dazu enthält das *SNMP-API* eine Instanz der C++-Klasse `SNMP`. Diese Klasse kapselt die verwendete SNMP-Programmierschnittstelle und stellt für die Kommunikation aller anderen Komponenten des Gateways mit der *SNMP-API* eine Menge an Methodenaufrufen zur Verfügung. Dabei wäre folgende Struktur wünschenswert:

```
class SNMP
{
        public:
                SNMP();
                ~SNMP();
                get(VarBind vBind,IpAdr ipAdr,Community comStr);
                set(VarBind vBind,IpAdr ipAdr,Community comStr);
                specialWalk(Oid oid,IpAdr ipAdr,ListeA indizes);
};
```

Die C++-Klasse `SNMP` ist für die Überwachung und Steuerung einer „SNMP-Session“ verantwortlich. Damit ist in diesem Zusammenhang folgendes gemeint[1]:

1. „Variable Bindings“ in PDUs einbinden und auspacken.

2. PDUs abschicken und empfangen.

3. Timeouts, Paketverluste, Paketduplikate usw. automatisch bearbeiten.

Wobei der Konstruktor eine „SNMP-Session“ aufbaut, der Destruktor diese abbaut. Die Methode `get` dient für das Lesen von SNMP-Variablen. Dazu wird in Parameter `vBind` eine Datenstruktur („Variable Binding“) übergeben, die sowohl die OIDs der gewünschten SNMP-Variablen als auch das (oder die) Ergebnis(se) der Anfrage enthält. Mit `ipAdr` wird der angesprochene SNMP-Agent identifiziert und mit `comStr` wird der benötigte Community-String übermittelt. Entsprechend werden mit der Methode `set` ein oder mehrere SNMP-Variablen auf neue Werte gesetzt. Die Methode `specialWalk` wurde ausschließlich für die *Global Polling-*

[1] nach [Mell96]

Komponente eingeführt (siehe 6.2). Ihre Aufgabe ist es, die Indizes aller Tabellenzeilen in der mit `oid` festgelegeten Tabelle in der Datenstruktur `indizes` zurückzugeben. Dies wird benötigt, um feststellen zu können, ob in einer Tabelle nach einer bestimmten Zeit neue bzw. veraltete Tabelleneinträge aufgetreten sind. Diese Methode basiert auf dem Aufrufen mehrerer SNMP-GetNextRequest-PDUs.
Die Methoden und Konzepte dieser Klasse mußten nun auf eine bestehende SNMP-Programmierschnittstelle abgebildet werden. Da sich die Implementierung des Gateways auf die Programmiersprache C++ abstützt, lag es nahe, eine C++-SNMP-Programmierschnittstelle zu verwenden. Trotz intensiver Bemühungen, auch in Zusammenarbeit mit [Ho96], konnte keine gefunden werden (das Ergebnis beschränkte sich lediglich auf die Klassendefinitionen in [Mell96]).
Daher blieb nur die Verwendung einer C-SNMP-Programmierschnittstelle übrig. Die in *IBM NetView for AIX* enthaltene Schnittstelle konnte aufgrund von Headerdatei-Konflikten nicht in die *IBM TMN WorkBench for AIX* integriert werden. So werden in der *IBM NetView for AIX* SNMP-Schnittstelle in einer Headerdatei verschiedene SNMP-Fehlertypen per `#define` definiert. Ebenfalls werden bei der *IBM TMN WorkBench for AIX* in einer Headerdatei die gleichen Bezeichnungen für OSI-Fehlertypen per `#define` festgelegt.
So basiert schließlich die Komponente *SNMP-API* auf der SNMP-Programmierschnittstelle „ucd-snmp“ in der Version 3.0.1, die an der *University of California at Davis* von Wes Hardaker entwickelt wurde. Sie stellt eine C-Bibliothek mit sämtlichen Funtkionen für die Kommunikation eines SNMP-Managers mit einem SNMP-Agenten zur Verfügung. Durch die Besonderheit dieser Schnittstelle, daß keine globale „SNMP-Session“ existiert, sondern für jeden zu überwachenden SNMP-Agenten die oben erwähnten Verwaltungsaufgaben einzeln ausgeführt werden müssen, macht eine Änderung der wünschenswerten Konzepte in der C++-Klasse `SNMP` notwendig. In der *ucd*-API wird die oben definierte „SNMP-Session“ folgendermaßen eingeschränkt: Beim Aufbau einer „ucd-Session“ muß eine Internet-Adresse und ein Community-String angegeben werden. Eine „ucd-Session“ ist damit nicht unabhängig von den zu überwachenden SNMP-Agenten; es muß somit für jeden Agenten eine eigene „ucd-Session“ aufgebaut und verwaltet werden. Dazu wird die Klasse `SNMP` um eine interne Liste (`hostList`) erweitert, die jeder Internet-Adresse eine „ucd-Session“ zuordnet. Bevor das Gateway mit einem SNMP-Agenten kommunizieren kann, muß es diesen mit dessen Internet-Adresse und dessen Community-String an der *SNMP-API* anmelden[2]. Dadurch wird eine „ucd-Session“ zu diesem SNMP-Agenten aufgebaut. Danach kann das Gateway beliebig über die Methoden `get`, `set` und `specialWalk` mit dem SNMP-Agenten kommunizieren. Die *SNMP-API* kann anhand der in den Methodenaufrufen übergebenen Internet-Adresse diesen Request einer bestehenden „ucd-Session“ zuordnen und damit die notwendigen SNMP-PDUs auslösen (siehe Ab-

[2]Dies kann im Zuge des Kreierens einer Instanz der Klasse „cmipsnmpProxyAgent“, welche diesen SNMP-Agent im Gateway repräsentiert, erfolgen.

bildung 6.1). Das Verhalten der Methoden soll sich natürlich nicht verändern:

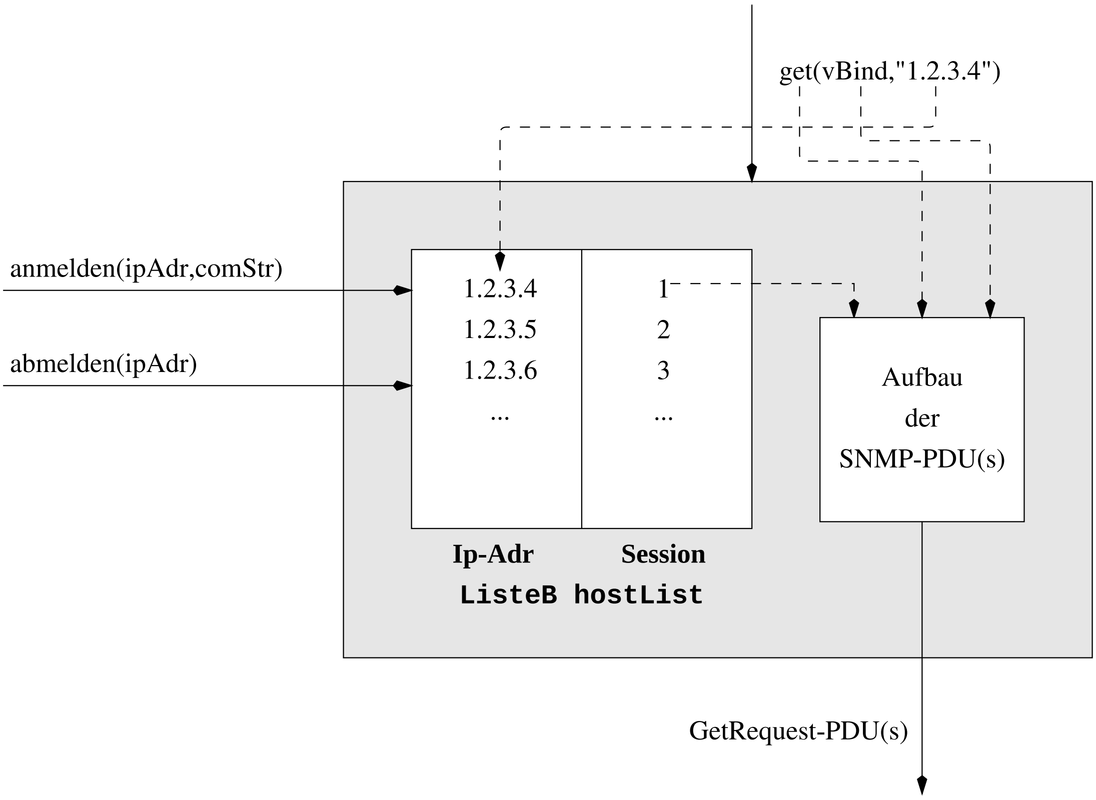


Abbildung 6.1: Die Funktionsweise der *SNMP-API*-Komponente

```
class SNMP
{
        private:
                ListeB hostList;
        public:
                SNMP();
                ~SNMP();
                anmelden(IpAdr ipAdr,Community comStr);
                abmelden(IpAdr ipAdr);
                get(VarBind vBind,IpAdr ipAdr);
                set(VarBind vBind,IpAdr ipAdr);
                specialWalk(Oid oid,IpAdr ipAdr,ListeA indizes);
};
```

Falls sich der Community-String während einer „ucd-Session“ verändert, muß die bestehende Session abgebaut und eine neue, mit dem neuen Community-String, aufgebaut werden.

6.1.2 Der SNMP-TrapController

Die Aufgabe des *TrapController*[3] läßt sich in zwei Kategorien einteilen (siehe 5.2.4):

1. Das Empfangen einer SNMP-Trap-PDU am Port 162.

2. Die Abbildung des empfangenen Traps auf eine OSI-Notifikation und deren Zuordnung zu einer OSI-Instanz.

Der *TrapController* besteht aus einer Instanz der C++-Klasse `SNMPTrap` und ist abgeleitet von der C++-Klasse `ResourceAccess`. Die letztere stellt eine Implementierung der in 3.2.1 vorgestellten *Resource Access*-Komponente dar und wird vom *Managed Object/Agent Composer* bereitgestellt. Damit steht nach dem TMN-Referenzmodell ein *m-Interface* zur Verfügung, also die Möglichkeit, SNMP-Traps zu empfangen und zu bearbeiten (siehe 5.2.4). Dieses *m-Interface* wird folgendermaßen realisiert:
Im *Agent*-Prozeß des Gateways, genauer in der *CoreAgent*-Komponente, wird eine zentrale Endlosschleife (Ereignisauswahlschleife) implementiert. Alle Requests des *Infratops* werden dort an speziellen Sockets empfangen und deren Bearbeitung ausgelöst. Die *CoreAgent*-Komponente ermöglicht es einem Entwickler, einen eigenen Socket mit einer Instanz einer von `ResourceAccess` abgeleiteten C++-Klasse in dieser Ereignisauswahlschleife einzubinden. Sobald eine Nachricht an diesem Socket anliegt, wird eine spezielle Methode (`doRead`) der `ResourceAccess`-Klasse aufgerufen. Diese Methode ist als rein virtuell

```
virtual doRead(...)=0;
```

definiert, das heißt, sie muß von einer abgeleiteten Klasse explizit neu realisiert werden. Die Aufgabe der C++-Klasse `SNMPTrap` ist es nun, einen UDP-Socket an den Port 162 zu binden. Anschließend wird dieser Socket und eine Instanz dieser Klasse selbst an der zentralen Ereignisauswahlschleife angemeldet. Es wird weiterhin in dieser Klasse die als virtuell definierte, geerbte Methode `doRead` implementiert. Diese wird von der zentralen Ereignisauswahlschleife aufgerufen, sobald ein Trap am Port 162 angekommen ist. In dieser Methode erfolgt die Abbildung des empfangenen Traps auf die OSI-Notifikation „internetAlarm“ aus [IIMCIMIBTR] nach den Regeln aus [IIMCPROXY]. Zuletzt muß diese Notifikation noch einer OSI-Instanz zugeordnet werden. Dies erfolgt nach folgendem Algorithmus:

1. Falls nicht festgestellt werden kann, von welchem SNMP-Agenten der Trap stammt, bzw. dieser Agent nicht vom Gateway überwacht wird, soll die Notifikation von der Instanz der Klasse „cmipsnmpProxy“weitergeleitet werden.

[3] Diese Komponente wurde noch nicht vollständig implementiert.

2. Falls der SNMP-Agent vom Gateway überwacht wird, dieser die MIB-2 aber nicht unterstützt, soll die Instanz der Klasse „cmipsnmpProxyAgent“, die diesen Agent repräsentiert, die Bearbeitung der Notifikation auslösen.

3. Im letzten Fall (das heißt, falls der SNMP-Agent vom Gateway überwacht wird und die MIB-2 unterstützt), soll die Instanz der Klasse „internetSystem“, die die MIB-2-Gruppe „system“ dieses SNMP-Agenten repräsentiert, die Weiterleitung der Notifikation übernehmen.

Die Weiterleitung der Notifikation zum Event Handler und damit zu den EFDs erfolgt mit Hilfe einer Methode, die standardmäßig vom *Managed Object/Agent Composer* in jede Instanz integriert ist.

6.2 Die *Global Polling* -Komponente

Die Aufgaben der *Global Polling*-Komponente wurden in den Abschnitten 5.2.1 und 5.2.2 motiviert und diskutiert. In diesem Abschnitt werden eine Reihe von C++-Klassen vorgestellt, um die Funktionalität dieser Komponente zu realisieren.
Die Reihenfolge der Vorstellung erfolgt nach den Vererbungshierarchien und Enthaltenseinsbeziehungen, die diesen Klassen zugrunde liegen. Es wird somit zu Beginn eine Basisklasse definiert, um anschließend in aufeinander aufbauenden Schritten die vollständige Strukturierung und Funktionsweise der *Global Polling*-Komponente zu erhalten:

6.2.1 Die C++-Klasse `MOIofSNMP_Table`

Diese Basisklasse soll Methoden bereitstellen, um für jede Tabellenzeile in einer SNMP-Tabelle OSI-Instanzen im Gateway kreieren bzw. um für veraltete Tabellenzeilen die entsprechenden OSI-Instanzen löschen zu können.

```
class MOIofSNMP_Table
{
        private:
                stringList *oldMOI, *newMOI;
                char oid[512];
                char ipAddress[512];
                int createMOIs;
                int deleteMOIs;
                char NamingAttrOID[512];
                char ClassOID[512];
                char ParentInstName[512];
```

```
        public:
                MOIofSNMP_Table(char *newOid,char *newIpAddress,
                        char *newNamingAttrOID,char *newClassOID,
                        char *newParentInstName);
                ~MOIofSNMP_Table();
                int updateMOIs(int &changes);
                char *getSNMP_OID() { return(oid); }
};
```

Dazu wird dem Konstruktor dieser Klasse

1. die Internet-Adresse des SNMP-Agenten,

2. die OID der SNMP-Tabelle,

3. die OID der OSI-Klasse, die im Gateway eine Tabellenzeile repräsentieren soll,

4. die OID des Naming Attributs dieser OSI-Klasse und schließlich

5. der DN (Distinguished Name) der Vaterinstanz, die der Instanz der OSI-Klasse, die die Tabellenzeile repräsentieren soll, in der Containment-Hierarchie übergeordnet ist

übergeben. Eine Instanz dieser C++-Klasse verwaltet weiterhin zwei Listen:

1. Die `oldMOI`-Liste enthält alle Indizes der SNMP-Tabellenzeilen, die gerade im Gateway von Instanzen repräsentiert werden.

2. In der `newMOI`-Liste werden alle Indizes der SNMP-Tabellenzeilen gespeichert, die gerade im SNMP-Agenten existieren.

Bei der Initialisierung durch den Konstruktor werden zwei leere Listen erzeugt. Der Aufruf der Methode `updateMOIs` bewirkt folgendes:

1. Es werden von der SNMP-Tabelle mit der OID `oid` in der MIB des SNMP-Agenten auf dem Host mit der Internet-Adresse `ipAddress` alle Indizes der gerade vorhandenen Tabellenzeilen in dieser Tabelle in der Liste `newMOI` abgespeichert. Dies wird mit der Methode `specialWalk` der *SNMP-API* (siehe 6.1.1) erreicht.

2. Anschließend werden die beiden Listen `newMOI` und `oldMOI` miteinander verglichen:

   (a) Falls ein Index in `newMOI`, aber nicht in `oldMOI` existiert, muß eine OSI-Instanz für diese Tabellenzeile im Gateway kreiert werden: Die

OSI-Klasse der neu zu kreierenden Instanz wird über `classOID` identifiziert. Der DN der neuen Instanz setzt sich aus dem `ParentInstName` und dem RDN dieser Instanz zusammen. Der RDN ergibt sich schließlich aus der OID des Naming Attributs und dem Index der SNMP-Tabellenzeile.

(b) Falls ein Index in der `oldMOI`, aber nicht in der `newMOI` existiert, muß die entsprechende Instanz im Gateway gelöscht werden. Die Identifizierung der zu löschenden Instanz erfolgt durch die `classOID` (damit wird die OSI-Klasse der Instanz bezeichnet) und dem DN (dieser wird aus dem `ParentInstName`, der Naming Attribut OID und dem Index berechnet, siehe Punkt (a)).

(c) Falls ein Index sowohl in `oldMOI` als auch in `newMOI` existiert, wird keine Bearbeitung dieser Tabellenzeile notwendig.

Somit kann durch regelmäßigen Aufruf der Methode `updateMOIs` die SNMP-Tabelle im Gateway repliziert werden, das heißt, jede Tabellenzeile einer SNMP-Tabelle wird von einer OSI-Instanz repräsentiert. Eventuell notwendiges Kreieren bzw. Löschen von OSI-Instanzen wird von dieser C++-Klasse automatisch ausgeführt.

6.2.2 Die C++-Klasse `PollingIntervall`

In dieser Klasse wird die Grundlage für die Implementierung der Strategie aus 4.4.2 bzw. 5.2.2 gelegt.

```
class PollingIntervall
{
        private:
                unsigned int pollingIntervall;
                int pollingClass;
        public:
                PollingIntervall(int newClass=2);
                ~PollingIntervall();
                int pollingClassUp(int step);
                int pollingClassDown(int step);
                int getPollingClass();
                int getPollingIntervall();
                void print();
};
```

Es wird ein 32 Bit-Integer (`pollingIntervall`) definiert, wobei jedes einzelne Bit eine Intervallgröße und damit eine Polling-Klasse festlegt. Es existieren Methoden (`pollingClassUp`, `pollingClassDown`), um von einer Polling-Klasse in eine beliebig andere zu wechseln. Mit `getPollingIntervall` wird die aktuelle Intervallgröße zurückgegeben, die dabei gültige Einheit (1,2,4,... Sekunden) wird nicht hier festgelegt.

6.2.3 Die C++-Klasse `dynMOICreation_ofSNMP_Table`

Diese Klasse verbindet die beiden oben vorgestellten. So ist sie abgeleitet von der Klasse `MOIofSNMP_Table` und enthält eine Instanz der Klasse `PollingIntervall`.

```
class dynMOICreation_ofSNMP_Table : protected  MOIofSNMP_Table
{
        private:
                PollingIntervall pollInt;
                int actualTime;
                int changes;
                int numberMOIs;
        public:
                dynMOICreation_ofSNMP_Table(char *newIpAddress,
                        char *newOid, char *newNamingAttrOID=NULL,
                        char *newClassOID=NULL,
                        char *newParentInstName=NULL) :
                MOIofSNMP_Table(newOid,newIpAddress,
                        newNamingAttrOID,newClassOID,
                        newParentInstName)
                {
                        actualTime=0;
                        numberMOIs=0;
                }
                ~dynMOICreation_ofSNMP_Table() { }
                int TestToCreateMOI();
                int CreateMOI();
                int getPollingIntervall();
                int getPollingClass();
};
```

Die Aufgabe einer Instanz dieser Klasse läßt sich in zwei Kategorien einteilen, entsprechend existieren dafür die beiden Methoden `TestToCreateMOI` und `CreateMOI`:
Die Methode `TestToCreateMOI` bildet zusammen mit einer Instanz der Klasse `PollingIntervall` (`pollInt`) die Realisierung der Polling-Strategie. Eine Voraussetzung dafür ist, daß diese Methode regelmäßig aufgerufen wird, wobei dieser konstante Zeitabstand zwischen dem Pollen die bei der Klasse `PollingIntervall` erwähnte Einheit darstellt. Zum Beispiel kann der Methodenaufruf jede Sekunde, alle zwei Sekunden, usw. erfolgen. Jeder Methodenaufruf wird in der Varia-

ble actualTime mitgezählt und stellt somit einen Zeitzähler dar (siehe Codeausschnitt unten). Erreicht dieser Zeitzähler die gerade aktuelle Intervallgröße, muß eine Replikation der SNMP-Tabelle, die dieser Klasse zugrundeliegt (da sie ja von MOIofSNMP_Table abgeleitet ist), erfolgen. Die Replikation wird mit der geerbten Methode updateMOIs ausgelöst. Anschließend muß festgestellt werden, ob OSI-Instanzen kreiert bzw. gelöscht worden sind, um entsprechend das Pollingintervall zu vergrößern oder zu verkleinern. Zuletzt wird der Zeitzähler actualTime auf Null gesetzt, damit die Zeit bis zur nächsten Replikation der SNMP-Tabelle im Gateway nach der aktuellen Intervallgröße neu berechnet werden kann.

```
int dynMOICreation_ofSNMP_Table :: TestToCreateMOI()
{
        actualTime++; // Zeitzaehler erhoehen = Anzahl der
        // Aufrufe der Methode zaehlen
        if (actualTime==pollInt.getPollingIntervall())
                // Pollingintervall abgelaufen ?
        {
                // Falls ja, rufe Methode zur Replikation auf
                numberMOIs = updateMOIs(changes);
                if (changes)
                        // Falls Klassen kreiert bzw. geloescht
                        // wurden, erniedrige die Intervallgroesse
                        pollInt.pollingClassDown(1);
                else
                        // sonst erhoehe die Intervallgroesse
                        pollInt.pollingClassUp(1);
                // setze Zeitzaehler auf 0
                actualTime = 0;
        }
        return(numberMOIs);
}
```

Die Methode CreateMOI steht unabhängig zu der gerade implementierten Polling-Strategie: Sie ruft ohne Beachtung des Zeitzählers direkt die geerbte Methode updateMOIs auf. Dies wird zur Realisierung der Aktion „aktualisiereTabelle“ benötigt. Dort soll ja die explizite, sofortige Replikation der SNMP-Tabelle im Gateway erfolgen.

6.2.4 Zusammenfassung der bisherigen Funktionalität

Durch die Instanziierung der Klasse dynMOICreation_ofSNMP_Table werden die Vorausetzungen geschaffen, eine SNMP-Tabelle einer MIB eines SNMP-Agenten in der Gateway-MIB zu replizieren. Dazu muß die Methode TestToCreateMOI in konstanten Zeitabständen (zum Beispiel eine Sekunde) aufgerufen werden. Anhand der implementierten Polling-Strategie wird überprüft, ob der Zeitpunkt für eine Replikation erreicht ist oder nicht (siehe Abbildung 6.2). Falls er erreicht ist,

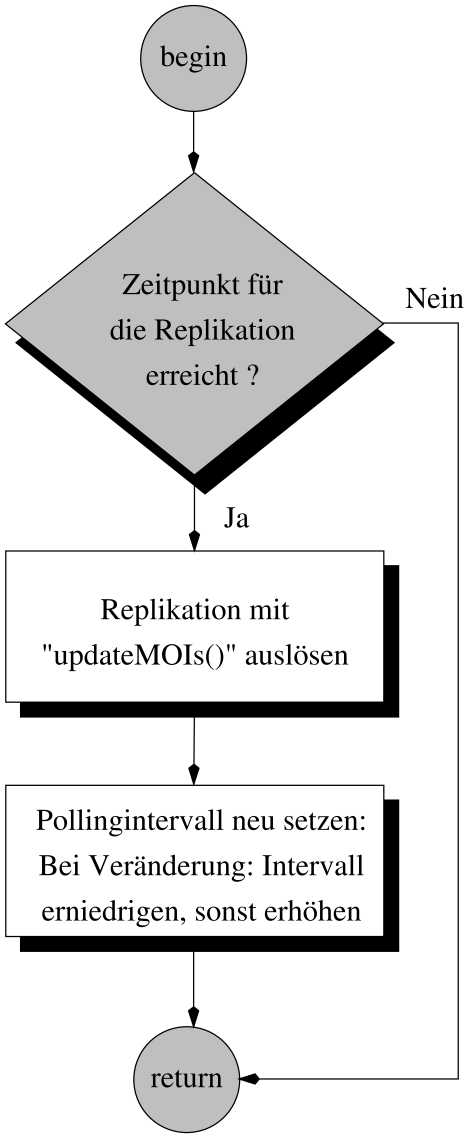


Abbildung 6.2: Ablaufdiagramm zur Replikation einer SNMP-Tabelle

wird die Replikation der SNMP-Tabelle in der Gateway-MIB durch den Aufruf der geerbten Methode `updateMOIs` ausgelöst. Entsprechend der Art der Managementinformation, die die Instanzen/Tabellenzeilen repräsentieren, wird die Polling-Klasse und damit der zeitliche Abstand zwischen zwei Replikationen, dynamisch festgelegt. Die Abbildung 6.3 zeigt die bisherige Vererbungs- und Enthaltenseinshierarchie der C++-Klassen:

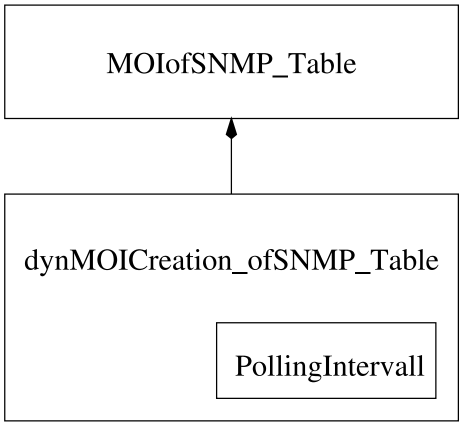


Abbildung 6.3: Die bisherige Vererbungs- und Enthaltenseinshierarchie der C++-Klassen

6.2.5 Die C++-Klasse `iimcPollingTab`

Die bisherige Funktionalität ermöglicht es lediglich, eine SNMP-Tabelle nach einer Polling-Strategie in der Gateway-MIB zu replizieren. Es ist aber notwendig, daß eine SNMP-Tabelle an der *Global Polling*-Komponente angemeldet bzw. abgemeldet werden kann und die Replikation aller angemeldeten Tabellen ausgeführt wird.
Dafür wurde eine einfach verkettete Liste implementiert, wobei jedes Listenelement genau eine SNMP-Tabelle repräsentiert und vom Typ `iimcPollingTab` ist.

```
class iimcPollingTab
{
        private:
                dynMOICreation_ofSNMP_Table *dynMOIPtr;
                char pollingTabName[512];
                iimcPollingTab *next;
        public:
                iimcPollingTab(dynMOICreation_ofSNMP_Table
                        *newDynMOIPtr,char *name);
                ~iimcPollingTab();
                dynMOICreation_ofSNMP_Table *getDynMOIPtr();
                char *getPollingTabName();
                iimcPollingTab *getNext();
                void setNext(iimcPollingTab *isNext);
};
```

Eine Instanz der Klasse `iimcPollingTab` (damit also jedes Listenelement) beinhaltet drei Variablen:

1. Es wird ein Zeiger auf eine Instanz der Klasse `dynMOICreation_ofSNMP_Table` gespeichert. Damit wird eine Verbindung von jedem Listenelement mit der zu repräsentierenden SNMP-Tabelle hergestellt und es kann auch von dem jeweiligen Listenelement aus die Replikation der SNMP-Tabelle nach der gegebenen Polling-Strategie im Gateway angestoßen werden.

2. Über den `pollingTabName` kann das Listenelement eindeutig identifiziert werden (Die Eindeutigkeit wird dadurch erreicht, daß dieser Name unter anderem auch aus der Registrierungs-OID der OSI-Klasse, die eine Tabellenzeile dieser SNMP-Tabelle repräsentieren soll, gebildet wird).

3. Der Pointer vom Typ `iimcPollingTab` zeigt auf das nächste Listenelement in der Liste.

Es existeren weiterhin Methoden, mit denen diese Variablen gelesen bzw. verändert werden können.

6.2.6 Die C++-Klasse `iimcPollingTabList`

Mit dieser Klasse steht schließlich eine einfach verkettete Liste zur Verfügung, in welcher alle in die Gateway-MIB zu replizierenden SNMP-Tabellen verzeichnet werden können. Für das Arbeiten mit der Liste werden im wesentlichen drei Methoden bereitgestellt:

```
class iimcPollingTabList
{
        private:
                iimcPollingTab *first;
                iimcPollingTab *last;
                iimcPollingTab *now;
        public:
                iimcPollingTabList();
                ~iimcPollingTabList();
                void append(dynMOICreation_ofSNMP_Table *dynMOI,
                        char *name);
                iimcPollingTab *get(char *name);
                iimcPollingTab *getFirst();
};
```

1. `append`: Es wird ein neues Listenelement vom Typ `iimcPollingTab` am Ende der Liste angefügt. Dazu wird zum einen ein Zeiger auf eine Instanz der Klasse `dynMOICreation_ofSNMP_Table` (für die Replikation der SNMP-Tabelle) und zum anderen ein Identifikator (ein eindeutiger Name des Listenelements) benötigt.

2. `get`: Diese Methode liefert zu einem übergebenen Namen (Identifikator) einen Zeiger auf das dazugehörige Listenelement vom Typ `iimcPollingTab`. Falls kein Listenelement mit dem übergebenen Namen exisitert, wird der NULL-Zeiger zurückgegeben.

3. `getFirst`: Diese Methode liefert einen Zeiger auf das erste Listenelement oder den NULL-Zeiger, falls die Liste leer ist.

6.2.7 Die C++-Klasse `myGlobalPollingClass`

Eine Instanz dieser Klasse stellt schließlich die *Global Polling*-Komponente dar und integriert alle bisher in diesem Abschnitt vorgestellten C++-Klassen:
Im Hauptprogramm des Gateways, genauer in der *CoreAgent*-Komponente, wird eine Instanz dieser Klasse kreiert. Damit (da die `myGlobalPollingClass` von `iimcPollingTabList` abgeleitet ist) wird eine leere Liste vom Typ `iimcPollingTabList` erzeugt.

```
class myGlobalPollingClass : public iimcPollingTabList {
        public:
                myGlobalPollingClass();
                ~myGlobalPollingClass();
                void doPolling();
};
```

Die in der Gateway-MIB zu replizierenden SNMP-Tabellen können nun mit der geerbten Methode `append` an der *Global Polling*-Komponente registriert werden. Damit steht für jede registrierte (zu replizierende) SNMP-Tabelle ein Listenelement vom Typ `iimcPollingTab` und damit ein Zeiger auf eine Instanz der Klasse `dynMOICreation_ofSNMP_Table` zur Verfügung. Es ist nun die Aufgabe einer Instanz dieser Klasse `myGlobalPollingClass`, für jede angemeldete SNMP-Tabelle regelmäßig die Methode `TestToCreateMOI` der Instanz der Klasse `dynMOICreation_ofSNMP_Table` aufzurufen, um damit die Replikation zu veranlassen.
Im Idealfall sollten die Methodenaufrufe `TestToCreateMOI` für jede SNMP-Tabelle in konstanten Zeitabständen (zum Beispiel eine Sekunde) erfolgen, da damit die Grundeinheit für die Polling-Strategie und damit für die Polling-Intervallgröße festgelegt wird (siehe oben). Da es sich aber bei dem zugrundeliegenden Betriebssystem (IBM AIX 4.2) um kein Echtzeitbetriebssystem handelt, kann dieser Idealfall nicht realisiert werden. Dies stellt aber keine Einschränkung dar, denn in der Praxis ist es nicht notwendig, daß das Polling in konstanten Intervallen ausgeführt wird. Solange das Pollingintervall klein genug ist, um jede Änderung festzustellen, ist es belanglos, ob die Zeiten zwischen dem Pollen in einem gewissen Maß voneinander abweichen.

Auf der Grundlage dieser Überlegungen wurde die Methode `doPolling` der *Global Polling*-Komponente in die zentrale Ereignisauswahlschleife eingebunden. Bei jedem Schleifendurchlauf wird die Methode einmal aufgerufen. Dies ist[4] die einzige Stelle in der *IBM TMN WorkBench for AIX* Agentenarchitektur, an der ein regelmäßiger Funktions-/Methodenaufruf stattfinden kann[5]. Damit stellt diese Positionierung der `doPolling`-Methode nur einen Kompromiß dar, denn die Zeit für einen vollständigen Schleifendurchlauf kann sehr schwanken: Falls eine sehr komplexe CMIS-Anforderung vom Gateways bearbeitet werden soll, verlängert sich die Zeit für diesen Schleifendurchlauf entsprechend. Benötigt beispielsweise ein CMIP-Request für seine Bearbeitung 40 Sekunden, beträgt der zeitliche Abstand zwischen zwei Methodenaufrufen von `doPolling` mindestens auch 40 Sekunden. Im anderen Extremfall kann nach einer sehr kurzen Bearbeitung eines CMIP-Requests der Abstand zwischen zwei Methodenaufrufen von `doPolling` im Sekundenbereich, eventuell sogar im Millisekundenbereich liegen.
Der Aufruf der Methode `doPolling` bewirkt, daß für jedes Listenelement und damit für jede registrierte SNMP-Tabelle, die Methode `TestToCreateMOI` aufgerufen wird, um somit die Replikation der jeweiligen SNMP-Tabelle in der Gateway-MIB anzustoßen. Die damit vollständigen Vererbungshierarchien und Enthaltenseinsbeziehungen der C++-Klassen der *Global Polling*-Komponente zeigt Abbildung 6.4:

6.3 C++-Klassen für die OSI-Klassen und OSI-Attribute

Es wurde bereits in Abschnitt 3.2.1 dargestellt, daß es die Aufgabe des *MIBcomposers* ist, die C++-Implementierungen der Managementobjekt-Klassen, die in der MIB des OSI-Agenten durch GDMO-Templates definiert werden, zu generieren. Diese Implementierung setzt sich aus einer Reihe von C++-Klassen zusammen, wobei jede OSI-Klasse und jedes OSI-Attribut von einer eigenen C++-Klasse realisiert wird.
Daraus wird sofort ersichtlich, daß eine OSI-Managementobjekt-Klasse von einer Vielzahl von C++-Klassen repräsentiert werden kann. Die OSI-Vererbungshierarchie spiegelt sich dabei in der C++-Vererbungshierarchie wieder: Die OSI-Klasse „internetSystem“ ist beispielsweise von der OSI-Klasse „top“ abgeleitet (siehe 3.1.1). Der *MIBcomposer* generiert dafür unter anderen die beiden C++-Klassen `CLASS_top` und `CLASS_internetSystem`. Die OSI-Vererbungshierarchie überträgt sich auf die C++-Klassen, indem die `CLASS_internetSystem` von `CLASS_top` abgeleitet wird.
Diese generierten C++-Klassen stellen die Implementierung von Mangementob-

[4] nach Wissen des Autors
[5] Es standen bei der Implementierung keine Threads zur Verfügung.

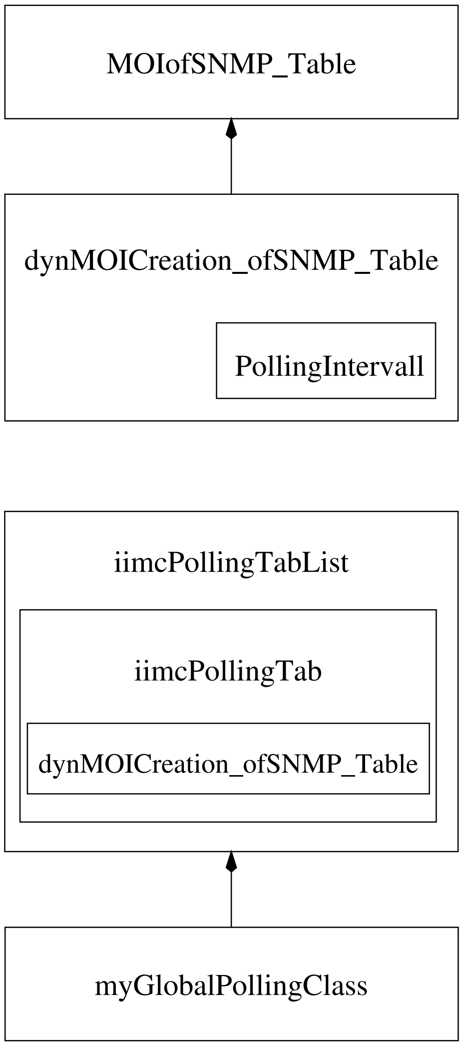


Abbildung 6.4: Die vollständige Vererbungs- und Enthaltenseinshierarchien der C++-Klassen der *Global Polling*-Komponente

jekten dar und deren Instanzen haben damit die Aufgabe, Managementoperationen auszuführen. Das Auslösen einer Managementoperation beruht dabei auf dem Aufrufen einer oder mehrerer Mehtode(n) dieser C++-Klassen. Diese Methoden basieren auf den schon vorgestellten *Callbacks*.
Ein vereinfachtes Beispiel soll die Zusammenhänge verdeutlichen: Ein GET-Request eines OSI-Managers auf einer beliebigen Instanz einer OSI-Klasse bewirkt einen Methodenaufruf (`common_get`) in der diese OSI-Klasse repräsentierenden C++-Klasse. Diese Methode realisiert das Aufrufen verschiedener *Callback*-Routinen, die schließlich für das Lesen von Attributwerten verantwortlich sind[6].
So generiert der *MIBcomposer* für alle möglichen CMIS-Anforderungen entspre-

[6]Für detailliertere Beschreibungen des Sachverhalts siehe [IBMWBMC95]

chende Methodenaufrufe in den C++-Klassen. Diese Methoden enthalten je nach Anforderung verschiedene *Callbacks*. Die Aufgabe eines Agentenentwicklers besteht in der Implementierung dieser *Callbacks*. Damit wird auch deutlich, daß ein Entwickler zu einer Managementobjekt-Klasse nicht alle *Callbacks* realisieren muß: Im Extremfall, falls er gar keinen *Callback*-Code erstellt, generiert der *MIB-composer* alle notwendigen C++-Klassen und damit die den möglichen CMIS-Requests entsprechenden Methoden[7]. Diese Methoden rufen die „leeren“ *Callbacks* auf, damit wird kein „spezielles Verhalten“ ausgeführt, der entsprechende CMIS-Request kann aber vollständig abgearbeitet werden.
Bei der Implementierung des Gateways mußten für alle OSI-Klassen und OSI-Attribute, die aus der Übersetzung der Internet-MIBs in OSI-MIBs entstanden sind, die entsprechenden *Callback*-Routinen programmiert werden.

6.3.1 Callbacks der OSI-Attribute, die SNMP-Variablen repräsentieren

Diese Attribute repräsentieren SNMP-Variablen (siehe 5.2.1), wobei der Zugriff auf ein Attribut direkt in SNMP-PDUs übersetzt wird (stateless Ansatz). Es gibt nur zwei mögliche Zugriffe auf eines dieser Attribute: einen lesenden und einen schreibenden Zugriff.
Damit sind hauptsächlich zwei *Callbacks* von besonderem Interesse:

1. *Real Value Access Callback*
   Er stellt den Wert einer Ressource, die durch dieses Attribut repräsentiert wird, während einer CMIS M-GET-Anforderung zur Verfügung.

2. *Real Value Update Callback*
   Er verändert den Wert einer Resource, die durch dieses Attribut repräsentiert wird, während einer CMIS M-SET-Anforderung.

Im *Real Value Access Callback* wurde ein `get`-Request-Methodenaufruf der *SNMP-API*-Komponente implementiert, der eine SNMP-GetRequest-PDU auslöst um damit den aktuellen Wert der zu repräsentierenden SNMP-Variable zu liefern.
Im *Real Value Update Callback* muß dagagen ein `set`-Methodenaufruf der *SNMP-API* erfolgen, um mittels einer SNMP-SetRequest-PDU den Wert der zu repräsentierenden SNMP-Variable zu verändern.

[7] und damit auch das allgemeine Verhalten das jeder OSI-Klasse und jedem OSI-Attribut automatisch zugeordnet wird (siehe 3.2.1 Unterabschnitt *MIBcomposer*)

6.3.2 Callbacks der OSI-Klassen, die SNMP-Ressourcen repräsentieren

Diese Klassen können entweder SNMP-Gruppen oder SNMP-Tabellenzeilen repräsentieren. Da das SNMP-Informationsmodell Gruppen bzw. Tabellenzeilen keine Werte bzw. Verhalten zuordnet, überträgt sich dies auch auf die OSI-Klassen. Sie dienen „lediglich" als Strukturierungs- bzw. Containerobjekte. Damit erfolgt auch keine explizite Codierung der *Callbacks*, das heißt, das allgemeine Verhalten, das jeder OSI-Klasse automatisch vom *MIBcomposer* zugeordnet wird (siehe 3.2.1 Unterabschnitt *MIBcomposer*), ist für die Repräsentation von SNMP-Gruppen bzw. Tabellenzeilen ausreichend. Es gibt jedoch eine Ausnahme: Im Abschnitt 5.2.6 wird zur Einschränkung von Inkonsistenzen in der Gateway-MIB die Aktion „aktualisiereTabelle" definiert. Diese Aktion wird in 5.2.6 bestimmten OSI-Klassen zugeordnet. Die Implementierung erfolgt mittels des *Action Callbacks*. Sobald ein OSI-Manager diese Aktion auf einer Instanz einer OSI-Klasse auslöst, die diese Aktion unterstützt, wird der *Action Callback* aufgerufen. Dieser *Callback* ruft die Methode `CreateMOI` der Klasse `dynMOICreation_ofSNMP_Table` aus 6.2 auf und führt somit die Replikation einer SNMP-Tabelle in der Gateway-MIB aus.

Im Übersetzungsalgorithmus in Abschnitt 3.1.1 wird definiert, daß der *Relative Distinguished Name (RDN)* einer Instanz einer Klasse, die eine SNMP-Gruppe repräsentiert, zu ASN.1 NULL wird. Dies hat sich in der Praxis als ungünstig herausgestellt, da die Managementplattform *IBM TMN Support Facility for AIX* (=OSI-Manager) zu sämtlichen Instanzen nur den RDN darstellt und nicht zusätzlich die Managementobjekt-Klasse der Instanz. Damit werden alle Instanzen, die eine SNMP-Gruppe repräsentieren mit „0" bezeichnet und können nicht unterschieden und damit nur schwer ihrer Managementobjekt-Klasse zugeordnet werden. Aus diesen Gründen wurden die übersetzten OSI-MIBs dahingehend verändert, daß dieser RDN als „GraphicString" definiert und ihm der OSI-Klassenname zugewiesen wurde. Damit ist nun eine sofortige Unterscheidung der Instanzen und eine Zuordnung zu ihrer jeweiligen Managementobjekt-Klasse möglich.

Weiterhin wird im Übersetzungsalgorithmus in Abschnitt 3.1.1 definiert, daß der RDN einer Instanz einer OSI-Klasse, die eine SNMP-Tabellenzeile repräsentiert, sich aus dem Index dieser SNMP-Tabellenzeile zusammensetzt. So wird zum Beispiel für die SNMP-Tabelle „tcpConnTable" der Index für eine Tabellenzeile dieser Tabelle aus den folgenden vier SNMP-Variablen gebildet:

```
INDEX { tcpConnLocalAddress, tcpConnLocalPort,
        tcpConnRemAddress, tcpConnRemPort }
```

Der Übersetzungsalgorithmus bildet daraus einen ASN.1 Datentyp für das *Naming Attribut* der OSI-Klasse „tcpConnEntry", die eine Tabellenzeile dieser

SNMP-Tabelle „tcpConnTable“ repräsentieren soll :

```
TcpConnEntryIdValue ::=
        SEQUENCE {
                tcpConnLocalAddress   [1] IpAddress,
                tcpConnLocalPort      [2] Integer64k,
                tcpConnRemoteAddress  [3] IpAddress,
                tcpConnRemotePort     [4] Integer64k
        }
```

Um nun eine Instanz der Klasse „tcpConnEntry“ kreieren zu können, muß der RDN dieser Instanz festgelegt werden. Dazu muß dieser ASN.1 Datentyp (`TcpConnEntryIdValue`) auf eine C++-Datenstruktur abgebildet werden. Weiterhin werden für diese neue C++-Datenstruktur mindestens die Methoden Konstruktur bzw. Destruktor und ein Vergleichsoperator benötigt. Damit müßte nun für jede SNMP-Tabelle und somit für den, den INDEX repräsentierenden, ASN.1 Datentyp eine C++-Datenstruktur aufgebaut werden. Aufgrund der verschiedenen C++-Datenstrukturen folgt eine unterschiedliche Bearbeitung (Replikation) jeder SNMP-Tabelle.
Dies wurde bei dieser Implementierung vermieden: Die verschiedenen ASN.1 Datentypen für die jeweiligen *Naming Attribute* wurden alle zu einem `GraphicString`. Der Wert dieses `GraphicStrings` setzt sich nun aus der „dotted Notation“ des Indizes der SNMP-Tabellenzeilen zusammen, wobei folgenden Beispiel den Sachverhalt verdeulichen soll:

SNMP-OID von „tcpConnTable“: `.1.3.6.1.2.6.13`
SNMP-OID von „tcpConnEntry“: `.1.3.6.1.2.6.13.1`
SNMP-OID der ersten Tabellenspalte: `.1.3.6.1.2.6.13.1.1`

Eine *GetNextRequest-PDU* auf die OID `.1.3.6.1.2.6.13.1` ergibt:
Unter anderem enthält die *GetResponse-PDU* auch den OID der ersten Tabellenzeile:
`.1.3.6.1.2.13.1.1.{´dotted Notation´ des Index}`
Die „dotted Notation“ wird schließlich zum Inhalt des *Naming Attributs.*

Dies hat vor allem diesen Vorteil:
Dadurch, daß alle *Naming Attribute* vom gleichen Typ (`GraphicString`) sind und dieser einfach in C++ als Zeiger auf `char` dargestellt werden kann, ist die Kodierung für das Zusammensetzen des RDN für alle *Naming Attribute* gleich. Dies stellt eine enorme Vereinfachung der Implementierung dar und wurde deswegen auf diese Weise realisiert.

6.4 Automatische Code-Generierung für die OSI-Klassen und OSI-Attribute

Wie im vorherigen Abschnitt beschrieben wurde, implementiert der Agentenentwickler ausschließlich *Callbacks*. Der *MIBcomposer* generiert daraus C++-Klassen und damit die Implementierung aller Managementobjekt-Klassen, die in der MIB des Agenten durch GDMO-Templates definiert werden.
Jeder einzelne *Callback* wird dabei in einer eigenen Datei im Verzeichnis „workspace" gespeichert, zum Beispiel die beiden Dateien der in 6.3.1 vorgestellten *Callbacks* des Attributs *sysContact*:

```
MIB2.sysContact.real_access.cpp.txt.cbk
MIB2.sysContact.real_update.cpp.txt.cbk
```

Der mit dem *MIBcomposer* daraus generierte C++-Code der Managementobjekt-Klassen befindet sich im Verzeichnis „build".
Die Funktionsweise eines Attribut-Lese-Zugriffs läuft für alle OSI-Attribute, die SNMP-Variablen repräsentieren, gleich ab: Es wird die Internet-Adresse, der Community-String und die OID der gewünschten SNMP-Variable benötigt. Ebenso verhält es sich bei einem Schreib-Zugriff.
Der C++-Programmcode für die beiden benötigten *Callbacks* (*Real Value Access* und *Real Value Update*) konnten nun so implementiert werden, daß die erforderlichen Parameter zur Laufzeit bestimmt werden. Daraus resultiert, daß für alle OSI-Attribute, die SNMP-Variablen repräsentieren, die gleichen *Callback*-Codes verwendet werden können. Diese *Callbacks* werden nun in gesonderten Dateien

```
muster.real_access.cpp.txt.cbk
muster.real_update.cpp.txt.cbk
```

gespeichert. Die ursprünglichen *Callback*-Dateien der Attribute werden nun zu Links auf diese gesonderten Dateien:

```
MIB2.sysContact.real_access.cpp.txt.cbk ->
                muster.real_access.cpp.txt.cbk
MIB2.sysContact.real_update.cpp.txt.cbk ->
                muster.real_update.cpp.txt.cbk
```

Dies hat entscheidende Vorteile:
Für eine große Anzahl an Attributen muß nur eine kleine Menge an *Callbacks* in den gesonderten Dateien gehalten werden. Ist irgendeine Änderung im Code notwendig, muß diese nur an einer zentralen Stelle vorgenommen werden und nicht für jedes Attribut einzeln erfolgen. Außerdem können so für neue MIBs und damit für neue Attribute lediglich durch Erstellen von weiteren Links diese Attribute codiert werden.

Die gleiche Situation tritt für den *Callback* ein, der die Aktion „aktualisiereTabelle“ implementiert. Er wurde so programmiert, daß er für alle OSI-Klassen identisch ist. Somit kann er ebenso in einer gesonderte Datei gespeichert und mit Links den jeweiligen OSI-Klassen zugeordnet werden.
Der *MIBcomposer* greift nun über diese Links auf den *Callback*-Code zu und generiert daraus schließlich den C++-Code der Managementobjekt-Klassen.
Für die automatische Erstellung der Links im Verzeichnis „workspace“ wurde ein Werkzeug entwickelt: *erstelleLinks*. Es enthält in einer Liste alle gesonderten Dateien mit den für alle Klassen bzw. Attribute, die SNMP-Ressourcen repräsentieren, notwendigen *Callbacks*.
Anhand der in der *Metadata Database* eingetragenen OSI-Klassen und OSI-Attributen kann dieses Werkzeug erkennen, welche SNMP-Ressourcen repräsentieren und erstellt für jedes dieser Klassen und Attribute, auf der Grundlage dieser Liste, die entsprechenden Links auf den *Callback*-Code in den gesonderten Dateien.

6.5 Erfahrungen mit der Entwicklungsumgebung

In diesem letzten Abschnitt des Kapitels sollen kurz Erfahrungen des Autors beim Einsatz der *IBM TMN WorkBench for AIX* beschrieben werden:

1. Der *Managed Object Compiler* kennt keine ASN.1 Macros. Dies stellt eine gewisse Einschränkung der Mächtigkeit dieses Werkzeuges dar.

2. Ein unangenehmes Problem tritt beim Arbeiten mit dem *MIBcomposer* auf: Dieser generiert (siehe Abbildung 3.2.3) aus den GDMO-Templates, den ASN.1-Typdefinitionen und dem *Callback*-Code die C++-Implementierungen der Managementobjekt-Klassen eines OSI-Agenten. Eine weitere Aufgabe des *MIBcomposers* ist es, diesen erzeugten Code zu kompilieren. Dies beruht im wesentlichen auf dem Ausführen eines *Makefiles*.
   Nachdem nun der *MIBcomposer* gestartet wurde, und Änderungen im *Callback*-Code eingebracht wurden, kompiliert der *MIBcomposer* den kompletten C++-Code und nicht nur die C++-Dateien, in welchen die Veränderungen stattfanden. Nachdem also der gesamte Agent einmal kompiliert wurde, arbeitet der *MIBcomposer* „normal“, das heißt, es werden nur die C++-Dateien kompiliert, in welchen auch wirklich nur Veränderungen stattfanden. Dies führte zu langen Wartezeiten, da jeweils beim ersten Aufrufen des *MIBcomposers* der gesamte Agent kompiliert wurde und dies etwa eine Stunde erforderte.

3. Die Repräsentation von SNMP-Tabellenzeilen durch Instanzen von Managementobjekt-Klassen im Gateway hat gezeigt, daß für die Replikation ein

Polling-Mechanismus notwendig war. Eine Folge daraus ist, daß gewisse Inkonsistenzen zwischen den OSI-Instanzen in der Gateway-MIB und den zu repräsentierenden SNMP-Tabellenzeilen unvermeidbar sind. Diese Problematik tritt aber für alle OSI-Agenten auf, die (dynamische) Managementinformationen durch Instanzen von Managementobjekt-Klassen repräsentieren müssen, wobei als weitere Voraussetzung diese Managementinformation keine Mitteilung bei einer Veränderung auslöst. Damit wird wiederum die Implementierung eines Polling-Mechanismus benötigt und mögliche Inkonsistenzen können nicht vermieden werden.

4. Durch die Reduzierung einer OSI-Agentenentwicklung auf die Implementierung von *Callbacks*, wird ein Programmierer von Protokollspezifikas ferngehalten. Funktionalitäten wie zum Beispiel für das Auflösen von Scopes und Filter müssen nicht vom Agentenentwickler beachtet werden, sondern werden automatisch ausgeführt. Dadurch reduziert sich sowohl der Einarbeitungsaufwand wie auch die Entwicklungsdauer erheblich.

5. Zuletzt sollen noch einige Zahlen den Umfang des implementierten CMIP/SNMP Gateways beschreiben: Das Gateway kann einem OSI-Manager die Überwachung und Steuerung von SNMP-Agenten ermöglichen, die entweder die *MIB-2* oder die *LRZ Systemagenten MIB* oder beide unterstützen. Dazu werden ca. 60 verschiedene OSI-Managementobjekt-Klassen und weit über 600 OSI-Attribute für die Repräsentation von SNMP-Ressourcen vom Gateway bereitgestellt. Das Binary des Gateways besitzt eine Größe von über 39 MByte.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Aufgrund der Existenz von verschiedenen Managementarchitekturen und der immer stärker werdenden Forderung nach einheitlichem Management werden die Integration dieser und deren Interoperabilität zu entscheidenden Voraussetzungen. Das Ziel dieser Diplomarbeit war es, mittels eines *CMIP/SNMP Gateways* das Internet-Management in die OSI-Managementarchitektur zu integrieren und damit eine OSI-Sichtweise auf SNMP-Ressourcen bereitzustellen.
Dazu wurde in Kapitel 4 festgestellt, daß sowohl die Informations- als auch die Kommunikationsmodelle beider Architekturen aufeinander abgebildet werden müssen.
Für die Informationsmodellabbildung wurden zwei Möglichkeiten diskutiert. Wegen der fehlenden Automatisierbarkeit und der nicht realisierbaren Standardisierung der *abstrakten Übersetzung* wurde schließlich für die Architektur des Gateways die *direkte Übersetzung* gewählt. Diese basiert auf einem in den Arbeiten der IIMC veröffentlichten und vom NMF standardisierten Algorithmus.
Bei der Kommunikationsmodellabbildung waren ebenfalls zwei Ansätze von Bedeutung: Anhand der Managementinformationshierarchie wurde zwischen einem *stateless* und einem *stateful* Gateway unterschieden. Als Fazit für die Kommunikationsmodellabbildung wurde festgestellt, daß ein optimales, idealisiertes Gateway einen Kompromiß eingehen würde, der die Vorteile jedes Ansatzes integriert und die Nachteile vermeidet: für dynamische Mangementinformation würde es sich wie ein stateless Gateway, für statische Managementinformation würde es sich wie ein stateful Gateway verhalten.
In Kapitel 5 wurde die Architektur eines CMIP/SNMP Gateway erstellt. Die Aufgabe bestand darin, die in Kapitel 4 vorgestellte Top-Down Sichtweise eines Mangementgateways mit der in Kapitel 3 beschriebenen Entwicklungsumgebung (Bottom-Up Sichtweise) zu verbinden. Das heißt, die in Kapitel 4 vorgestellten Informations- bzw. Kommunikationsmodellabbildungen wurden auf die Agenten-

architektur der *IBM TMN WorkBench for AIX* projiziert. Daraus resultiert folgende Gatewayarchitektur:
All jene SNMP-Ressourcen, welche aufgrund der Einschränkungen der *IBM TMN WorkBench for AIX* nicht direkt auf SNMP-PDUs abgebildet werden können (dem entsprechen SNMP-Gruppen und -Tabellenzeilen), weil sie durch Instanzen im Gateway repräsentiert werden müssen (und somit die Containment-Hierarchie aufbauen), sollen nach dem stateful Ansatz implementiert werden. Alle anderen SNMP-Ressourcen (dem entsprechen alle SNMP-Variablen, sie werden durch Attribute in den OSI-Klassen dargestellt) sollen nach dem stateless Ansatz realisiert werden, das heißt, der Wert eines OSI-Attributs soll direkt aus dem SNMP-Agenten gelesen werden.
Eine Beschreibung der Implementierung dieses CMIP/SNMP Gateways erfolgte in Kapitel 6.
Das Gateway stellt einem OSI-Manager eine dem OSI-Informationsmodell-konforme Sichtweise auf die Internet-MIBs *MIB-2* und *LRZ Systemagenten MIB* bereit. Es können auf relativ einfache Weise weitere Internet-MIBs eingebunden werden. Dazu muß das Gateway jedoch neu kompiliert werden.

7.2 Ausblick

Einige Konzepte des CMIP/SNMP Gateways müssen sicherlich an einigen Stellen erweitert werden. Dazu zählen beispielsweise die Sicherheitsaspekte und die Fehlerbehandlung. Weiterhin wird sich in der Praxis zeigen, inwieweit die implementierte Polling-Strategie geeignet ist, die SNMP-Managementinformationen im Gateway zu replizieren. Eventuell muß diese Strategie noch verbessert werden. Ein schon erwähnter Übergang zu einem reinen stateful Gateway bedarf noch detaillierterer Diskussionen, gerade im Hinblick auf den vielleicht zu erwarteten Performance-Vorteil, der aber mit dem Preis von Inkonsistenzen zwischen der Gateway-MIB und den MIBs der SNMP-Agenten erkauft wird.

Die folgenden beiden Problemstellungen betreffen die Erweiterung der Funktionalität und das Umfeld des Gateways. Sie können als Anregung dienen, diese Thematik, eventuell in weiteren Diplomarbeiten, zu vertiefen:

1. Das Gateway soll beliebige Internet-MIBs überwachen und steuern können: Dazu soll die jeweilige neue Internet-MIB dynamisch vom Gateway mittels Instanzen von „generischen“ OSI-Klassen modelliert werden. Die dabei auftretenden Fragestellungen sind, ob dies überhaupt möglich ist und wenn ja, wie diese „generischen“ Klassen auszusehen haben.

2. Für das OSI-Management stehen noch sehr wenig Managementanwendungen zur Verfügung. So kann als ein erster Schritt das Gateway an die Manage-

mentplattform *IBM TMN Support Facility for AIX* gekoppelt werden, um zum Beispiel eine Sicht auf alle gerade vom Gateway überwachten SNMP-Agenten mit ihren MIBs zu geben. Dazu müßte das Gateway dahingehend verändert werden, daß es bei jedem Kreieren bzw. Löschen von Instanzen von Managementobjekt-Klassen eine Notifikation auslöst. Ein Dämonprozeß könnte durch das Kreieren eines EFDs (Event Forwarding Discriminator) im Gateway alle diese Notifikationen anfordern. Für jede kreierte Instanz würde er in einer speziell für das Management mit dem Gateway erstellten Topologie-Map ein Icon erzeugen. Ebenso würde er für jede gelösche Instanz in der Map das entsprechende Icon entfernen. Ein Benutzer könnte weiterhin durch einen „Doppelklick“ auf ein Icon sämtliche Attribute der Instanz der OSI-Klasse, die dieses Icon repräsentiert, darstellen.

Zuletzt soll noch angemerkt werden, daß ein produktreifes CMIP/SNMP Gateway das heutzutage weitverbreitete Management von TCP/IP-Rechnernetzen entscheidend verändern könnte: Das Internet-Management gilt trotz des de-facto-Standards im Bereich des Managements der TCP/IP-Rechnernetze als unzureichend. Es ist für komplexe Managementaufgaben in großen und heterogenen Rechnernetzen zu wenig mächtig. Mangelnde Flexibilität und das Fehlen eines Domänenkonzepts sowie eines leistungsfähigen Funktionsmodell sind unter anderem Begründungen dafür. Durch die Integration des Internet-Management in die OSI-Managementarchitektur mittels eines CMIP/SNMP Gateways können viele der gerade aufgezählten Nachteile vermieden werden: Die mächtigen und flexiblen Funktionen der integrierenden Architektur wie zum Beispiel die Managementobjektauswahl durch Scoping und Filtering, die Verwendung sämtlicher SMFs und damit die Existenz eines mächtigen Funktionsmodells und ein Domänenkonzept können auf die Ressourcen der integrierten Architektur angewendet werden. Es muß allerdings noch erwähnt werden, daß dadurch nicht das Problem des unzureichenden Sicherheitskonzepts von SNMP gelöst wird.
Durch die Bereitstellung von relevanten Managementinformationen durch relativ einfache SNMP-Agenten können diese in entsprechenden OSI-Managementanwendungen auf sehr hohe und abstrakte Weise verarbeitet werden. Denkbar sind dabei beispielsweise folgende Anfragen:

- An welchen Rechnern ist die Auslastung im Zeitraum zwischen 20 Uhr abends und 7 Uhr morgens kleiner als 50% ?

- Welche Mitarbeiter arbeiten weniger als 5 Stunden am Tag mit ihrem Terminal/PC ?

- Wieviele Rückweisungen gab es in einem bestimmten Zeitraum aufgrund zu hoher Auslastung beim Zugriff auf eine Datenbank?

Als weiterer Schritt wäre die Definition von Zielvorgaben und deren Durchsetzung denkbar. Dies ist auch unter dem Begriff „Management Policies“ bekannt und Gegenstand der aktuellen Forschung.

Anhang A

Installation, Konfiguration und Starten des Gateways

A.1 Die Installation des Gateways

Für den Betrieb des implementierten CMIP/SNMP Gateways werden die beiden Softwarepakete

1. **IBM TMN Support Facility for AIX**, Release 2 und
2. **IBM TMN WorkBench for AIX**, Release 2

benötigt. Die Implementierung des Gateways befindet sich im Verzeichnis `tmndev/CmipSnmpProxy`. Es existieren darin eine Reihe von Unterverzeichnissen (siehe Abbildung A.1):

Verzeichnis	Beschreibung
agent	In diesem Verzeichnis sind • das Binary des *Agent*-Prozesses, • ein Makefile, • das *Local Registration File (LRF)* (`agent.lrf`), für die Registrierung von Managementobjekt-Klassen in der *ORS-Datenbank* (siehe A.2.3), • eine Datei mit der `main`-Funktion des *Agent*-Prozesses und • weitere Dateien, die für das Starten des *Agent*-Prozesses notwendig sind enthalten.

build — Der *MIBcomposer* speichert hier sämtlichen erstellten C++-Quellcode des Gateways.

documents — In diesem Verzeichnis sind alle GDMO-, ASN.1-, OMP- und DEF-Dateien gespeichert, auf denen das Gateway aufbaut. Hier müssen sich auch die GDMO- und ASN.1-Dateien befinden, die aus der Übersetzung von Internet-MIBs in OSI-MIBs entstanden sind. So existieren hier für die *MIB-2*:

```
iimc1213.gdm
iimc1213.asn
iimc1213.omp
```

Und für die *LRZ Systemagenten MIB*:

```
sysAgent.gdm
sysAgent.asn
sysAgent.omp
```

Weiterhin werden in den Dateien

```
cmipsnmpProxy.gdm
cmipsnmpProxy.asn
cmipsnmpProxy.omp
```

die GDMO- und ASN.1-Definitionen für die „local Objects“ 5.2.5 gespeichert.

run — Hier werden alle die Dateien gehalten, die für das Starten des Gateways benötigt werden.

scripts — Dieses Verzeichnis enthält Dateien für den *MIBscripter*, um entsprechende Instanzen von Managementobjekt-Klassen im Gateways kreieren zu können. Beispiele dafür sind `ibm.script` und `sun7.script`, um entweder die Internet-MIB der `ibmhegering1` oder der `sunhegering7` für das OSI-Management vom Gateways bereitzustellen.

source — In diesem Verzeichnis werden weitere C++-Klassen definiert und implementiert.

workspace — Hier wird vom *MIBcomposer* der *Callback*-Code gespeichert. Es enthält weiterhin das Programm `erstelleLinks` (siehe 6.4), um automatisch den Code für die OSI-Klassen und OSI-Attribute, die SNMP-Ressourcen repräsentieren, generieren zu können.

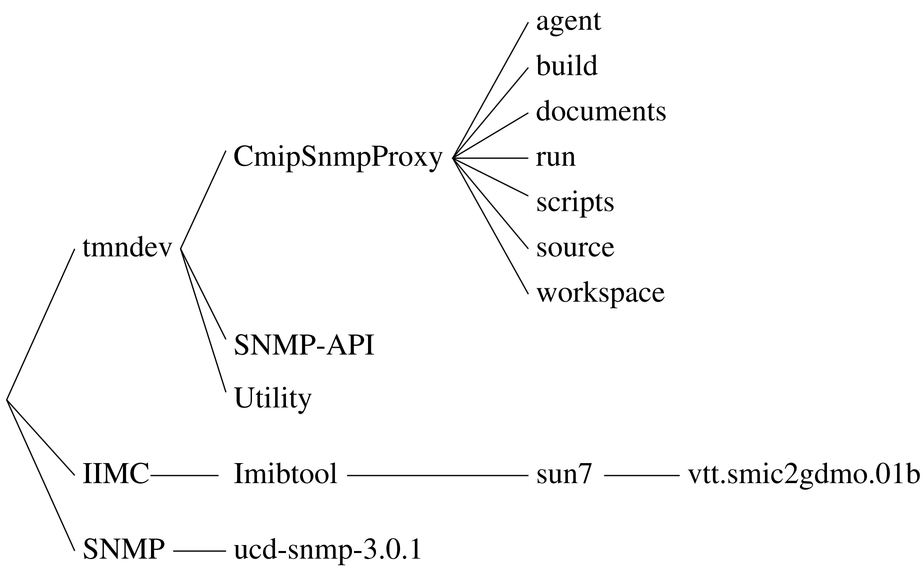


Abbildung A.1: Die Verzeichnisstruktur der Implementierung

Im Verzeichnis `SNMP/ucd-snmp-3.0.1` befindet sich die *ucd-Programmierschnittstelle*. Diese wird von der C++-Klasse `SNMP` gekapselt, welche im Verzeichnis `tmndev/SNMP-API` implementiert ist. Weiterhin werden im Verzeichnis `tmndev/Utility` allgemeine Funktionen bereitgestellt.
Eine Installation des MIB-Compilers *SMIC2GDMO* für SunOS befindet sich in `IIMC/Imibtool/sun7/vtt.smic2gdmo.01b`.

A.2 Die Konfiguration des Gateways

Die Konfiguration des Gateways besteht im wesentlichen aus drei Schritten, dem Aktualisieren sowohl von Initialisierungsdateien und *XMP*-Libraries als auch der beiden Datenbanken *Metadata Database* und *ORS-Datenbank*.

A.2.1 Initialisierungsdateien und *XMP*-Libraries

Die beiden Prozesse *Infratop* und *Discr* des Gateways benötigen im `run`-Verzeichnis eine Reihe von Initialisierungsdateien. Das Shellskript `zMetadata` erstellt (bzw. erneuert) automatisch diese Initialisierungsdateien, wobei dazu die Dokumente in dem Verzeichnis `documents` erfolderlich sind. Weiterhin werden von

diesem Skript die benötigten *XMP*-Libraries erstellt.
Mit dem Befehl

```
cd tmndev/CmipSnmpProxy
/usr/TMN/WorkBench/zMetadata -d documents -r run
```

werden auf der Grundlage der Dokumente im Verzeichnis `documents` alle benötigten Initialisierungsdateien (`*.dat`, `xmpcfg.libraries` und *XMP*-Library Dateien) im Verzeichnis `run` erstellt.
Als *Root* werden mit dem Befehl

```
cd tmndev/CmipSnmpProxy
/usr/TMN/WorkBench/zMetadata -d documents -x
```

die *XMP*-Libraries im Verzeichnis `/usr/OV/lib` aktualisiert. Detailliertere Informationen befinden sich in [IBMPrUG95] auf der Seite 19 und ab Seite 55.

A.2.2 Die *Metadata Database*

Für den in 5.2.1 beschriebenen und in 5.2.2 erweiterten Algorithmus wird vorausgesetzt, daß sämtlich existierenden Managementobjekt-Klassen mit ihren NAME BINDINGs in der *Metadata Database* eingetragen sind. Für den Ablauf des Speicherns von GDMO- und ASN.1-Dokumenten in diese Datenbank wird auf [IBMWBPG95] bzw. auf [IBMWBCB95] verwiesen.

A.2.3 Die *ORS-Datenbank*

Sobald ein OSI-Manager auf eine bestimmte Instanz einer Managementobjekt-Klasse zugreifen will, muß er zuerst feststellen, zu welchem Agenten er die Anfrage schicken soll. Diese Abbildung erfolgt in der *ORS-Datenbank* (Object Registration Service). Dort wird zu jeder Instanz eines Managementobjekts

- der AET (Application Entity Title) und die Adresse des Agenten, der die angesprochene Instanz verwaltet,
- die Managementobjekt-Klasse der Instanz
- und weitere Informationen

gespeichert.
Falls ein Agent eine zusätzliche Klasse verwalten soll, oder der Agent noch gar keine Klasse in der *ORS-Datenbank* gespeichert hat, muß diese Datenbank mit dem Befehl `ovaddobj` aktualisiert werden (siehe [IBMPrUG95], Seite 32 und [IBMPrIC95]). Dazu wird die Datei `agent.lrf` im Verzeichnis `tmndev/CmipSnmpProxy/agent` benötigt. Dieses *Local Registration File (LRF)* wird automatisch beim Binden des Gateways mit dem Befehl `zBuild` (siehe auch

[IBMPrUG95], Seite 33 bzw. Seite 73) erstellt und enthält alle Informationen für die Registrierung von Managementobjekt-Instanzen in der *ORS-Datenbank*. Für die Ausführung dieses Befehls werden *Root*-Rechte benötigt:

```
cd tmndev/CmisSnmpProxy/agent
ovaddobj -O agent.lrf :psel1,ssel1,tsel1,T129.187.214.19+102
```

A.3 Das Starten und Beenden des Gateways

A.3.1 Das Gateway starten

Um das Gateway zu starten, sind folgende Schritte notwendig (siehe auch [IBMPrUG95], Seite 28 bzw. Seite 59):

1. Die folgende Umgebungsvariable muß in den `PATH` aufgenommen werden:

   ```
   export ZSMESF=/usr/TMN/SupportFacility
   export PATH=$PATH:$ZSMESF/bin
   ```

2. In das Verzeichnis `tmndev/CmipSnmpProxy` wechseln.

3. Den Befehl `zStart` ausführen.

Danach sollte ein Ausschnitt des Befehls `ps` erscheinen und die Existenz der folgenden vier Prozesse aufzeigen (beispielhaft):

```
langerm    112  0.2 11.1  702  808 pp0 S    13:59   0:20 agent
langerm    322  0.2  0.0 1506    0 pp0 S    14:04   0:19 discr
langerm    761  0.3  3.1  169  228 pp1 S    15:12   0:18 logr
langerm   1447  0.0  3.0   73  224 pp0 R    16:39   0:00 infratop
```

Das Gateway steht nun bereit, SNMP-Ressourcen einem OSI-Manager zur Verfügung zu stellen. Da aber zu diesem Zeitpunkt noch kein zu überwachender Host und damit SNMP-Agent ausgewählt wurde, existieren noch keine Instanzen von Managementobjekten im Gateway, die SNMP-Ressourcen repräsentieren. So wurde für die schon oben erwähnten Hosts `ibmhegering1` und `sunhegering7` jeweils ein Skript erstellt, welches im Gateway eine Instanz der Klasse „cmipSnmpProxyAgent“ (siehe 5.2.5) kreiert. Damit wird die jeweilige Internet-MIB des Hosts für das OSI-Management bereitgestellt. Der Syntax lautet (Der Befehlsaufruf muß vom Verzeichnis `tmndev/cmipSnmpProxy` aus erfolgen):

```
MIBscripter -w 30 -v scripts/ibm.script
MIBscripter -w 30 -v scripts/sun7.script
```

Anschließend können von einem OSI-Manager aus die SNMP-Ressourcen überwacht und gesteuert werden. Ein Beispiel für einen OSI-Manager stellt der `mbe` dar (kann auch von der *IBM TMN Support Facility for AIX* Plattform aus aufgerufen werden):

1. `mbe` aufrufen.
2. MOC (`DMI.system`) und MOI

   `ibm500.genericNetworkId="TelcoNet";systemId=name:"TelcoSys"`

   eingeben (siehe Abbildung A.2).

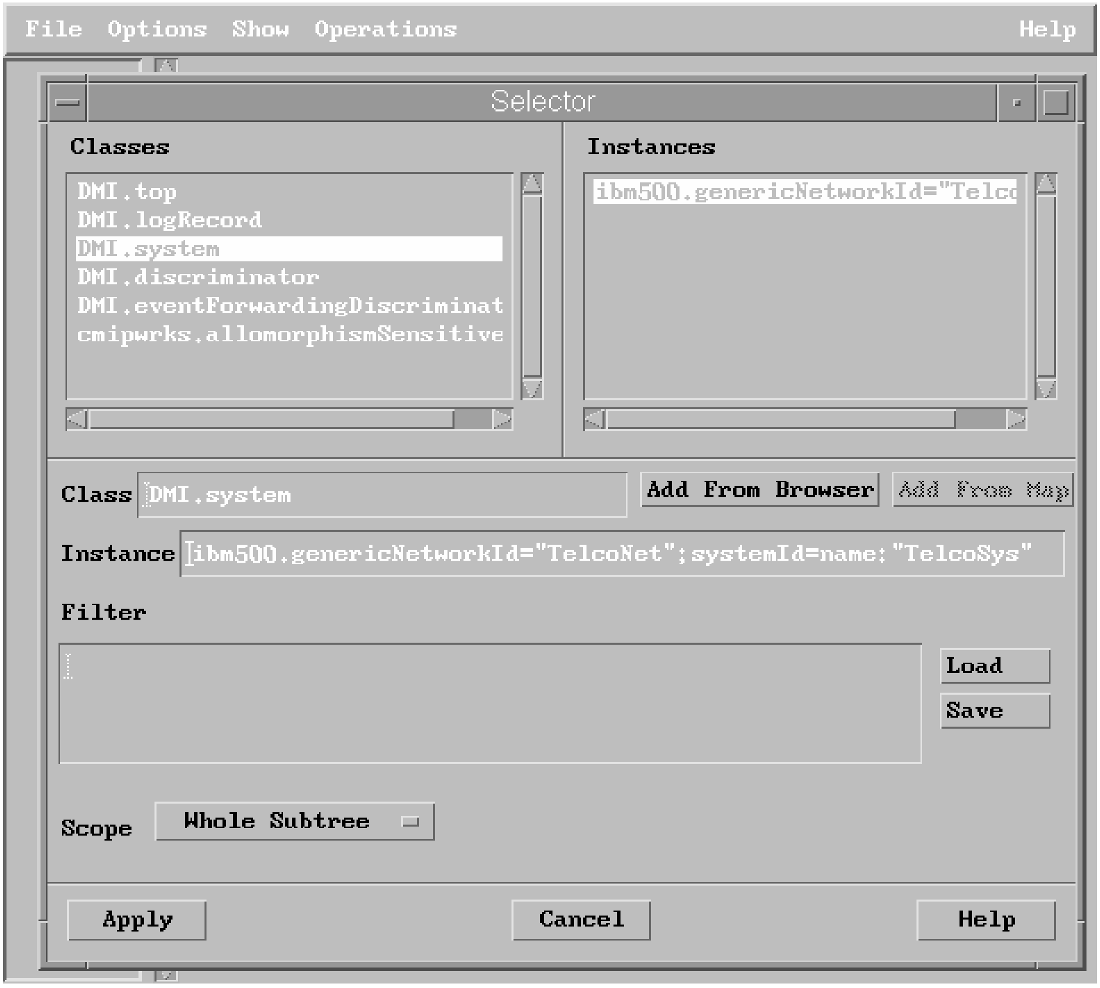


Abbildung A.2: Auswahl eines Root-Mangementobjekts für die Containment-Hierarchie

3. Den Scope `Whole Subtree` auswählen.

4. Mit `Apply` bestätigen.

Dadurch wird im Containment-Fenster die Enthaltenseinshierarchie vollständig angezeigt.

A.3.2 Das Gateway beenden

Falls ein gestartetes Gateway beendet werden soll, müssen alle vier Prozesse (*Infratop*, *Agent*, *Discr* und *Logr*) gestoppt werden. Das Beenden des Prozesses *Infratop* (mit `kill [PID von Infratop]`) bewirkt, daß automatisch die anderen drei Prozesse ebenfalls terminieren.

Anhang B

Einbinden neuer Internet-MIBs in das Gateway

Für das Einbinden einer neuen Internet-MIB in das Gateway, das heißt, dem Bereitstellen einer neuen Internet-MIB für das OSI-Management, werden folgende Schritte benötigt:

1. Das Übersetzen der neuen Internet-MIB mit dem in 3.1.2 vorgestellten MIB-Compiler *SMIC2GDMO* zu einer OSI-MIB (GDMO- bzw. ASN.1-Dateien):
   (a) Erstellen der benötigten C-Datenstrukturen, siehe 3.1.2 bzw. [Reilly].
   (b) Das Ausführen der Kompilierung.
   (c) Die Nachbearbeitung der NAME BINDINGs bzw. der Registrierungs-OIDs, siehe 3.1.2 und [Reilly93].
2. Überprüfung dieser erzeugten GDMO- bzw. ASN.1-Dateien mit dem *Managed Object Browser and Editor* und Eintragung der Dokumente in die *Metadata Database*, siehe 3.2.1, A.2.2 und den Entwicklungsablauf in Abbildung 3.6.
3. Automatische Link-Erzeugung mit `erstelleLinks`, siehe 6.4.
4. Aktualisieren des *Definition Files* `cmipsnmpProxy.def`, siehe [IBMWBMC95] Seite 39.
5. Aufrufen des *MIBcomposers*, siehe [IBMWBMC95] Seite 25.
   (a) Neue OSI-Klassen dem bestehenden *Ensemble* `osi_snmp_gateway` hinzufügen, siehe [IBMWBMC95] Seite 41.
   (b) Code, Makefile und Konfigurationsdateien generieren, siehe [IBMWBMC95] ab Seite 123.
   (c) Den gerade erstellten Code kompilieren.

6. Erstellen der Initialisierungsdateien und *XMP*-Libraries, siehe A.2.1.

7. Mit dem Befehl `zBuild` den erstellten Code binden, siehe [IBMPrUG95] Seite 21.

8. Zuletzt muß noch die *ORS-Darenbank* aktualisiert werden, siehe auch A.2.3. Dazu werden unter *Root*-Berechtigung die neuen Managementobjekt-Klassen mit

   ```
   cd tmndev/CmipSnmpProxy/agent
   ovaddobj -O agent.lrf :psel1,ssel1,tsel1,T129.187.214.19+102
   ```

   in die Datenbank eingetragen.

Nun kann das um die neue Intenet-MIB erweiterte Gateway gestartet werden (siehe A.3.1) und alle in den integrierten Internet-MIBs enthaltenen SNMP-Ressourcen für das OSI-Management bereitstellen.

Anhang C

Abkürzungsverzeichnis

A

AET	Application Entity Title
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
AT	Address Translation (Gruppe in der MIB-2)

B

BML	Business Management Layer (TMN-Managementarchitektur)

C

CCITT	Consulative Committee on International Telephone and Telegraphy
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CORBA	Common Object Request Broker Architekture

D

DCE	Distributed Computing Environment
DN	Distinguished Name
DME	Distributed Management Environment
DPI	Distributed Protocol Interface

E

EFD	Event Forwarding Discriminator (OSI-Managementarchitektur)
EGP	Exterior Gateway Protocol
EML	Element Management Layer (TMN-Managementarchitektur)

G

GDMO	Guidelines for the Definition of Managed Objects (OSI-Managementarchitektur)

I

IAB	Internet Activity Board
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IIMC	ISO-Internet Management Coexistence
IP	Internet Protocol
ITU	International Telecommunication Union

L

LRF	Local Registration File
LRZ	Leibniz Rechenzentrum

M

MIB	Management Information Base
MF	Mediation Function (TMN-Managementarchitektur)
MO	Managed Object
MOC	Management Object Class
MOI	Managed Object Instance

N

NEF	Network Element Function (TMN-Managementarchitektur)
NEL	Network Element Layer (TMN-Managementarchitektur)
NMF	Network Management Forum
NML	Network Management Layer (TMN-Managementarchitektur)

O

OID	Object Identificator
OMA	Object Management Architekture
OMG	Objekt Management Group
ORS	Object Registration Service
OSF	Open Software Foundation
OSF	Operations System Function (TMN-Managementarchitektur)
OSI	Open Systems Interconnection

P

PDU	Protocol Data Unit

Q

QAF	Q Adapter Function (TMN-Managementarchitektur)

R

RDN	Relative Distinguished Name
RFC	Request for Comments

RMON	Remote Network Monitoring

S

SMF	Systems Management Function (OSI-Managementarchitektur)
SMFA	Systems Management Functional Area (OSI-Managementarchitektur)
SMI	Structure of Management Information
SML	Service Management Layer (TMN-Managementarchitektur)
SNMP	Simple Network Management Protocol
SNMPv1	Simple Network Management Protocol Version 1
SNMPv2	Simple Network Management Protocol Version 2

T

TCP	Transmission Control Protocol
TMN	Telecommunication Management Network

U

UDP	User Datagram Protocol

W

WSF	Workstation Function (TMN-Managementarchitektur)

X

XMP	X/Open Management Protocol

Literaturverzeichnis

[AbClHo93] Abeck, S.; Clemm, A.; Hollberg, U.: Simply Open Network Management: An Approach for the Integration of SNMP into OSI Management Concepts. In: Proceedings of the second IFIP/IEEE International Symposium on Integrated Network Management III. H.-G. Hegering and Y. Yemini (Editors). North-Holland. IFIP 1993

[Beier93] Beier, B.: Die Integration von SNMP-Management in OSI-Netzmanagement. Diplomarbeit, Universität Karlsruhe, März 1993

[Black92] Black, U.: Network Management Standards - The OSI, SNMP and CMOL Protocols. McGraw-Hill 1992

[BrLeMa93] Brady, S.; Levine, D.W.; Mazumdar, S.: Design of Protocol Independent Management Agent to Support SNMP and CMIP Queries. In: Proceedings of the second IFIP/IEEE International Symposium on Integrated Network Management III. H.-G. Hegering and Y. Yemini (Editors). North-Holland. IFIP 1993

[Cohen94] Cohen, R.S.: The Telecommunications Management Network (TMN). In: Network and Distributed Systems Management, Edited by Morris Sloman. Addison Wesley 1994

[CoKl91] Cohen, R.S.; Klerer, S.M.: Distribution of Managed Object Fragments and Management Object Replication: The Data Distribution View of Management Information. In: Proceedings of the second IFIP/IEEE International Symposium on Integrated Network Management II. I. Krishnan & W. Zimmer (Editors). North-Holland. IFIP 1991

[FeHeNi95] Feridun, M.; Heusler, L.; Nielsen, R.: Research Report - Implementing OSI Agents for TMN. IBM Research Division November 1995

[Garbe91] Garbe, K.: Management von Rechnernetzen. B.G. Teubner 1991

[Gering93] Gering, M.: CMIP versus SNMP. In: Proceedings of the second IFIP/IEEE International Symposium on Integrated Network Management III. H.-G. Hegering and Y. Yemini (Editors). North-Holland. IFIP 1993

[GuNe95] Gutschmidt, M.; Neumair, B.: Integration von Netz- und Systemmanagement: Ziele und erste Erfahrungen. Veröffentlicht in der 3. Fachtagung für Arbeitsplatz-Rechensysteme (APS'95)

[HaHa95] Haubelt, N.; Hauck, R.: Implementierung einer MIB für Systemmanagementaufgaben. Fortgeschrittenenpraktikum, Technische Universität München, März 1995

[HeAb93] Hegering, H.-G.; Abeck, S.: Integriertes Netz- und Systemmanagement. Addison Wesley 1993

[HeKeNe96] Heilbronner, S.; Keller, A.; Neumair, B.: Integriertes Netz- und Systemmanagement mit modularen Agenten. In: Proceedings of SIWORK'96, Zürich, Switzerland. Mai 1996

[HeNeWi95] Hegering, H.-G.; Neumair, B.; Wies, R.: Integriertes Management verteilter Systeme - Ein Überblick über den State-of-the-Art -. LMU München, Institut für Informatik. Bericht 9503. (Januar 1995)

[Ho96] Höller T.: Entwurf und Realisierung eines CORBA/SNMP Gateways. Diplomarbeit, Technische Universität München, August 1996

[IBMNVPG] IBM NetView for AIX Programmer's Guide, Version 4, (SC31-8164), First Edition, Juli 1995

[IBMNVRG] IBM NetView for AIX Programmer's Reference, Version 4, (SC31-8165), First Edition, Juli 1995

[IBMPrGI95] IBM TMN Products for AIX: General Information, Release 1, (GC31-8016), First Edition, September 1995

[IBMPrIC95] IBM TMN Products for AIX: Installation and Configuration, Release 1, (SC31-8008), First Edition, September 1995

[IBMPrMe95] IBM TMN Products for AIX: Messages, Release 1, (SC31-6200), First Edition, September 1995

[IBMPrDM95] IBM TMN Products for AIX : Developing TMN Management Applications, Release 1, (SC31-8141), First Edition, September 1995

[IBMPrUG95] IBM TMN Products for AIX: TMN Agent User's Guide, Release 1, (SC31-8157), First Edition, September 1995

[IBMSFUG95] IBM NetView TMN Support Facility for AIX: User's Guide, Release 1, (SC31-8017), First Edition, September 1995

[IBMSFPGR95] IBM NetView TMN Support Facility for AIX: Programmer's Guide and Reference, Release 1, (SC31-8018), First Edition, September 1995

[IBMWBPG95] IBM TMN WorkBench for AIX: Programmer's Guide, Release 1, (SC31-8139), First Edition, September 1995

[IBMWBPR95] IBM TMN WorkBench for AIX: Programmer's Reference, Release 1, (SC31-8140), First Edition, September 1995

[IBMWBCB95] IBM TMN WorkBench for AIX: CookBook, Release 1, (SC31-8007), First Edition, September 1995

[IBMWBMC95] IBM TMN WorkBench for AIX: Managed Object/Agent Composer's User's and Programmer's Guide, Release 1, (SC31-8006), First Edition, September 1995

[IIMCIMIBTR] Network Management Forum: Forum 026, Translation of Internet MIBs to OSI/CCITT GDMO MIBs, Issue 1.0, 1993

[IIMCSEC] Network Management Forum: Forum 027, ISO/CCITT to Internet Management Security, Issue 1.0, Oktober 1993

[IIMCPROXY] Network Management Forum: Forum 028, ISO/CCITT to Internet Management Proxy, Issue 1.0, 1993

[IIMCMIB-II] Network Management Forum: Forum 029, Translation of Internet MIB-II (RFC 1213) to ISO/CCITT GDMO MIB, Issue 1.0, Oktober 1993

[IIMCOMIBTR] Network Management Forum: Forum 030, Translation of OSI/CCITT MIBs to Internet MIBs, Issue 1.0, Oktober 1993

[ISO7498-4] Information Technology - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework

[ISO10164-5] Information Technology - Open Systems Interconnection - Systems Management Function - Part 5: Event Report Management Function. IS 10164-5. ISO/IEC. (Juni 1993)

[ISO10164-6] Information Technology - Open Systems Interconnection - Systems Management Function - Part 6: Log Control Function. IS 10164-6. ISO/IEC. (Juni 1991)

[ISO10165-2] Information Technology - Open Systems Interconnection - Management Information Model - Part 2: Definition of Management Information

[ISO10165-4] Information Technology - Open Systems Interconnection - Management Information Model - Part 4: Guidelines for the Definition of Managed Objects

[KaSe93] Kalyanasundaram, P.; Sethi, A.S.: An Application Gateway Design for OSI-Internet Management. In: Proceedings of the second IFIP/IEEE International Symposium on Integrated Network Management III. H.-G. Hegering and Y. Yemini (Editors). North-Holland. IFIP 1993

[Ke96] Keller, A.: Interner Vortrag. Münchner Netz Management Team, Juni 1996

[Mell96] Mellquist, P.E.: SNMP++ - An Open Specification for Object Oriented Network Management Development Using C++. Revision 2.5e. Hewlett Packard Company 1996

[Murr93] Murrill, B.: OMNIPoint: An Implementation Guide to Integrated Networked Information Systems Management. In: Proceedings of the second IFIP/IEEE International Symposium on Integrated Network Management II. H.-G. Hegering and Y. Yemini (Editors). North-Holland. IFIP 1993

[OMNIP93] Network Management Forum : Discovering OMNIPoint. PTR Prentice Hall, 1993

[Perkins92] Perkins, D.T.: User's Guide for SNMP MIB Compiler program, SMIC. SynOptics Communications, Inc., July 1992

[Reilly] Reilly, J.: README to SMIC2GDMO

[Reilly93] Reilly, J.: VTT IIMC Notes - Modification to SMIC „SNMP MIB Compiler“ to produce GDMO MIB Definitons. Technical Research Centre of Finland, Telecommunications Laboratory (VTT/TEL), Mai 1993

[RFC1155] Rose, M.T.; McCloghrie, K.: Structure and Identification of Management Information for TCP/IP-based Internets, 1990

[RFC1157] Case, J.D.; Fedor, M.; Schoffstall, M.L.; Davin, C.: Simple Network Management Protocol (SNMP), Mai 1990

[RFC1213] McCloghrie, K.; Rose, M.T.: Management Information Base for Network Management of TCP/IP-based Internets: MIB-II, 1991

[RFC1902] Case, J.; McCloghrie, K.; Rose, M.; Waldbusser, S.: Structure of Management Information for Version 2 of the Simple Network Management Protovol (SNMPv2), RFC 1902, Januar 1996

[RFC1903] Case, J.; McCloghrie, K.; Rose, M.; Waldbusser, S.: Textual Conventions for Version 2 of the Simple Network Management Protovol (SNMPv2), RFC 1903, Januar 1996

[RFC1904] Case, J.; McCloghrie, K.; Rose, M.; Waldbusser, S.: Conformance Statements for Version 2 of the Simple Network Management Protovol (SNMPv2), RFC 1904, Januar 1996

[RFC1908] Case, J.; McCloghrie, K.; Rose, M.; Waldbusser, S.: Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework, RFC 1908, Januar 1996

[Stal93] Stallings, W.: SNMP, SNMPv2 and CMIP - The Practical Guide to Network-Management Standards. Addision Wesley 1993

[Shrews94] Shrewsbury, J.K.: TMN in a Nutshell. Version 1.0 November 1994