INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwurf und Realisierung eines CORBA/SNMP Gateways

Bearbeiter	: Thomas Höller
Aufgabensteller	: Prof. Dr. Heinz–Gerd Hegering
Betreuer	: Alexander Keller
	Dr. Bernhard Neumair

Abgabetermin : 15. August 1996

Zusammenfassung

Der Einsatz der Common Object Request Broker Architecture (CORBA) als Basis von Managementsystemen ist derzeit Thema vieler Forschungsaktivitäten. Viele Standardisierungsaktivitäten und die Aussagen vieler Managementsoftware-Hersteller, demnächst mit ihren Produkten CORBA-konform zu sein, deuten darauf hin, daß der Ansatz der Object Management Group (OMG) für das Management von Systemen und Anwendungen möglicherweise eine wichtige Rolle spielen wird. Dennoch wird sich CORBA gegenüber dem OSI- und Internet-Management nicht vollständig durchsetzen können. So hat sich beispielsweise das Internet-Management aufgrund seiner einfachen Managementlösungen (und der Möglichkeit einer entsprechend schnellen Realisierung dieser Lösungen) zu einem De-facto-Standard auf dem Gebiet der Datennetze entwickelt. In Rechnernetzen werden also noch einige Zeit mehrere Managementsysteme nebeneinander im Einsatz sein. In Hinsicht auf ein einheitliches Management ist die Integration und die Interoperabilität von Managementarchitekturen eine wichtige Voraussetzung. Es sind Übergänge erforderlich, die es Managementanwendungen ermöglichen, auf Managementinformation in anderen Systemen zuzugreifen. Beim Einsatz von neuen Managementarchitekturen wie CORBA kann auf diese Weise eine bestehende Informationsbasis weiterhin verwendet werden.

In dieser Diplomarbeit wurde ein Konzept für ein CORBA/SNMP-Gateway entworfen. Ein CORBA/SNMP-Gateway ermöglicht es, SNMP-Ressourcen von einer CORBA-Umgebung aus zu verwalten. Es liegt zwischen einer CORBA-Managementanwendung und den Netzressourcen, auf der Grenze zwischen den beiden Protokollwelten. Es integriert die Internet-Architektur in die CORBA-Welt, indem es SNMP-Managementobjekte auf CORBA-Objekte abbildet einem CORBA-Manager zugänglich macht. Weder die CORBA-Managementanwendung noch die SNMP-Agenten müssen dafür verändert werden. Die entgegengesetzte Richtung, also der Entwurf eines Gateways derart, daß ein SNMP-Manager CORBA-Objekte verwalten kann, wurde dabei nicht behandelt. Der Ansatz der Internet-Architektur ist konzeptionell zu schwach, um die viel mächtigere CORBA-Umgebung zu modellieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die Fähigkeit zur effektiven und konsequenten Kommunikation wird heutzutage von immer mehr Unternehmen als fast überlebenswichtige Voraussetzung für Erfolg gewertet. Die Aufgabe des Netz- und Systemmanagements ist es, dafür zu sorgen, daß Kommunikationssysteme, die entsprechende Kommunikationsdienste zur Verfügung stellen, möglichst effizient und störungsfrei genutzt werden können. Die wachsende Heterogenität in den Rechnernetzen, speziell in verteilten Systemen, erfordert dazu oft komplexe Vorkehrungen.

Bestrebungen, das Management zu standardisieren, führten zu einer Vielzahl unterschiedlicher Managementarchitekturen. Sehr verbreitet sind das OSI-Management von der ISO und der ITU (International Telekommunikation Union) auf dem Sektor der Telekommunikationsnetze und das Internet-Management von der IETF (Internet Engineering Task Force) im Bereich der privaten Datennetze. Beispiele für andere Managementarchitekturen sind das Telecommunikation Management Network (TMN von ITU), das LAN/MAN-Management (IEEE), Distributed Management Environment (DME von Open Software Foundation, OSF), das Desktop Management Interface (DMI von der Desktop Management Task Force, DMTF) und Omnipoints (Network Management Forum, NMF).

Mit der Object Management Architecture (OMA) liefert die Object Management Group (OMG) einen weiteren Ansatz. Zentrales Element der OMA ist ein Object Request Broker (ORB), der Dienste für eine synchrone Kommunikation zwischen Objekten zur Verfügung stellt. Der ORB wurde von der OMG inzwischen mit der *Common Object Request Broker Architecture* (CORBA 2.0) standardisiert ([OMG95a]). Alle namhaften Hersteller von Managementsoftware wollen demnächst mit ihren Produkten CORBA-konform sein. Im Bereich Systemmanagement ist also die Tendenz feststellbar, CORBA als Basis von Managementsystemen einzusetzen.

1.1 Motivation

Von den drei bedeutendsten Architekturen OSI, Internet und CORBA hat jede aufgrund der Zielvorgabe bei der Entwicklung und wegen der gegebenen Managementanforderungen im geplanten Einsatzgebiet (Telekommunikationsnetz, Datennetz) unterschiedliche Vor- und Nachteile bezüglich verschiedener Managementszenarien. So scheint CORBA beispielsweise wegen des objektorientierten Ansatzes besonders für System- und Anwendungsmanagement geeignet. Das Internet-Management wird sich aber mit seinen einfachen und implementierungsfreundlichen Managementlösungen gegenüber CORBA und dem OSI-Management noch länger behaupten können. Außerdem ist eine Umstellung auf ein neues (z.B. CORBA-basiertes) Managementsystem aufgrund der hohen Kosten ohnehin nur langfristig finanziell tragbar. In Rechnernetzen werden aus diesen Gründen noch einige Zeit mehrere Managementsysteme parallel (in verschiedenen Teilnetzen) im Einsatz sein. Die Managementakti-

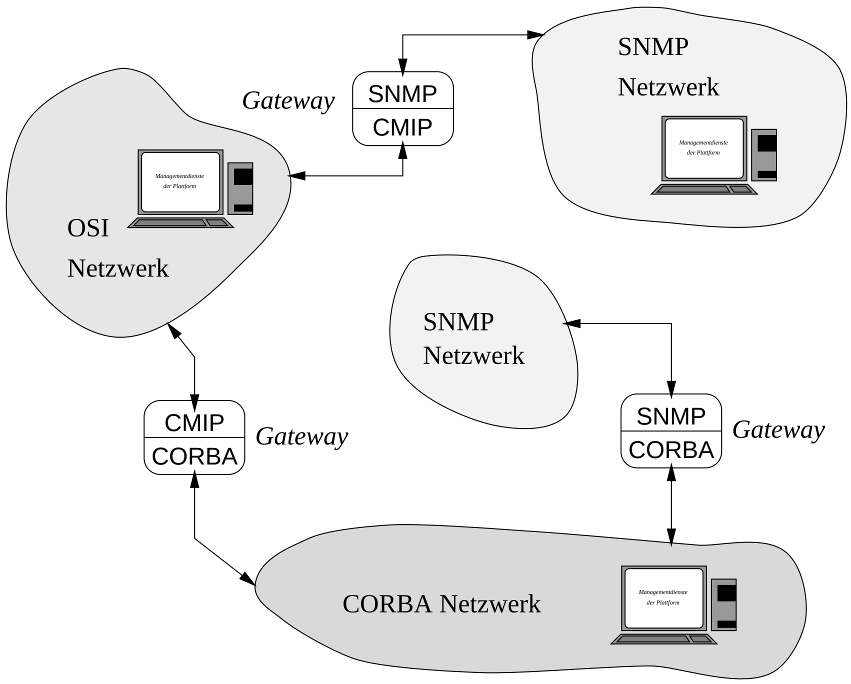

Abbildung 1.1: Interoperabilität von Managementarchitekturen

vitäten in den einzelnen Teilnetzen müssen koordiniert werden, damit die Kommunikationsfähigkeit im Gesamtnetz gewährt bleibt. In Hinsicht auf ein einheitliches Management zur Lösung dieser Problematik ist die Integration und die Interoperabilität von Managementarchitekturen eine wichtige Voraussetzung. Es müssen Übergänge geschaffen werden, die es Managementanwendungen ermöglichen, auf Managementinformation in anderen Systemen — idealerweise transparent — zuzugreifen. Dieser Zugriff wird beispielsweise mittels sogenannter Management-Gateways realisiert. Abbildung 1.1 zeigt den Einsatz verschiedener Gateways für das integrierte Management von OSI-, OMA- und Internet-Ressourcen in den jeweiligen Teilnetzen.

1.2 Aufgabenstellung

Durch die Einbettung von einer Managementarchitektur in eine andere, können die Vorteile der integrierenden Architektur auch für die Überwachung und Steuerung der Managementobjekte in der integrierten Architektur genutzt werden, ohne daß jene in irgendeiner Weise geändert werden müssen. Damit lassen sich neue, moderne — d.h. für das komplexe Management heutiger Rechnernetze besser geeignete — Managementsysteme mit vertretbarem finanziellen und technischen Aufwand einführen. Als Konsequenz ist es für den Erfolg eines neuen Architekturkonzepts ein wichtiges Kriterium, inwieweit alte Systeme (sogenannte Legacy-Systeme) eingegliedert werden können.

Diese Frage stellt sich insbesondere in Bezug auf CORBA, das, wie Eingangs dieses Kapitels erwähnt, immer mehr an Bedeutung auch für das Management gewinnt, und dem Internet-Management, das sich im Laufe der vergangenen Jahre als Defacto-Standard auf dem Gebiet des Netz- und Systemmanagements etabliert hat. Im konkreten Fall sollen Ressourcen, die durch Internet-Managementobjekte repräsentiert werden, von einer CORBA-Managmentapplikation weiterhin verwaltet werden können (Abb. 1.2). Ziel dieser Diplomarbeit ist es daher, ein geeignetes Konzept für ein CORBA/SNMP-Gateway zu entwerfen und nach Möglichkeit prototypisch zu implementieren.

Das Gateway soll einer CORBA-Managementanwendung folgendes ermöglichen:

- Lesenden sowie schreibenden Zugriff auf Managementinformationen, die von SNMP-Agenten bereitgestellt werden.
  - Die SNMP-Agenten dürfen dafür nicht verändert werden.
  - Der Übergang in die SNMP-Protokollwelt soll für die Managementanwendung transparent sein.
  - SNMP und SNMPv2 sollen gleichermaßen unterstützt werden.
- Empfangen von SNMP-Traps aus der SNMP-Umgebung.
  - Die Managementanwendung soll dazu weiterhin den CORBA-Event-Service (vgl. 2.3.1) verwenden können.
- Senden und Empfangen von SNMP-Inform-PDUs (SNMPv2) und damit die Kommunikation mit SNMP-Managern.

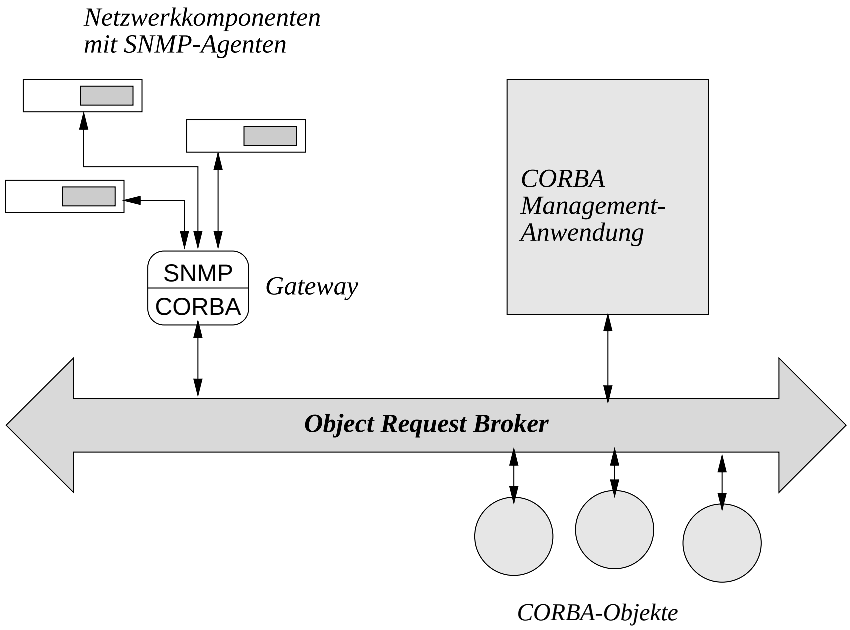

Abbildung 1.2: Integration des Internet-Managements

Die entgegengesetzte Richtung, also die Erweiterung des Gateways, sodaß ein SNMP-Manager ebenso CORBA-Objekte verwalten kann, entfällt. Der Ansatz der Internet-Architektur ist konzeptionell zu schwach, um die viel mächtigere CORBA-Umgebung zu modellieren.

1.3 Gliederung der Diplomarbeit

Die Diplomarbeit gliedert sich in 6 Kapitel. Kapitel 2 beschreibt einführend den Stand der Technik im Bereich des Netz- und Systemmanagements. Im Kapitel 3 werden die Schritte beschrieben, die bei einem Übergang von Managementsystemen im allgemeinen und von SNMP nach CORBA im speziellen vorgenommen werden müssen. Der Schwerpunkt liegt dabei auf der Beschreibung der SNMP-Managementinformation in CORBA-konformer IDL-Notation (s. 3.1.1).

Auf der Grundlage der in Kapitel 3 erarbeiteten Voraussetzungen werden im 4. Kapitel zunächst die Anforderungen an das Gateway festgestellt. Zur Erfüllung dieser Anforderungen werden verschiedene Konzepte betrachtet und bewertet. Als Überleitung zur Implementierung des Konzepts wird in Kapitel 5 die Entwicklungsumgebung SOM/DSOM vorgestellt, und beschrieben, wie sie zur Realisierung des Gateways prinzipiell eingesetzt wird. Die Ergebnisse der Arbeit werden schließlich in Kapitel 6 zusammengefaßt.

Kapitel 2

Stand der Technik

Die Managementanforderungen bei verteilten Systemen reichen inzwischen weit über die Aufgaben des Netzmanagements hinaus. Wegen der breiten Produktpalette, die den weiteren Einsatz von SNMP bzw. CMIP in diesem Bereich attraktiv macht, liegt es nahe, das Internet- bzw. OSI-Management für die neuen Managementszenarien (vor allem System- und Anwendungsmanagement) zu erweitern. Beispielsweise liegt mit dem *Simple Management Protokoll Framework* schon eine neue Version des Internet-Management-Frameworks vor, das auch für diese neuen Szenarien geeignet sein soll. Aufgrund der flexibleren OSI-Managementkonzepte (darunter ein objektorientiertes Informationsmodell, das Domänenkonzept und die *System Management Functions*, eine Sammlung managementspezifischer Basisdienste für Managementanwendungen) ist es in dieser Hinsicht jedoch wahrscheinlicher, daß sich das OSI-Management durchsetzt. Für die Entwicklung einer integrierten Managementlösung ist man allerdings der Meinung, daß mehr Wert auf ein Neben- und Miteinander als auf eine Konfrontation der beiden Ansätze gelegt werden muß.

2.1 Methoden der Integration von Architekturen

2.1.1 Multiarchitekturelle Plattformen

Eine integrierte Managementplattform bietet eine Infrastruktur, in die die verschiedensten Managementanwendungen (Managementdienste wie Ereignismanagement, Leistungsmonitor, Konfigurationsmanagement, etc.) eingebettet werden können. Außerdem können mehrere Managementarchitekturen angebunden werden. Abbildung 2.1(a) zeigt eine Management-Plattform, die sich als Basiswerkzeug für OMA- und Internet-Management eignet. Beispiele für kommerzielle Plattformen sind HP OpenView, IBM NetView 6000, Tivoli TME und Cabletron Spectrum. Ein Konzept zur

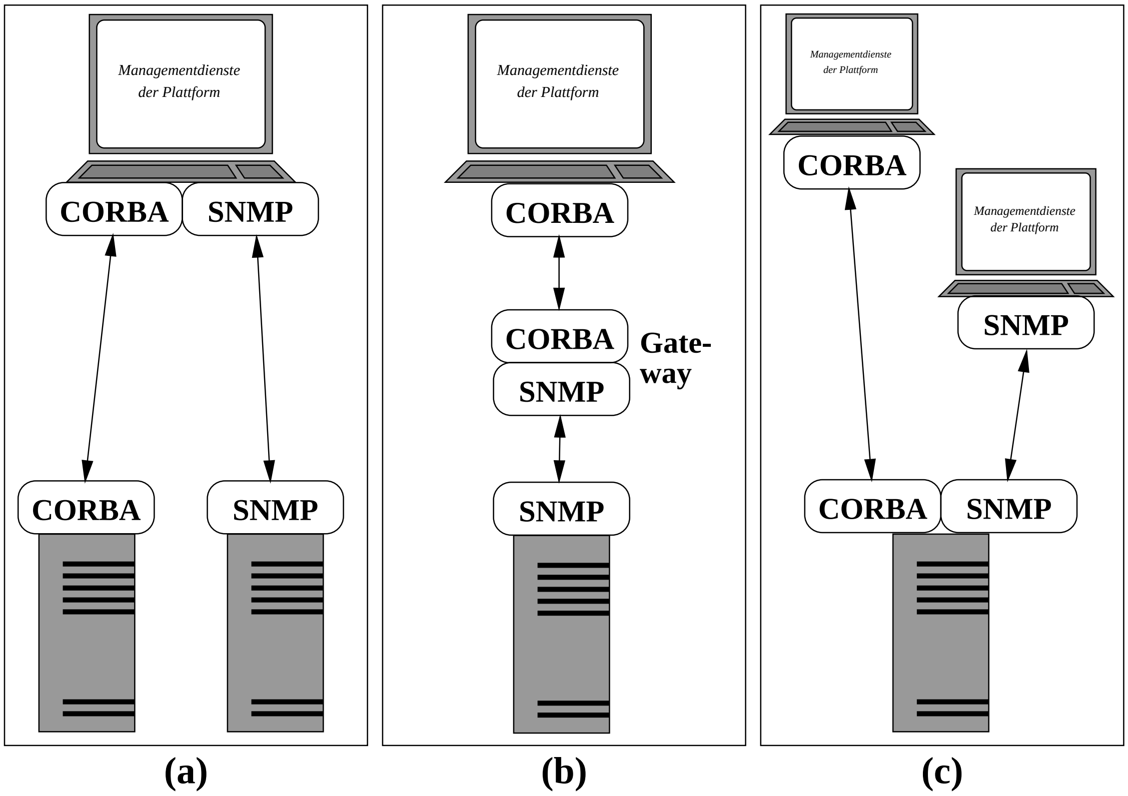

Abbildung 2.1: Integrationstechniken

Anbindung einer CORBA-Umgebung an die SNMP/CMOT[1]-basierte Netzmanagementplattform NetView wird in [Vog96] vorgestellt.

2.1.2 Management-Gateways

Ein *Management-Gateway* (auch *application gateway* [ACH93], [KS93] oder *Proxy-Agent* [HA93] genannt) sitzt auf der Grenze zwischen verschiedenen Protokollwelten und übernimmt die notwendige Abbildung der architekturspezifischen Unterschiede. Ein Management-Gateway ist in Abbildung 2.1(b) dargestellt. Der Vorteil eines Management-Gateways ist, daß weder Managementanwendung noch Agent verändert werden müssen. Dieser Ansatz eignet sich deshalb besonders für die Migration/Einbettung von Managementarchitekturen (vgl. auch [KS93]). Bei der Integration des OSI-Managements in eine CORBA-Umgebung z.B., wird mit einem CORBA/CMIP-Gateway dieser Ansatz verfolgt ([Sou94]). In dieser Diplomarbeit soll entsprechend ein CORBA/SNMP-Management-Gateway zur Integration des Internet-Managements in CORBA entworfen werden.

[1]CMIP (Common Management Information Protocol) over TCP/IP

Gateways zwischen CORBA- und SNMP-basierten Managementarchitekturen

Zwischen CORBA und der Internet-Managementarchitektur sind die in Tabelle 2.1 festgehaltenen Gateways möglich. Basieren sowohl Manager als auch Agent auf derselben Architektur, so ist kein Gateway notwendig. Soll ein SNMP-Manager CORBA-Agenten verwalten, so wird ein SNMP/CORBA-Gateway benötigt; umgekehrt verwendet ein CORBA-Manager für SNMP-Agenten ein CORBA/SNMP-Gateway. Diese beiden unidirektionalen[2] Gateways können in einem bidirektionalen Gateway vereint werden, das dann von CORBA- bzw. SNMP-Managern gleichermaßen benutzt werden kann. Allerdings hat es kaum einen Sinn, mittels eines SNMP-Managers ein viel mächtigeres und komplexeres CORBA-System zu verwalten, weshalb ein SNMP/CORBA-Gateway allgemein nicht in Betracht gezogen — und folglich in dieser Diplomarbeit nicht behandelt wird.

		Manager	
		CORBA	SNMP
Agent	CORBA	—	SNMP/CORBA Gateway
	SNMP	CORBA/SNMP Gateway	—

Tabelle 2.1: Gateway-Arten

2.1.3 Multiarchitekturelle Agenten

Die dritte Möglichkeit, Managementobjekte einer fremden Architektur zu verwalten, ist, die entsprechenden Agenten um Schnittstellen zu erweitern, sodaß die Managementanwendungen unterschiedlicher Architekturen auf die Informationen des Agenten zugreifen können (Abb. 2.1(c)). In [MBL93] wird beispielsweise ein protokollunabhängiger Agent besprochen, auf den mit SNMP und CMIP zugegriffen werden kann. Dieser Ansatz ist allendings nur bei Ressourcen realisierbar, die den dazu notwendigen Speicher und die nötige Performance besitzen (z.B. Workstations). Kleine Netzkomponenten erfüllen diese Voraussetzungen in der Regel nicht.

[2] im Sinne der Beziehung verwaltendes/verwaltetes System, die das Gateway impliziert

2.2 Integrationsaktivitäten bei Managementsoftware

Mit dem Ziel eines integrierten Managements versuchen verschiedene Gremien, Übergänge zwischen den unterschiedlichen Management- und Plattformarchitekturen zu definieren, um Koexistenz und Kooperation zu ermöglichen. Die Managementarchitekturen der ISO und der IAB finden dabei am meisten Beachtung.

- Im Bereich Systemmanagement versucht die Herstellervereinigung *X/Open* mit der Programmierschnittstelle *X/Open Management Protokoll* (XMP) die Portabilität von herstellerspezifischen CMIP- und SNMP-Anwendungen zu verbessern. Die XMP-Schnittstelle vereinheitlicht den Zugriff auf die beiden Protokolle, wodurch die Implementierung von Anwendungen vereinfacht wird. Die Unterschiede der Protokolle und der dazugehörigen Informationsmodelle werden allerdings kaum vor den Managementanwendungen verdeckt.

- Einen ähnlichen Weg schlägt die Organisation *Desktop Management Task Force* (DMTF) ein. Sie möchte eine vorhandene Managementarchitektur für das Management von Arbeitsplatzrechnern verbessern und erweitern, indem ein einheitlicher konsistenter Zugriff auf die zu verwaltende Ressourcen geschaffen wird. Das *Desktop Management Interface* (DMI) definiert dazu einen Broker zwischen Hard- und Softwarekomponenten und (Management-)Protokollimplementierungen.

- Mit dem *Distributed Management Environment* (DME) will die *Open Software Foundation* (OSF) eine Infrastruktur für ein integriertes Management anbieten. Derzeit wird jedoch nicht versucht, eine neue Managementarchitektur zu etablieren, sondern vorhandene zu integrieren. Schwerpunkt ist wiederum der Versuch, OSI- und Internet-Management zusammenzuführen.

- Das *Network Management Forum* (NMF) konzentriert sich derzeit auf das Ziel, die Koexistenz des ISO-OSI- und IAB/IETF-Ansatzes zu erreichen. Die *ISO-Internet-Management Coexistence* (IIMC) beinhaltet neben Protokollabbildungen und Abbildungen zwischen den Informationsmodellen die Festlegung von Profilen. Diese Profile sollen Implementierungen vereinfachen und die Interoperabilität fördern.

Neben der Tendenz, bestehende Managementarchitekturen den neuen Managementanforderungen anzupassen, werden andererseits neue Architekturen wie z.B. OMG CORBA definiert, wobei versucht wird, von vornherein möglichst viele Kriterien zu berücksichtigen. Wie im vorherigen Kapitel begründet, ist es für den Erfolg dieser neuen Ansätze aber wichtig, daß die verbreiteten Managementarchitekturen in die neue Umgebung eingebettet werden (können). Mit dieser Problematik befaßt sich die *X/Open Joint Inter-Domain Management Group* (s.u.).

2.3 Die Common Object Request Broker Architecture

Client/Server-Versorgungsstrukturen, aufgesetzt auf verteilte Rechnersysteme, gestatten eine flexible und effiziente Anordnung der Bausteine von verteilten Anwendungen. Problem ist dabei die Heterogenität solcher Computersysteme, die Aufrufe über maschinenspezifische Rechnergrenzen hinweg nötig macht. Gängige Verfahren dafür sind unter dem Schlagwort *RPC (Remote Procedure Call)* bekannt und wurden in mehreren Standards (ONC-RPC, DCE-RPC, ROSE) spezifiziert. Auch wenn dadurch die Entwicklung von verteilten Software-Systemen einfacher ist, zeigt sich, daß mit der Forderung nach Interoperabilität zwischen solchen Anwendungen die RPC-Mechanismen allein nicht mehr ausreichen. Es fehlt eine komplette Architektur für die Zusammenarbeit verschiedener Applikationen. Dies und die monolithische Struktur und schlechte Kombinierbarkeit heutiger SW-Systeme waren 1989 Anlaß zur Gründung der Object Management Group (OMG), einem herstellerübergreifenden Konsortium mit inzwischen mehr als 600 Mitgliedern. Ziel war die Interoperabilität, Wiederverwendbarkeit und Portabilität von Software.

Im November 1992 veröffentlichte die OMG mit dem *Object Management Architecture Guide* ([OMG92]) ein Rahmenwerk für die Kooperation von heterogenen verteilten Systemen. Zentrales Element dieser Architektur ist ein Object Request Broker (ORB), standardisiert in der *Common Object Request Broker Architecture* (CORBA 2.0, [OMG95a]). Er stellt Mechanismen bereit, die die Kooperation von Anwendungen in einer verteilten Umgebung ermöglichen. Die Kooperation und Kommunikation beruht dabei auf dem Aufruf von Operationen auf Objekten: Der ORB nimmt den Aufruf (Request) eines Aufrufers (Client) entgegen, sucht eine geeignete Instanz (Server) des gewünschten Typs, ruft die entsprechende Operation auf und reicht eventuelle Ergebnisse an den Aufrufer zurück. Der ORB bietet dazu eine Reihe von Aufrufschnittstellen und Diensten an (Abb. 2.2):

- *Statische Aufrufschnittstelle*: Für jede Operation eines Objektes wird aus der IDL[3]-Beschreibung zur Compile-Zeit eine Stub-Routine erzeugt, die in Client-Programme eingebunden werden kann. Damit kann das Client-Programm die entsprechende Operation verwenden.

- *Dynamische Aufrufschnittstelle*: Diese Aufrufschnittstelle ist für alle Implementierungen gleich. Sie ermöglicht es Client-Programmen, Serverobjekte, Prozeduren und ihre Parameter zur Laufzeit zu benennen. Für den Methodenaufruf muß ein Client die Objektreferenz[4] des Zielobjektes, die aufzurufende

[3]Interface Definition Language, eine programmiersprachenunabhängige Notation zur Definition von Objektschnittstellen.

[4]Eine Objektreferenz ist ein eindeutiger Identifikator eines Objektes. Der ORB weist jedem Objekt mindestens eine Objektreferenz zu, anhand der er das Objekt lokalisieren kann.

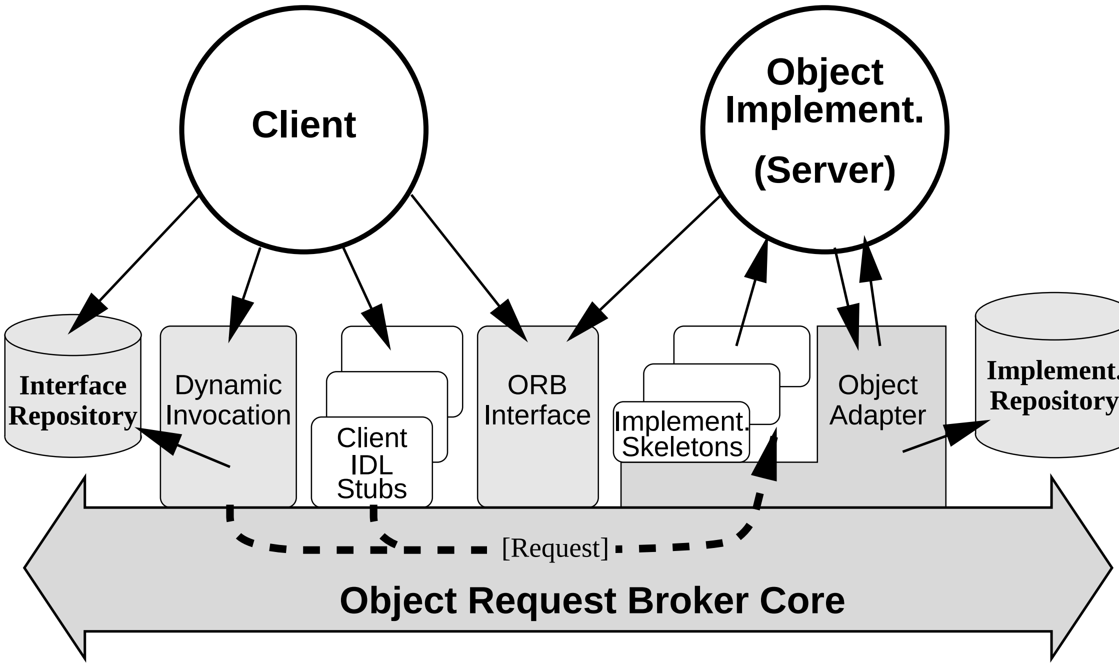

Abbildung 2.2: Die Struktur von CORBA

Methode sowie dazugehörige Parameter angeben.

- *Interface Repository*: Im Interface Repository sind die IDL-Beschreibungen aller Objektklassen abgelegt. Die Informationen können beispielsweise bei der dynamischen Zusammensetzung eines Requests abgerufen werden.

- *ORB Schnittstelle*: An dieser Schnittstelle bietet der ORB Dienste an, die für aufrufende und aufgerufene Objekte gleichermaßen von Nutzen sein können. Unter anderem geschieht der Zugriff auf das Interface Repository über diese Schnittstelle.

- *Object Adapter*: Der Object Adapter gibt Aufrufe über die IDL-Skeletons (Stub-Routine für aufgerufene Objektmethoden) weiter. Außerdem stellt er generelle Dienste für die Objekt-Implementierungen zur Verfügung, z.B. zur Generierung von Objektreferenzen oder zur Registrierung neuer Implementierungen.

- *Implementation Repository*: Im Implementierungs-Verzeichnis sind Informationen über die aktuellen Implementierungen der dem ORB bekannten Objekte (Lokalisation, Sicherheitsinformation, etc.) gespeichert.

Aufgrund zahlreicher Literatur ([OMG95a], verschiedene Artikel: [Rö93], [Gei92] ...) wird hier nicht näher auf CORBA eingegangen.

Die Basisfunktionalität des ORBs wird durch Dienste spezieller Objekte erweitert. Diese Dienste können von allen Objekten benützt werden (z.B. bei der Erzeugung von Objekten). Der entsprechende Standard, die *Common Object Services Specification* ([OMG96]), spezifiziert einen *LifeCycle Service, Persistence Object Service, Naming Service, Event Service, Concurrency Controll Service, Transaction Service, Relationship Service, Query Service* und einen *Externalization Service*.

2.3.1 Die CORBA Object Services

Der CORBA LifeCycle Service

Der *LifeCycle Service* ([OMG96]) definiert Dienste und Konventionen, um Objekte in einer verteilten Umgebung zu erzeugen, zu löschen, zu kopieren und zu verschieben. Das LifeCycle-Modul besteht aus

- dem interface FactoryFinder,
- dem interface LifeCycleObject und
- dem interface GenericFactory:

Das FactoryFinder-Interface unterstützt die Suche nach Factories. Eine Factory ist ein Objekt, mit dem ein Objekt mit einer bestimmten Schnittstelle erzeugt werden kann. Die Suche erfolgt nach dem Namen einer Factory. Über die Schnittstellen oder Implementierungen, die die zurückgegebenen Factories und die Objekte, die von diesen erzeugt werden, unterstützen, wird nichts ausgesagt.

Das GenericFactory-Interface definiert eine generische Operation create_object() und stellt damit einen Dienst zur Erzeugung von Objekten zur Verfügung. Die Aufgabe der GenericFactory ist, Aufrufe von create_object() an implementierungsspezifische Factories, welche den angegebenen Kriterien genügen, weiterzuleiten.

Das LifeCycleObject-Interface schließlich definiert Methoden, um Objekte in einer verteilten Umgebung zu kopieren, zu verschieben und zu löschen. Objekte, welche den LifeCycle Service verwenden, unterstützen diese Schnittstelle.

Der CORBA Naming Service

Der *Naming Service* ([OMG96]) von CORBA bietet die Möglichkeit, Namen zu Objekten in sogenannten *Name Bindings* zuzuordnen. Ein derartiges Name Binding wird immer relativ zu einem *Naming Context* definiert. Ein Naming Context ist ein

Objekt mit einer Menge von Name Bindings, in dem alle Namen eindeutig sind. Verschiedene Namen können zu einem Objekt im selben und/oder in einem anderen Kontext gebunden werden.

Da auch ein Naming Context ein Objekt ist, kann diesem ebenfalls ein Name in einem Naming Context zugeordnet werden. Es entsteht ein *Naming Graph*, ein gerichteter Graph, in dem die Knoten Kontexte sind. Damit können zusammengesetzte Namen (*Compound Names*) für Objekte erzeugt werden. Jeder Compound Name besteht aus Komponenten, die alle bis auf die letzte Komponente Namenskontexte bezeichnen. die letzte Komponente bezeichnet das zum zusammengesetzten Namen gebundene Objekt. Abbildung 2.3 zeigt ein Beispiel eines möglichen Naming Graphs.

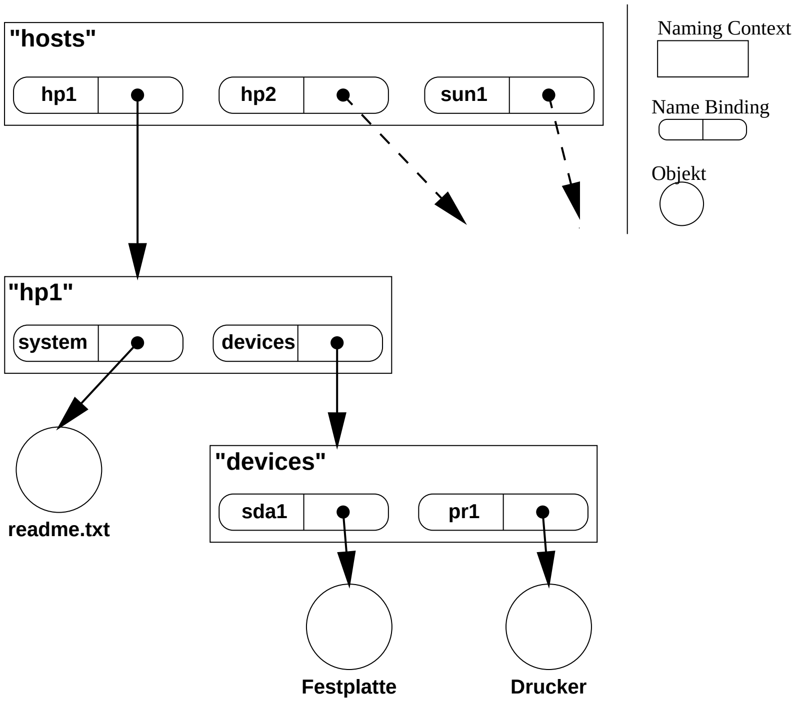

Abbildung 2.3: Ein Naming Graph

Die „Wurzel" des dort gezeigten Naming Graphs ist der Naming Context „Hosts". Er enthält drei Name Bindings. Der Name „hp1" bezeichnet dabei einen weiteren Naming Context. In diesem befinden sich zwei Name Bindings: der Name „system" wird einem Objekt (etwa eine Datei) zugeordnet, der Name „devices" bezeichnet einen Naming Context (mit den Namen bzw. den Name Bindings der I/O-Geräte des Hosts „hp1"). Im Beispiel ist der zusammengesetzte Namen des Objektes „Drucker" also: hosts.hp1.devices.pr1. Mit diesem Namen ist dieses Objekt eindeutig identifizierbar.

Der CORBA-Event-Service

Im *Event Service* ([OMG96]) werden die Regeln für das Versenden und Empfangen von asynchronen Ereignismeldungen zwischen Objekten festgelegt. Objekte, welche Events erzeugen und verschicken, werden *Supplier*, Objekte, welche diese Events empfangen und verarbeiten, *Consumer* genannt. Für den Austausch der Ereignismeldungen zwischen Supplier und Consumer werden übliche CORBA-Requests verwendet. Dabei stehen zwei Möglichkeiten zur Verfügung:

- Im sogenannten *Push*-Modell ergreift der *PushSupplier* die Initiative. Er übermittelt einem *PushConsumer* eine Ereignismeldung, indem er die Methode push des *PushConsumers* aufruft.

- Beim *Pull*-Modell kann der *PullConsumer* vom *PullSupplier* Ereignismeldungen *pollen*, indem er die Methode pull des *PullSuppliers* aufruft. Die Methode try_pull des *PullSuppliers* kann verwendet werden, um zu prüfen, ob Ereignismeldungen beim *PullSupplier* vorliegen.

In beiden Modellen beruht die Event-Kommunikation zwischen Objekten also auf Aufrufen spezieller push- und pull-Methoden. Bei der *generischen* Kommunikation von Ereignismeldungen haben diese Methoden jeweils nur einen Parameter vom Datentyp any. Im Gegensatz dazu können bei der *typisierten* Kommunikation mit bestimmten Einschränkungen beliebige „anwendungsspezifische“ Parameter verwendet werden. Bei den push-Methoden eines *TypedConsumers* sind keine Rückgabewerte und nur in-Parameter erlaubt. Bei den pull- bzw. try_pull-Methoden des dazugehörigen *TypedSuppliers* (Pull-Modell) sind entsprechend nur out-Parameter zulässig. Vorteil bei der typisierten Event-Kommunikation ist die Möglichkeit, verschiedene Arten von Ereignismeldungen zu definieren, zu unterscheiden und zu selektieren. Dadurch eignet sich diese Methode besonders für das Netz-, System- und Anwendungsmanagement.

Für die Kommunikation von beliebig vielen Suppliern mit beliebig vielen Consumern werden sogenannte *Event Channels* eingesetzt. Supplier können Ereignismeldungen statt direkt an einen Consumer an einen Event Channel schicken. Jedem Consumer-Objekt, das sich bei diesem Event Channel registriert hat, wird die Ereignismeldung weitergereicht. Der Event Channel ruft hierzu wiederum Methoden der Consumer auf (*Push*-Modell). Der Event Channel selbst ist also sowohl Supplier als auch Consumer von Events.

Der Event Channel wird inkrementell erzeugt, d. h. bei der Erzeugung sind noch keine Supplier/Consumer beim Event Channel registriert. Diese können sich nach und nach beim Event Channel anmelden. Die Anmeldung erfolgt durch Austausch von Objektreferenzen. Ein Consumer gibt dem Event Channel bei der Registrierung die Schnittstelle bekannt, die der Event Channel für das Weiterreichen der Events verwenden kann. Damit stehen die Events, die ein Consumer empfangen will, implizit fest. Analog gehen die Supplier von Events vor, um dem Event Channel zu sagen, welche Events sie ihm schicken wollen. Wie die Anmeldung im Detail erfolgt, kann in [OMG96] nachgelesen werden.

Der CORBA Concurrency Control Service

Der *Concurrency Control Service* ([OMG96]) enthält Mechanismen, um den gleichzeitigen und konkurrierenden Zugriff mehrerer Objekte auf ein Objekt (z. B. auf ein gemeinsames Betriebsmittel) zu kontrollieren. Es werden Verfahren unterstützt, um Deadlocks zu verhindern, zu entdecken und zu beheben.

Der CORBA Externalization Service

Der *Externalization Service* ([OMG96]) ermöglicht, Zustände von Objekten in einen Datenstrom umzuwandeln und umgekehrt. Der Zustand eines Objektes entspricht den Werten seiner Attribute. Auf diese Weise kann ein Objekt beispielsweise über ein Netz verschickt oder sein Zustand ausgedruckt werden.

Der CORBA Relationship Service

Der *Relationship Service* ([OMG96]) dient der Verwaltung von Beziehungen (z. B. ownership, containmentship, ...) zwischen Objekten. Einheiten *Entities, Roles* und Relationen können explizit definiert werden. Es stehen Mechanismen zur Verfügung, mit denen die Konsistenz von in Relation stehenden Objekten erhalten werden kann (z. B. beim Löschen oder Verschieben eines Objektes). Objekte müssen keine bestimmte Schnittstelle unterstützen, um im Relationship Service verwendet werden zu können. Es ist zu jeder Zeit möglich, Objekte in Relation zu anderen Objekten zu setzten.

Der CORBA Persistence Object Service

Der ORB bietet Funktionen, mit denen Objektreferenzen persistent gemacht werden können ([OMG95a]). Der Zustand der entsprechenden Objekte wird dadurch aber nicht gesichert. Schnittstellen des *Persistence Object Service* ([OMG96]) können von Objekten verwendet werden, um den gesamten oder Teile ihres dynamischen Zustandes persistent zu machen. Aus dem persistenten Zustand (etwa in einer Datei) kann ein Objekt seinen dynamischen Zustand wieder herstellen.

Der CORBA Transaction Service

Der *Transaction Service* ([OMG96]) gewährt die Möglichkeit, eine oder mehrere Operationen auf einem oder mehreren Objekten in einer Transaktion (mit den ACID-Eigenschaften) zusammenzufassen.

Der CORBA Query Service

Der *Query Service* ([OMG96]) definiert Operationen auf Gruppen von Objekten mit gleichen Prädikaten, Attributen oder Properties (s. unten). Er bietet außerdem ein Framework, um derartige Operationen (Queries) zu verwalten.

Der CORBA Property Service

Die *Properties* eines Objekten sind äquivalent zu den Attributen des Objektes, werden aber zur Laufzeit diesem Objekt zugewiesen. Der *Property Service* ([OMG95c] stellt alle in diesem Zusammenhang notwendigen Funktionen (Definieren, Aufzählen, Zuweisen von Properties ...) zur Verfügung.

Der CORBA Security Service

Der sehr mächtige *Security Service* ([OMG95b]) unterstützt wichtige Sicherheitsmechanismen, beispielsweise für die Identifizierung, Authentifizierung und Authorisierung von aufrufenden Objekten. Der Standard für den Security Service steht erst seit Juli 1996 zur Verfügung. Er konnte aus zeitlichen Gründen in dieser Diplomarbeit nicht mehr tiefergehend berücksichtigt werden.

2.3.2 Vorteile durch den Einsatz von CORBA

Vorteile durch den Einsatz von CORBA als Basis für Managementsysteme sind unter anderem:

- Die OMA ist prinzipiell für alle verteilten Anwendungen gedacht. Managementapplikationen können also als verteilte Anwendungen ebenso auf der CORBA-Infrastruktur aufbauen.
- Die symmetrische Kommunikationsform läßt eine flexible wechselseitige Auftragsbeziehung zu. Funktionalität von Managementobjekten kann beliebig verlagert werden.

- CORBA-konforme Managementanwendungen lassen sich beliebig wiederverwenden, erweitern und mit anderen Anwendungen kombinieren. Erklärungen vieler Herstellerfirmen versprechen in absehbarer Zeit ein breites Angebot an CORBA-konformen Softwareprodukten.
- OMA-Basisdienste zur Objektverwaltung können von Managementanwendungen verwendet werden. Spezielle Facilities für das Systemmanagement werden derzeit den CORBA-Facilities hinzugefügt.

Trotz dieser Vorteile ist es jedoch nicht absehbar, wie weit sich das OMG-Modell für das Management durchsetzen wird. Eine Vielzahl an Forschungsaktivitäten in Bezug auf CORBA lassen auf gute Chancen für den OMG-Ansatz schließen, doch müssen erst Erfahrungen im praktischen Einsatz von CORBA als Basis von Managementsystemen gesammelt werden, die letztendlich die Tragfähigkeit dieses Konzepts bestätigen.

2.3.3 Forschungsaktivitäten mit Bezug auf CORBA

Die X/Open Systems Management Working Group

Die *X/Open System Management Working Group (XoTGsysMan)* beschäftigt sich mit der Spezifikation von allgemeinen Diensten für das Systemmanagement. Diese Common Management Facilities sollen ein Rahmenwerk für die Entwicklung von CORBA-basierten Managementanwendungen zur Verfügung stellen. Derzeit umfaßt ihre noch nicht standardisierte Spezifikation ([X/O95]) folgende Dienste:

- Der *Instance Management Service* stellt Möglichkeiten zu Verfügung, Instanzen von (Management-)Objekten zu verwalten. Sogenannte *Instance Manager* sind dabei für Instanzen genau eines Objekttyps zuständig. Jeder Instance Manager kennt Name und Lage der „seiner“ Instanzen. Damit ist etwa feststellbar, wieviele Objekte mit einer bestimmten Schnittstelle bereits instantiiert wurden.
- Der *Managed Set Service* unterstützt die Organisation von Objekten in Gruppen. Jede Gruppe zeigt dabei ein typisches Verhalten. Einfache Beispiele für Gruppen sind etwa Mengen (typisch: kein Element doppelt, ...) oder Binärbäume (typisch: können zur Liste entarten). Gruppen können hierarchisch angeordnet werden. Ein Objekt kann Element mehrerer Gruppen sein. So kann beispielsweise ein Objekt „Festplatte“ zu einer Gruppe „Festplatten_im_System“ und gleichzeitig zu einer Gruppe „Komponenten_des_PCs_nr.17“ gehören. Objekte, die die Schnittstellen des Managed Set Service unterstützen, können von einer Managementanwendung sehr komfortabel angeordnet und kombiniert werden.

- Der *Policy Management Service* ermöglicht die Definition von Regeln und Bedingungen, mit denen eine bestimmte Managementstrategie in einem System durchgesetzt wird. Es kann festgelegt werden, mit welchen Anfangswerten Objekte bei ihrer Erzeugung belegt werden, unter welchen Bedingungen ein Objekt „gültig“ ist und auf welche Weise überprüft werden kann, ob Objekte diese Bedingungen erfüllen. Zusammen mit dem Managed Set Service und dem *Policy-driven Base Service*ist es möglich, Managementdomänen festzulegen, in denen jeweils eigene Gesetzmäßigkeiten gelten.

Die Joint Inter-Domain Management Group

Die Liste der Organisationen, welche sich mit der Integration von Managementsystemen befassen (s. 2.2), muß an dieser Stelle ergänzt werden. Die Arbeitsgruppe *Joint Inter-Domain Management* (JIDM) ist ein gemeinsames Projekt von X/Open und NMF. Der Schwerpunkt ihrer Arbeit liegt darin, Werkzeuge zur Verfügung zu stellen, die die Interoperabilität von CMIP-, SNMP- und CORBA-basierten Managementframeworks ermöglichen.

In zweierlei Hinsicht soll die Interoperabilität erreicht werden:

- Die *Specification Translation* ([JID95]) beschreibt den Prozeß, durch den Objektspezifikationen aufeinander abgebildet werden. Dazu gehört vor allem die Abbildung zwischen der Beschreibungssprache ASN.1, mit denen die Managementinformation der OSI- und Internet-Architektur beschrieben werden, und OMG IDL, mit der CORBA-Objekte definiert werden. Der Algorithmus wird in Kapitel 3 beschrieben.

- Die *Interaction Translation* beschreibt die (dynamische) Protokollumsetzung zwischen CMIP/SNMP und CORBA. Diese Protokollumsetzung übernehmen Gateways. Zur Interaction Translation liegen derzeit nur wenige Vorschläge vor ([Sou95]), über die noch eine Einigung erzielt werden muß.

Kapitel 3

Kooperation von Managementarchitekturen

Um Interoperabilität zwischen verschiedenen Managementsystemen zu ermöglichen, muß ein Übergang von der einen Architektur in die andere geschaffen werden. Dazu werden die einzelnen Teilmodelle der Management-Architekturen (Informationsmodell, Kommunikationsmodell, Funktionsmodell und Organisationsmodell) aufeinander abgebildet. Da im Internet-Management bislang keine Ansätze zu einem Organisationsmodell und zu einem Funktionsmodell zu finden sind[1], sind für den Übergang des SNMP-Managements in eine CORBA-Umgebung nur Informations- und Kommunikationsmodell relevant. Der Schwerpunkt diese Kapitels ist die Abbildung von Internet-Objektspezifikationen auf OMG-Schnittstellenspezifikationen im Rahmen des Übergangs zwischen den jeweiligen Informationsmodellen.

3.1 Abbildung der Informationsmodelle

Das Informationsmodell einer Managementarchitektur beschreibt die dieser Architektur zugrundeliegende Datenbasis. Beispielsweise werden im Informationsmodell Zusammensetzung, Identifikation und Verhalten von managementrelevanten Informationen oder Objekten festgelegt. Beim Internet-Informationsmodell wird hierfür die Beschreibungssprache ASN.1 (Abstract Syntax Notation One) verwendet. Die Managementobjekte (Managed Objects) sind Informationseinheiten (einfache Variablen), welche als Blätter in einem Registrierungsbaum (der sogenannten Internet-MIB) zusammengefaßt und identifiziert werden. Die Internet-MIB (Managementinformationsbasis, Management Information Base) wird als konzeptioneller Behälter aller instantiierbaren Objekte und deren Beschreibungen aufgefaßt. Den Anteil an

[1] Als erster Schritt in Richtung Funktionsmodell kann die RMON-MIB (Remote-Network-Monitoring-MIB, [Wal95]) gewertet werden

Managementinformation, den ein Agent bereithält, wird in seiner Agenten-MIB vorgegeben.

Dem Datentypansatz des Internet-Managements steht die strikte Objektorientierung des CORBA-Ansatzes gegenüber. Die Schnittstellen der CORBA-Objekte werden in der C++-ähnlichen Beschreibungsnotation IDL (Interface Definition Language) festgelegt. Merkmale von Schnittstellen können vererbt werden, mit Einschränkungen wird sogar Mehrfach-Vererbung unterstützt.

Damit ein CORBA-Manager ohne geändert zu werden SNMP-Managed Objects verwalten kann, müssen diese Unterschiede in den Informationsmodellen transparent werden. Da auch die SNMP-Agenten, welche diese managementrelevante Information speichern, nicht verändert werden sollen, kann dies nur bewerkstelligt werden, indem die Internet-Managementobjekte in der vom Manager „verständlichen" IDL-Notation beschrieben werden. Eine wesentliche Maßnahme bei der Einbettung des Internet-Managements in eine CORBA-Umgebung ist folglich die Abbildung von Internet-Objektspezifikationen auf OMG-Schnittstellenspezifikationen. Die Makrosprache ASN.1 muß dafür in CORBA-eigenes IDL (und somit die SNMP-Objekte in CORBA-Objekte) übersetzt werden. Die Abbildung von CORBA-Managementob-

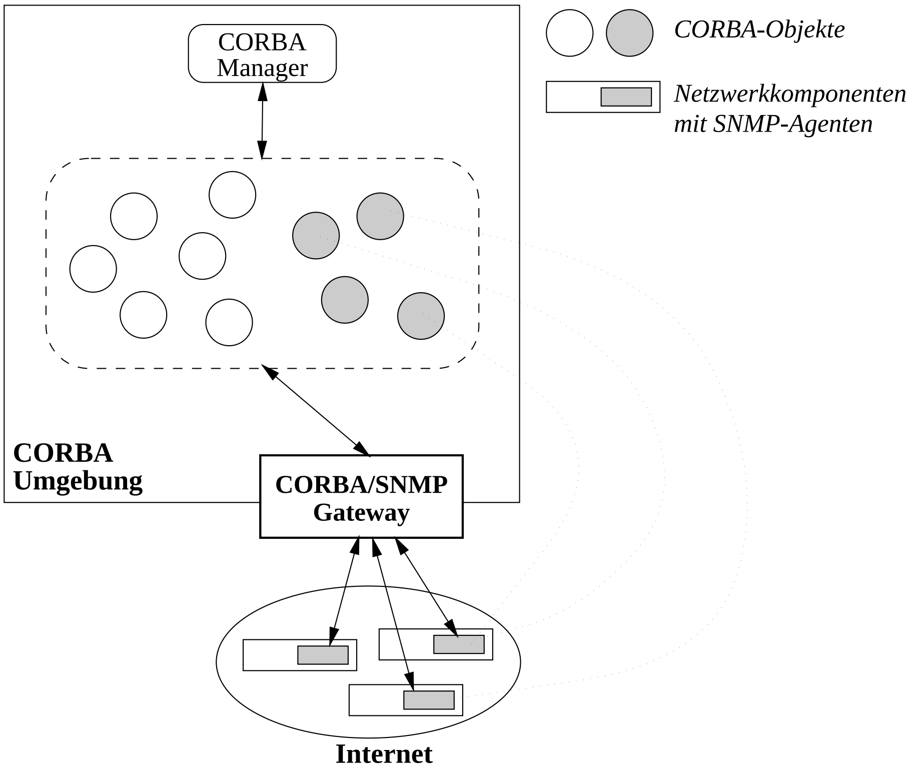

Abbildung 3.1: Einheitliche Sicht aller Managementobjekte

jekten auf Internet-Managementobjekte erübrigt sich, da es nicht sinnvoll ist, auf der Grundlage der Internet-SMI — ohne die Möglichkeit, Managementinformation ausreichend zu strukturieren — eine weitaus komplexere und mächtigere CORBA-Umgebung darzustellen oder sogar zu verwalten.

Mit der Beschreibung von Internet-Managementobjekten durch IDL (siehe nächster Abschnitt) werden die SNMP-Variablen und -Tabellen in CORBA-Objekte gepackt, wodurch dem Manager eine CORBA-konforme Sicht auf die fremden Ressourcen gewährt wird, welche diese Managementobjekte repräsentieren (Abb. 3.1, vgl. dazu auch das *Global Network Model* in [ACH93]). Dieser Vorgang ist statisch, d.h. er muß für jede MIB nur einmal geschehen. Liegt eine MIB-spezifische Objektdefinition in IDL vor, kann eine Management-Applikation auf entsprechende Objektinstanzen (und damit auf SNMP-Ressourcen) so zugreifen, wie sie es bei „normalen" CORBA-Objektinstanzen gewöhnt ist, nämlich über deren Methoden. Für den Manager bleibt es transparent, wie die Managementinformation aus der SNMP-Umgebung funktionell zur Verfügung gestellt wird. Dies ist die Aufgabe des Gateways.

Wie letztendlich die Internet-Managementobjekte in IDL-Schnittstellen übersetzt werden ist Thema der schon erwähnten *Specification Translation*([JID95]). Das dort beschriebene Verfahren wird im folgenden Abschnitt in groben Zügen wiedergegeben.

3.1.1 Übersetzung von ASN.1-Makros in IDL

Lexikalische Übersetzung, Namenskonventionen

Zuerst muß festgestellt werden, welche Unterschiede bei den jeweils verwendeten Zeichensätzen eventuell beachtet werden müssen, und wie Namen von Variablen und Typen systematisch abgebildet werden sollen. Die Abbildung vom ASN.1 Zeichensatz auf den von IDL ist einfach: beidesmal wird der Zeichensatz ISO Latin-1 (8859.1) angewandt.

Bei der Namensgebung ist es etwas komplizierter. Im Gegensatz zu ASN.1-Namen wird bei den entsprechenden IDL-Identifikatoren nicht zwischen Groß- und Kleinbuchstaben unterschieden. Außerdem müssen die drei Namensräume von ASN.1 (Typreferenzen, Wertreferenzen und Identifikatoren) auf einen einzigen IDL Namensraum abgebildet werden. Infolgedessen muß bei der Umsetzung darauf geachtet werden, daß keine Namenskonflikte auftreten. Damit die Abbildung nach IDL so einfach und direkt wie möglich bleibt, werden die Namen übernommen und müssen systematisch ergänzt werden, sodaß sie eindeutig sind.

Umsetzung einfacher Datentypen

Tabelle 3.1 zeigt die Umsetzung von (einigen) ASN.1-Typen in IDL Datentypen. Sie werden in einem File „ASN1Types.idl" abgelegt und in SNMP/SNMPv2-spezifischen .idl-Dateien (nicht notwendigerweise explizit) eingebunden. Der Grund, warum für

ASN.1-Typen	IDL-Typen
`NULL`	typedef octet ASN1_Null;
`BOOLEAN`	typedef boolean ASN1_Boolean;
`INTEGER`	typedef long ASN1_Integer;
`REAL`	typedef double ASN1_Real;
`ENUMERATED`	enum { item1, ... ,itemn };
`BIT STRING`	typedef sequence<octet> ASN1_BitSring;
`OCTET STRING`	typedef sequence<octet> ASN1_OctetSring;
`STRING`	typedef sequence<octet> ... ; typedef string ... ; typedef sequence<long> ... ; typedef sequence<unsigned short> ... ;
`OBJECT IDENTIFIER`	typedef string ASN1_Objectidentifier;
`CHOICE`	union/switch
`SET,SEQUENCE`	struct
`SET,SEQUENCE OF type`	typedef sequence<type> ... ;

Tabelle 3.1: Abbildung von ASN.1-Typen auf IDL-Typen

ASN.1 `STRING` mehrere IDL-Typen definiert wurden, sind die verschiedenen Varianten dieses Typs. Ein ASN.1 `STRING` kann eine einfache Zeichenkette oder eine Folge eines bestimmten Datentyps (z.B. ein 32-Bit-Wert) sein, wofür jeweils ein entsprechender IDL-Datentyp definiert werden muß.

Aus der obigen Liste der ASN.1-Datentypen sind in der Internet-SMI nur die Typen `NULL, INTEGER, OCTET STRING` und `OBJECT IDENTIFIER` sowie die Datentypen `SEQUENCE, SEQUENCE OF` zugelassen. Sie werden verwendet, um SNMP-anwendungsspezifische Datentypen (Zähler, IP-Adresse ...) zu definieren (s. [MR90]). Für diese gilt die Umsetzungstabelle 3.2. Bei SNMPv2 kommen noch die in Tabelle 3.3 aufgeführten ASN.1-Typen dazu (oder werden neu definiert, s. [CMRW96c], SNMPv2-SMI).

SNMP	IDL
`IpAddress`	sequence<octet, 4> IpAddressType;
`Counter`	typedef unsigned short CounterType;
`Gauge`	typedef unsigned short GaugeType;
`TimeTicks`	typedef unsigned short TimeTicksType;
`Opaque`	sequence<octet> OpaqueType;

Tabelle 3.2: SNMP-Typen in IDL

SNMPv2	IDL
`Integer32`	typedef long Integer32Type;
`Counter32`	typedef unsigned long Counter32Type;
`Gauge32`	typedef unsigned long Gauge32Type;
`TimeTicks`	typedef unsigned long TimeTicksType;
`Counter64`	typedef long Counter64Type;
`Unsigned32`	typedef unsigned long Unsigned32Type;

Tabelle 3.3: SNMPv2-Typen in IDL

MODULE-IDENTITY und OBJECT-IDENTITY

Beim Internet-Management werden die Managementobjekte über den Internet-Registrierungsbaum identifiziert und benannt. Dieser Baum besteht aus Strukturierungsknoten und Informationsknoten. Letztere sind die Blätter des Baumes, in denen Agenten (managementrelevante) Information abspeichern können[2]. Ein SNMP(v2)-Informationsmodul enthält die Beschreibung (in ASN.1 Notation) eines Teilbaumes der Internet-MIB, mit Strukturierungs- und/oder Informationsknoten. Beispiel für ein solches Modul ist die MIB-II ([MR91]):

```
RFC1213-MIB DEFINITIONS ::= BEGIN
IMPORTS
        mgmt, NetworkAddress, IpAddress, Counter, Gauge,
                TimeTicks
            FROM RFC1155-SMI
```

[2]Genaugenommen muß man unterscheiden zwischen dem gesamten Internet-Registrierungsbaum (der Internet-MIB) und einer Agenten-MIB: in der Internet-MIB sind *alle* instantiierbaren Objekte mit deren Beschreibungen, (Zugriffseigenschaften, Verhalten etc.) gesammelt. Sie ist also als ein konzeptioneller Behälter aller Managed Objects aufzufassen. Ein Agent hingegen verwaltet in seiner MIB *Instanzen von Objekten eines Teilbaumes* der Internet-MIB. In diesen Instanzen wird Managementinformation gespeichert.

```
        OBJECT-TYPE
                FROM RFC-1212;
mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }

-- textual conventions
DisplayString ::=
    OCTET STRING
PhysAddress ::=
    OCTET STRING
-- groups in MIB-II

system       OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces   OBJECT IDENTIFIER ::= { mib-2 2 }
at           OBJECT IDENTIFIER ::= { mib-2 3 }
ip           OBJECT IDENTIFIER ::= { mib-2 4 }
icmp         OBJECT IDENTIFIER ::= { mib-2 5 }
tcp          OBJECT IDENTIFIER ::= { mib-2 6 }
udp          OBJECT IDENTIFIER ::= { mib-2 7 }
egp          OBJECT IDENTIFIER ::= { mib-2 8 }
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
snmp         OBJECT IDENTIFIER ::= { mib-2 11 }

-- Formale Beschreibung der einzelnen Gruppen

END
```

Jedes solche Modul, wird auf ein IDL-Modul mit gleichem Identifikator (im Beispiel `RFC1213-MIB`) abgebildet. Die dazugehörige Datei wird nach demselben Modul und dem Suffix „.idl“ benannt. Diese Ausgabedatei — in sie werden alle den ASN.1-Makros des Moduls entsprechenden IDL-Definitionen geschrieben — heißt bei obigem Beispiel „rfc1213.idl“ und sieht (für obigen Auszug) folgendermaßen aus[3]:

```
// rfc1213.idl
#ifndef RFC1213_MIB_idl
#define RFC1213_MIB_idl
#include "SNMPMgmt.idl"
module RFC1213_MIB {
#include "rfc1155.idl"
typedef RFC1155_SMI::TimeTicksType TimeTicksType;
typedef RFC1155_SMI::GaugeType GaugeType;
```

[3]Man achte dabei auch auf die Ähnlichkeit von IDL mit C++; Die Datei wurde automatisch von einem SNMP/IDL Übersetzer generiert

```
typedef RFC1155_SMI::CounterType CounterType;
typedef RFC1155_SMI::IpAddressType IpAddressType;
typedef RFC1155_SMI::NetworkAddressType NetworkAddressType;
#define mgmt RFC1155_SMI::mgmt;
const ASN1_ObjectIdentifier mib_2 = "mgmt.1";
const ASN1_ObjectIdentifier system = "mib_2.1";
const ASN1_ObjectIdentifier interfaces = "mib_2.2";
const ASN1_ObjectIdentifier at = "mib_2.3";
const ASN1_ObjectIdentifier ip = "mib_2.4";
const ASN1_ObjectIdentifier icmp = "mib_2.5";
const ASN1_ObjectIdentifier tcp = "mib_2.6";
const ASN1_ObjectIdentifier udp = "mib_2.7";
const ASN1_ObjectIdentifier egp = "mib_2.8";
const ASN1_ObjectIdentifier transmission = "mib_2.10";
const ASN1_ObjectIdentifier snmp = "mib_2.11";
// IDL-Beschreibung der Gruppen
}
#endif
```

Wichtig ist, daß die Objektidentifikatoren als konstante Strings übernommen werden. Die Objektidentifikatoren werden für die Adressierung der Managed Objects in den SNMP(v2)-PDUs benötigt.

Abbildung der Managementobjekte

Im folgenden werden SNMP und SNMPv2 — wenn nicht explizit unterschieden — synonym verwendet. Die Struktur der Managementinformation bei SNMPv2 ist umfangreicher und schließt (im Sinne der Abwärtskompatibilität) jene der Vorgängerversion SNMP mit ein. SNMP-MIB-Module können in SNMPv2-Module unter Vorlage von [CMRW96a] umgewandelt werden.

Das `OBJECT-TYPE`-Makro definiert den Typ eines ASN.1-Managementobjektes. Man unterscheidet zwischen einfachen *skalaren Variablen* und *Listen* bzw. *Tabellen*. Letztere Tabellen-Objekte sind genaugenommen wiederum nur einzelne Variablen, die entsprechend angeordnet sind. Variablen und Tabellen werden im MIB-Baum in Gruppen zusammengefaßt. Beispielsweise findet man in der Standard-MIB (MIB-II) 10 (disjunkte) Gruppen. Eine dieser Gruppen, „interfaces“, besteht aus einer Variablen und einer Tabelle mit 22 Spalten (Abb. 3.2, vgl. [MR91]). Analog zur Strategie von IIMC bei der Übersetzung von SNMP-Dokumenten in GDMO-Dokumente werden diese SNMP-Gruppen aufgespalten und zwar definiert der *Translation Algorithm*:

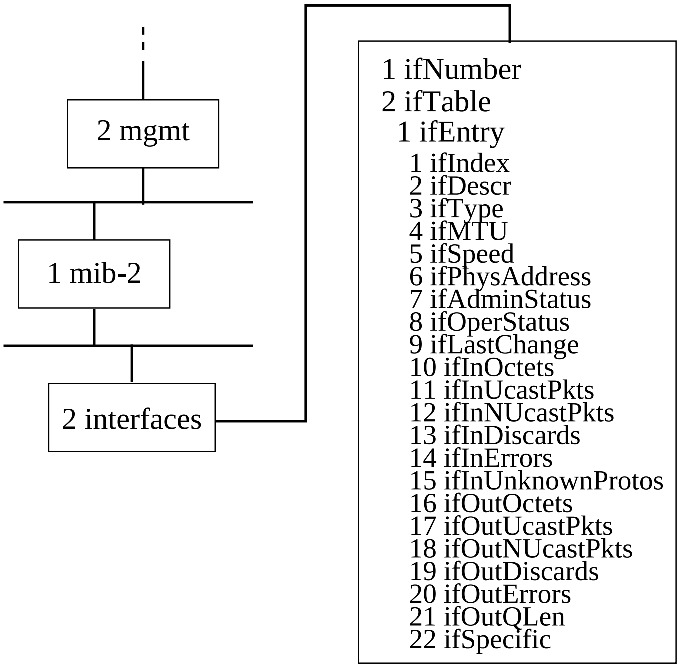

Abbildung 3.2: Die “interfaces”-Gruppe der MIB-II

- für jede ASN.1-Tabelle eine IDL-Objektklasse, auch „(Tabellen-)Zeile-Objekt“ (engl. row of a table object, [JID95]) genannt; Die Spalten der ursprünglichen Tabelle werden zu Attributen dieser Klasse. Eine Instanz einer solchen Objektdefinition entspricht also genau einer Zeile der Ausgangstabelle, nicht der Tabelle selbst.

- eine IDL-Objektklasse (oder „Gruppenobjekt“, engl. group object) für die Variablen einer SNMP-Gruppe. Die Variablen werden zu den Attributen dieser Klasse. Ein Gruppenobjekt kann als spezielle Tabelle aufgefaßt werden, die nur aus einer einzigen Zeile besteht.

Jede neue Schnittstellendefinition erbt von einer gemeinsamen Objektklasse SmiEntry (s. Abb. 3.3). Sie dient als Platzhalter für Attribute und Operationen, die einen generellen Zugriff auf SNMP-Managementobjekte implementieren. Gruppen- und Zeilenobjekte befinden sich in der Vererbungshierarchie der IDL-Schnittstellen auf gleichem Niveau (Abb. 3.3). Insbesondere sind alle Objekte, das IDL-Gruppenobjekt und jedes Zeilenobjekt, die zu einer SNMP-Gruppe erzeugt werden, gleichberechtigt. Die ursprüngliche Zugehörigkeit von Variablen und Tabellen zu einer bestimmten SNMP-Gruppe ist also nicht mehr direkt sichtbar. Die „Enthaltensein“-Relation zwischen SNMP-Gruppen und den ASN.1-Tabellenobjekten bleibt aber durch die mitübernommenen Objektidentifikatoren erhalten, d.h. die Zusammengehörigkeit

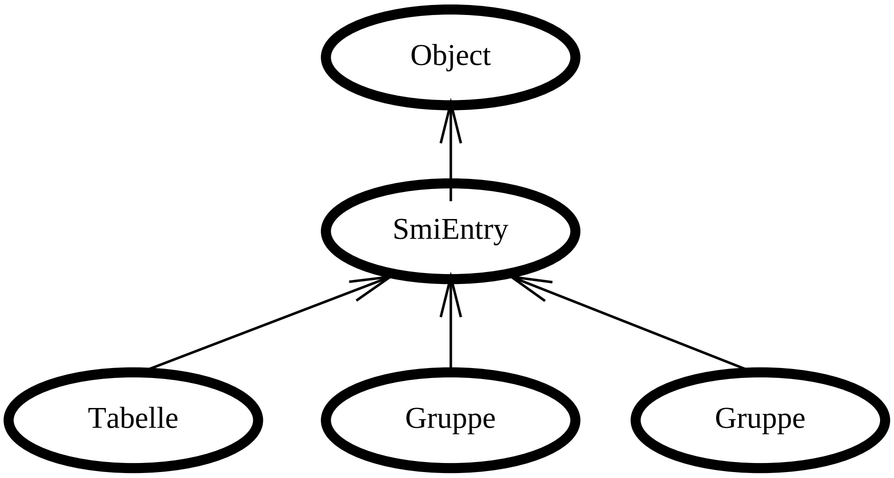

Abbildung 3.3: Vererbungshierarchie der IDL-Schnittstellen für Tabellen/Gruppen

von IDL-Zeilenobjekten und IDL-Gruppenobjekten ist weiterhin eindeutig.

Anhand der erwähnten SNMP-Gruppe „interfaces" soll nun der Vorgang etwas genauer besprochen werden. Ausgangspunkt ist die Definition dieser SNMP-Gruppe in [MR91]:

```
...
-- the System group
...

-- the Interfaces group

ifNumber OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
          ``The number of network interfaces ... ``
     ::= { interfaces 1 }

ifTable OBJECT-TYPE
     SYNTAX SEQUENCE OF IfEntry
     ACCESS not-accessible
     STATUS mandatory
     DESCRIPTION
          `` A list of interface entries. ...''
```

```
     ::= { interfaces 2 }

ifEntry OBJECT-TYPE
     SYNTAX IfEntry
     ACCESS not-accessible
     STATUS mandatory
     DESCRIPTION
          ``An interface entry containing objects ... ``
     INDEX { ifIndex }
     ::= { ifTable 1 }

IfEntry ::=
     SEQUENCE {
          ifIndex INTEGER,
          ifDescr DisplayString,
          ...
          ifSpecific OBJECT IDENTIFIER
     }

ifIndex OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
          ``A unique value for each interface. ...''
     ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
     ...
     ::= { ifEntry 2 }

...

ifSpecific OBJECT-TYPE
     SYNTAX OBJECT IDENTIFIER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
          ``A reference to MIB definitions specific ...''
     ::= { ifEntry 22 }

-- the Address Translation group
...
```

Diese Gruppe besteht aus einer skalaren Variable (`ifNumber`) und einer Tabelle (`ifTable`) mit 22 Spalten (`ifEntry 1-22`). Es werden also eine Schnittstellendefinition für ein Gruppenobjekt und eine für ein Tabellen-Zeile-Objekt angelegt. Beide Klassen werden von der Klasse SmiEntry, die im Modul SNMPMgmt definiert wurde, abgeleitet.

Der Name der Klasse für das Gruppenobjekt ergibt sich aus dem Namen der SNMP-Gruppe und dem Suffix Group. Das Managementobjekt `ifNumber` wird auf ein Attribut dieser Klasse abgebildet. Dabei werden der Name aus dem Deskriptor der Variablen, Zugriffsrechte aus der `ACCESS`-Klausel und Typ aus der `SYNTAX`-Klausel der Variablen übernommen. Da die SNMP-Gruppe keine weiteren *non-tabular* Objekte besitzt, hat die IDL-Schnittstelle nur dieses eine Attribut; die Tabelle der SNMP-Gruppe wird ignoriert, da ein eigene IDL-Schnittstelle dafür vorgesehen ist. Die OID der SNMP-Variablen wird als Konstante definiert. Da diese Konstante den gleichen Namen trägt wie die Variable bzw. ihr entsprechendes Attribut, muß dies, um Namenskonflikte zu vermeiden, außerhalb der interface-Deklaration erfolgen:

```
...
interface InterfacesGroup:SNMPMgmt::SmiEntry {
/* DESCRIPTION:
"The number of network interfaces ..." */

readonly attribute ASN1_Integer ifNumber;

};
const ASN1_ObjectIdentifier ifNumber="interfaces.1";
```

Eine Tabelleninstanz auf einem SNMP-Agenten wird in der CORBA-Umgebung durch Objektinstanzen der Zeilen dieser Tabelle repräsentiert. Deshalb wird eine Klasse für eine Tabellenzeile und nicht für eine ganze Tabelle definiert. Da eine Zeile einer SNMP-Tabelle eine Liste (`SEQUENCE`) von einzelnen skalaren Variablen ist (s. [CMRW96c]), ergibt sich der Name einer solchen Klasse aus dem Namen dieser Liste (und nicht aus dem Namen der Tabelle) und dem Suffix Object. Im obigem Beispiel besteht eine Zeile der Tabelle `ifTable` aus den in der Liste `IfEntry` angeführten Spaltenelementen. In der zu erzeugenden Schnittstelle IfEntryObject, werden die Spalteneinträge, Variablen einfacher skalarer Typen, zu den Attributen dieser neuen Klasse. Analog zu oben werden Namen, Zugriffsrechte und Datentypen von den Variablen übernommen, und die Objektidentifikatoren von `ifTable`, `ifEntry` und von den Spaltenelementen werden auf Strings abgebildet:

```
...
interface IfEntryObject:SNMPMgmt::SmiEntry {
```

```
/* INDEX ifEntry */
//—————————————-
    /* DESCRIPTION:
    “A unique value for each interface. ...” */

    readonly attribute ASN1_Integer ifIndex;
//—————————————-
    /* DESCRIPTION:
    “ A textual string ... “ */
    typedef sequence <DisplayStringType, 255> IfDescrType;

    readonly attribute IfDescrType ifDescr;
// ————————————
    ...
    DESCRIPTION:
    ”A reference to MIB definitions specific ...” */

readonly attribute ASN1_ObjectIdentifier ifSpecific;
};
const ASN1_ObjectIdentifier ifTable=”interfaces.2”;
const ASN1_ObjectIdentifier ifEntry=”ifTable.1”;
const ASN1_ObjectIdentifier ifIndex=”ifEntry.1”;
const ASN1_ObjectIdentifier ifDescr=”ifEntry.2”;
...
const ASN1_ObjectIdentifier ifSpecific=”ifEntry.22”;
```

Abbildung 3.4 zeigt die SNMP-Gruppe „interfaces“ im Internet-Registrierungsbaum und als IDL-Klasse(n) in der IDL-Schnittstellenhierarchie. Ein Managementsystem kann sich für die Variable `ifNumber` und für jede Zeile der SNMP-Tabelle `ifTable` ein Objekt dieser Klassen instantiieren, um auf die Informationen der SNMP-Gruppe zuzugreifen.

Abbildung des NOTIFICATION-TYPE Makros

Ereignismeldungen sind beim Internet-Management SNMP-Protokolleinheiten, die von einer SNMP-Einheit (einem Manager oder einem Agenten) unaufgefordert an eine zweite SNMP-Einheit gesendet werden. Im Gegensatz dazu entspricht die Meldung eines Events in der CORBA-Umgebung im Prinzip dem Aufruf einer bestimmten (*push-* oder *pull-*)Methode eines Objektes (s. 2.3.1). Bei der *push-event-*Kommunikation etwa ruft ein Objekt, das ein Ereignis einem anderen Objekt melden will, mit einem CORBA-Request eine *push*-Methode auf diesem Objekt auf. Um

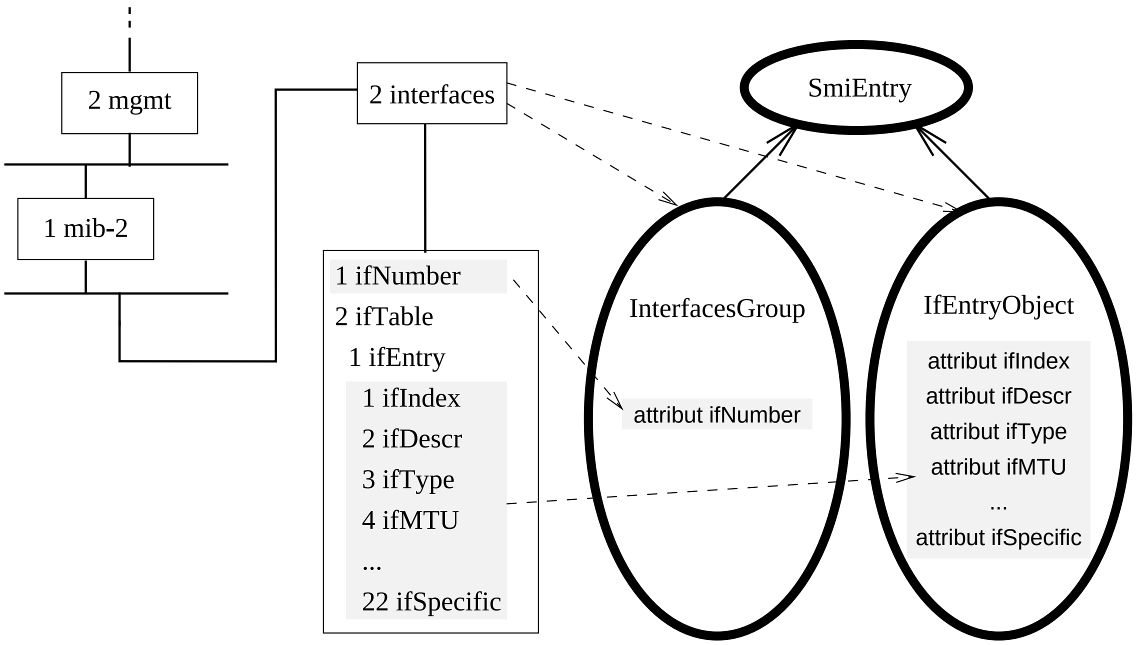

Abbildung 3.4: Die SNMP-Gruppe „interfaces“ und die entsprechenden Klassen

diesen Mechanismus zu gewährleisten, werden für jedes SNMP-Modul, in dem Ereignismeldungen vorgesehen sind, genau zwei Schnittstellen definiert, die entsprechende *push*- bzw. *pull*-Methoden zur Verfügung stellen:

- SnmpNotifications mit den *push*-Methoden für die typisierte *push-event*-Kommunikation (s. Kap. 2);
- PullSnmpNotifications mit den *pull*-Methoden für die typisierte *pull-event*-Kommunikation (s. Kap. 2).

Die *push*- bzw. *pull*-Methoden ergeben sich aus den NOTIFICATION-TYPE Makros eines MIB-Moduls, da mit diesem festgelegt wird, welche Managementinformation in einem SNMP-Trap enthalten ist.

Für jedes NOTIFICATION-TYPE Makro werden diese Schnittstellen um insgesamt drei Methoden erweitert und zwar:

- die Schnittstelle SnmpNotifications um die Funktion <notification_name>, wobei <notification_name> der Identifikator des NOTIFICATION-TYPE-Makros ist,
- die Schnittstelle PullSnmpNotifications um die beiden Funktionen try_<op> und pull_<op>. <op> ist der Name der korrespondierenden Operation in

SnmpNotifications.

Die Parameter enthalten die OID und den Kontext der Quellinstanz sowie den Zeitpunkt des Traps. Folgendes Beispiel des NOTIFICATION-TYPE `linkUp` erläutert die Vorgangsweise genauer:

```
...

linkUp NOTIFICATION-TYPE
     OBJECTS { ifIndex }
     STATUS current
     DESCRIPTION
          ''A linkUp trap signifies that the SNMPv2 entity ...''
     ::= { snmpTraps 4 }
...
```

Damit stehen zunächst die Namen der Funktionen fest: linkUp für die Schnittstelle SnmpNotifications einerseits, und pull_linkUp sowie try_linkUp für die Schnittstelle PullSnmpNotifications andererseits. Mit der Funktion try_linkUp kann abgefragt werden, ob ein Supplier eine solche Ereignismeldung vorrätig hat (s. 2.3.1). Ihr Rückgabewert ist abhängig davon entweder *true* oder *false*. Die beiden anderen Funktionen liefern keine Werte zurück.

In der optionalen OBJECTS-Klausel werden jene MIB-Objekte aufgelistet, deren Information zusätzlich mit der Ereignismeldung verschickt wird. Im Beispiel besteht diese Liste nur aus dem Managed Object `ifIndex`, also dem Zeilenindex des soeben „eingeschalteten“ Links in der `Interface`-Tabelle. Die OBJECTS-Klausel wird auf ein IDL-Strukt abgebildet. Der Name dafür ist der Deskriptor des NOTIFICATION-TYPE-Makros mit dem Suffix Type. Die Elemente sind die Werte der Managed Objects in der OBJECTS-Liste, den Typ der Elemente erhält man aus der SYNTAX-Klausel des jeweiligen OBJECT-TYPE-Makros. Dieser IDL-Strukt wird ein weiterer in-Parameter der Methode linkUp bzw. ein weiterer out-Parameter der Methoden pull_linkUp und try_linkUp. Es sei darauf hingewiesen, daß bei der generischen Event-Kommunikation nur ein Parameter vom Typ *any* erlaubt ist (s. 2.3.1). Dies war mitunter ein Grund, für SNMP-Traps typisierte Events vorzusehen. Dennoch sind im Modul SNMPMgmt für generische Eventkommunikation die Schnittstellen GenericNotification und PullGenericNotification mit den Methoden snmp_notification bzw. pull_snmp_notification und try_snmp_notification definiert.

Schließlich wird der OBJECT IDENTIFIER des NOTIFICATION-TYPE-Makros als konstante Zeichenkette übernommen. Für dieses Beispiel erhält man das folgende IDL-Mapping:

struct IfIndexType {

```
    ASN1_ObjectIdentifier var_name;
    ASN1_ObjectIdentifier var_value;
};
struct LinkUpType {
    ASN1_Integer ifIndex;
};
const ASN1_ObjectIdentifier linkUp = "snmpTraps.4";
/* DESCRIPTION:
    "A linkUp trap signifies that the SNMPv2 entity, ..." */

interface SnmpNotifications {
    void linkUp(
        in ASN1_ObjectIdentifier srcParty,
        in ASN1_ObjectIdentifier snmpContext,
        in SNMPv2TC::TimeStampType eventTime,
        in LinkUpType notification_info);
};
interface PullSnmpNotifications {
    void pull_linkUp(
        out ASN1_ObjectIdentifier srcParty,
        out ASN1_ObjectIdentifier snmpContext,
        out SNMPv2TC::TimeStampType eventTime,
        out LinkUpType notification_info);
    boolean try_linkUp(
        out ASN1_ObjectIdentifier srcParty,
        out ASN1_ObjectIdentifier snmpContext,
        out SNMPv2TC::TimeStampType eventTime,
        out LinkUpType notification_info);
};
```

Es sei nochmals erwähnt, daß die Schnittstellen SnmpNotifications und PullSnmpNotifications für jedes Modul nur einmal definiert werden. Wenn mehrere NOTIFICATION-TYPE-Makros in einem MIB-Modul vorhanden sind, so werden diese Schnittstellen um entsprechende Methoden ergänzt.

Zusammenfassung

Bei der Umsetzung von SNMP-MIBs in IDL Objektdefinitionen sind folgende Schritte durchzuführen:

1. Jedes einzelne SMI-Modul wird einem IDL-Modul zugeordnet. In diesem werden alle sich ergebenden Schnittstellen, Typen und Konstanten abgelegt. Wenn

es mindestens ein NOTIFICATION-TYPE-Makro im SNMP Informationsmodul gibt, werden zusätzlich zwei IDL-Schnittstellen SnmpNotifications und PullSnmpNotifications für die typisierte *push*- respektive *pull*-Event-Kommunikation definiert;

2. ASN.1-Typen werden auf IDL-Typen abgebildet; insbesondere werden die Typspezifikationen der TEXTUAL-CONVENTION-Makros zu IDL-Typen;

3. MODULE-IDENTITY-, OBJECT-IDENTITY- und OBJECT-TYPE-Makros werden auf konstante IDL Literale vom Typ ASN1_ObjectIdentifier abgebildet;

4. Für jedes Tabellenobjekt und für jede Gruppe im SMI-Dokument wird eine Schnittstelle definiert. Dabei wird

   - zu jeder Spaltenvariable einer Tabelle und
   - zu jeder Variable einer Gruppe

   ein IDL-Attribut in der entsprechenden Schnittstelle deklariert, wobei Identifikator, Typ und Zugriffsrechte der Attribute aus den OBJECT-TYPE Makros der dazugehörigen Variablen abgeleitet werden.

5. Zu jedem NOTIFICATION-TYPE Makro werden drei Funktionen deklariert:

   - im Interface SnmpNotifications die Funktion <notification_name>, wobei <notification_name> der Identifikator des NOTIFICATION-TYPE-Makros ist,
   - im Interface PullSnmpNotifications die beiden Funktionen try_<op> und pull_<op>. <op> ist der Name der korrispondierenden Funktion in SnmpNotifications.

3.1.2 Bewertung des Algorithmus

Der vorgestellt Algorithmus kann in vieler Hinsicht verbessert werden, z.B. durch

Definition von Superklassen: Viele Schnittstellendefinitionen benutzen dieselben Attribute (z.B. eine Indexvariable). Diese Schnittstellen können unter einer Superklasse (mit den entsprechenden Eigenschaften) zusammengefaßt werden.

Subklassenbildung: Viele Schnittstellen sind bloß Verfeinerungen anderer Schnittstellen. Dieses Schnittstellen können von letzteren die gemeinsamen Eigenschaften erben und nur schnittstellenspezifische Eigenschaften hinzufügen. Z. B. wenn eine herstellerspezifische Agenten-MIB in der SNMP-Gruppe `System` neben den per Definition vorgeschrieben Variablen auch noch eine Variable

`SysNickName` unterstützt, so könnt das entsprechende IDL-Interface von der Schnittstellendefinition SystemGroup abgeleitet werden:

```
interface Hrst_SysGroup:SNMPMgmt::SmiEntry::SystemGroup {

attribute ASN1_String SysNickName;
};
```

Behandlung der Pushbuttons: Sogenannte „Pushbutton“-Variable sind Variable, welche bei schreibenden Zugriff eine Aktion auslösen. Diese Variablen besitzen also die Semantik von Funktionsaufrufen. Es wäre demzufolge korrekter, wenn in der zu erzeugenden IDL-Schnittstellen statt eines Attributs eine Methode für diesen Pushbutton definiert würde.

Es würde den Rahmen dieser Diplomarbeit sprengen, einen Sprachcompiler zu bauen, der diese Aspekte bei der Erzeugung von IDL-Schnittstellen berücksichtigt. Stattdessen wird ein bestehender Sprachcompiler verwendet, der nach dem beschriebenen Muster arbeitet.

3.2 Abbildung der Kommunikationsmodelle

Neben der Beschreibung der managementrelevanten Information, müssen in einer Managementarchitektur auch Konzepte zum Austausch dieser Information festgelegt werden. Dies geschieht im Kommunikationsmodell einer Managementarchitektur. Charakteristische Gesichtspunkte eines Kommunikationsmodell sind die Definition von Kommunikationsmechanismen und -diensten, die Definition von Syntax und Semantik von Austauschformaten und schließlich auch die Festlegung der kommunizierenden Partner.

Beim Informationsaustauch zwischen Datenquelle und -senke müssen, je nach zugrundeliegender Managementarchitektur, genau die in dem Kommunikationsmodell spezifizierten Regeln eingehalten werden. Die im vorherigen Abschnitt beschriebene Informationsabbildung betrifft nur die Information, die ausgetauscht werden soll. Wenn über die Grenzen zweier Managementarchitekturen kommuniziert werden soll, müssen deshalb die architekturspezifischen Unterschiede in den jeweiligen Kommunikationsmodellen ebenso betrachtet und überwunden werden.

Beim Internet-Management werden Daten über das SNMP-Protokoll übertragen. Ein Manager kann mit den Protokolleinheiten `GetRequest`, `GetNextRequest`, `GetBulk` und `SetRequest` auf die vom Agenten bereitgestellte Information zugreifen. Mit SNMPv2 steht ihm außerdem eine Inform-PDU zur Verfügung, mit der Informationen zwischen anderen Managern ausgetauscht werden können. Der Agent antwortet seinerseits mit einer `GetResponse-PDU`, in die er die angeforderten Daten packt. Außerdem hat der Agent die Möglichkeit, Ereignismeldungen in Form von `Trap-PDUs` an einen Manager zu schicken (Abb. 3.5, links).

Im Gegensatz dazu beruht „Kommunikation“ im CORBA-System auf Methodenaufrufen auf Objekten. Ein Client-Objekt kann durch den Aufruf einer Methode eines Server-Objektes Informationen oder Dienste anfordern. Der Name des Zielobjektes, die gewünschte Funktion und eventuelle Parameter werden in einem sogenannten CORBA-*Request* dem ORB übergeben, der die Anfrage an das adressierte Objekt weiterleitet und Rückgabewerte (eine CORBA-Response) gegebenenfalls an den Client zurückgibt. Ein Server-Objekt kann bei der Abarbeitung einer Operation zu einem Client-Objekt eines dritten Objektes werden (Abb. 3.5, rechts). Auch Events werden durch den Aufruf spezieller Supplier/Consumer-Methoden weitergegeben (vgl. 2).

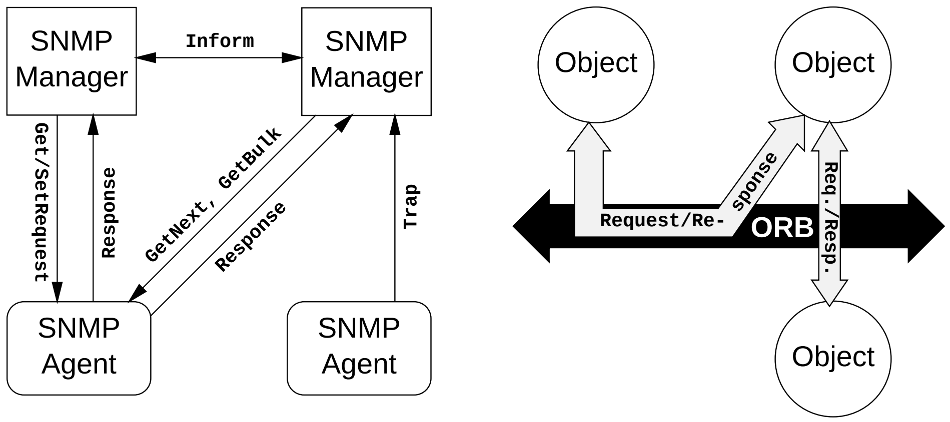

Abbildung 3.5: Kommunikation zwischen SNMP-Einheiten, CORBA-Objekten

Für den Datenaustausch zwischen den zwei Protokollwelten CORBA und SNMP bedarf es einer Protokollumsetzung. Diese Umsetzung übernimmt ein CORBA/SNMP-Gateway. Es stellt sich die Frage, wieviel Funktionalität das Gateway sinnvollerweise besitzen soll. Um nur ein Beispiel zu nennen, könnte das Gateway SNMP-Managementinformation speichern. Es könnte dann selbst entscheiden, ob ein Zugriff auf einen SNMP-Agenten notwendig ist. Um derartige Anforderungen festzustellen werden im folgenden Kapitel mögliche Gateway-Einsatzszenarien analysiert.

Kapitel 4

Das CORBA/SNMP Gateway

4.1 Anforderungen

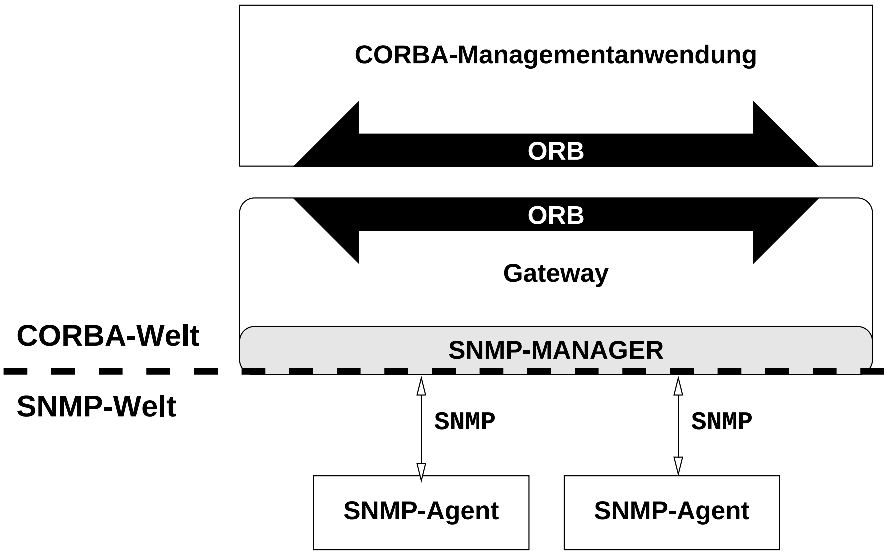

Abbildung 4.1: Die Rolle des CORBA/SNMP-Gateways

Ein CORBA/SNMP-Gateway vermittelt zwischen einer CORBA- und einer SNMP-Welt. In der SNMP-Umgebung befinden sich mehrere SNMP-Agenten, welche Managementinformation über Ressourcen bereitstellen. Diese sollen von einer CORBA-Managementanwendung verwaltet werden. Die Aufgabe des Gateway ist es, dies zu ermöglichen. Dazu lassen sich zunächst zwei unterschiedliche Rollen des Gateways feststellen (Abbildung 4.1):

- aus CORBA-Sicht ist das Gateway eine Anwendung, die über einen ORB mittels CORBA-Requests angesprochen werden kann. Insbesondere kann eine CORBA-Managementanwendung in der gewohnten Weise mit dem Gateway kommunizieren.

- aus der Sicht der SNMP-Agenten hingegen ist das Gateway ein SNMP-Manager.

Innerhalb des Gateways findet der Übergang von der CORBA- in die SNMP-Umgebung statt. Dieser Übergang soll sowohl für den Manager als auch für SNMP-Agenten verdeckt sein. Im Detail bedeutet dies, daß das Gateway folgende Anforderungen erfüllen muß:

Darstellung der SNMP-Ressourcen in einer für den Manager zugänglichen Form

Die Managementinformation in den Agenten wird in Form von *Managementobjektinstanzen* (kurz: Managementobjekte, SNMP-Instanzen, Managed Objects) gespeichert. Das Gateway soll die Inhalte der Agenten-MIBs dem Manager zugänglich machen. Dazu werden die Agenten-MIBs von einem Sprachcompiler in IDL-Klassen übersetzt (s. Kapitel 3). Die entsprechenden CORBA-konformen Objektinstanzen sind Proxy-Objekte der Managementobjekte in der SNMP-Umgebung. Um spätere Mißverständnisse durch die Bezeichnung „Proxy-Objekt“ in Bezug auf die SOM/DSOM Entwicklungsumgebung auszuschließen (siehe Kapitel 5), werden sie als *Schattenobjekte (Shadow Objects, Gateway-Objekte)* bezeichnet.

Aufgabe des Gateways ist es nun, die Managementobjekte in Form von Schattenobjekten konsistent zum Manager hin abzubilden:

1. Jede Variableninstanz eines SNMP-Agenten wird von genau einem Attribut oder von einer Methode eines Schattenobjektes repräsentiert. Letzterer Fall ist beispielsweise bei „push-button“-Variablen denkbar (vgl. 3.1.2).

2. Es darf kein Schattenobjekt existieren oder erzeugt werden, wenn es zu den Attributen des Objektes keine entsprechenden SNMP-Variableninstanzen gibt.

3. Wenn Inhalte der SNMP-Informationseinheiten in den Attributen von Schattenobjekten gespeichert werden, müssen diese Attributwerte zum *Zeitpunkt der Abfrage* mit den Werten in den entsprechenden Managed Objects konsistent sein.

Manager und Agent werden nicht verändert

Der Übergang in die SNMP-Umgebung soll für die Managementanwendung transparent bleiben, d. h. sie bzw. deren Code wird für den Zugriff auf SNMP-Ressourcen nicht geändert. Ein bestehender CORBA-Manager schickt CORBA-Requests an die Schattenobjekte oder an das Gateway und erhält CORBA-Responses von diesen zurück. Für ihn ist es nicht erkennbar, daß dabei auf SNMP-Ressourcen zugegriffen wird.

Weiters dürfen beim Entwurf des Gateways keine Annahmen über die Erweiterbarkeit von Agenten gemacht werden. Die Erweiterung von Agenten, beispielsweise um eine CORBA-Schnittstelle (vgl. multiarchitekturelle Agenten, 2.1.3), ist nur selten und nur auf Ressourcen mit hoher Performance möglich. Bei der Mehrheit der Agenten stehen weder Quellcode noch eine API zur Verfügung. Deshalb soll kein Agent für die Kommunikation mit dem Gateway geändert werden; der Zugriff auf alle Agenten darf ausschließlich über die standardisierte SNMP-Schnittstelle erfolgen. Mit anderen Worten, muß aus Sicht des Agenten die Syntax und Semantik des Informationsaustausches gleich bleiben, ungeachtet dessen, ob mit einem SNMP-Manager oder mit dem Gateway kommuniziert wird.

Replikation von Operationen

Operationen, die der Manager auf Schattenobjekten ausführt, müssen in die SNMP-Umgebung repliziert werden. Voraussetzung ist, daß das Gateway bei einem Zugriff erkennen kann, welches Schattenobjekt und damit welches Managed Object auf welchem SNMP-Agenten das Zielobjekt des Zugriffs ist. Außerdem muß die Art des Zugriffs (lesen, schreiben, löschen, ...) für das Gateway erkennbar sein.

- Beim Löschen und Erzeugen eines Schattenobjektes müssen, wenn möglich, die betroffenen Managed Objects gelöscht bzw. erzeugt werden. Dieser Fall betrifft nur die Schattenobjekte, die eine SNMP-Tabellenzeile repräsentieren, da nur Managementobjekte einer solchen SNMP-Tabellenzeile „gelöscht“ (als ungültig gekennzeichnet) oder “erzeugt“ (schreibender Zugriff auf eine in der SNMP-Tabelle noch nicht existierende Zeile) werden können (s. Hinweis unten).

- Eine Schreiboperation auf ein Attribut eines Schattenobjektes muß auf eine Schreiboperation (also auf eine `SetRequest-PDU`) auf das entsprechende Managed Object abgebildet werden.

- Eine Leseoperation auf ein Attribut eines Schattenobjektes muß auf eine Leseoperation (eine `GetRequest-PDU`, eine `GetNextRequest-PDU` oder eine `GetBulk-PDU`) auf das oder die entsprechenden Managed Objects abgebildet werden.

Um Operationen auf Schattenobjekten in die SNMP-Umgebung zu replizieren, ist es notwendig, Schattenobjekte und deren Attribute auf entsprechende Managementobjektinstanzen abzubilden. Diese Abbildung findet im Gateway statt. Für den Zugriff auf eine konkrete Variableninstanz in der SNMP-Umgebung wird folgende Information benötigt:

- den Kontext der Variableninstanz, i.e. die Adresse des Agenten,
- der Objektidentifikator (OID) der Variable im Registrierungsbaum und
- einen *Zugriffsidentifikator*, i.e. eine Erweiterung des Objektidentifikators, der die Instanz des Objektes identifiziert. Bei einzelnen Variablen ist dies eine „.0“, bei Tabellen ein *Index-Objekttyp*, welcher mehrere Zeileninstanzen derselben Spalte eindeutig kennzeichnet.

Diese Information muß in der jeweils notwendigen SNMP-PDU enthalten sein.

Umgekehrt müssen Operationen, die in der SNMP-Umgebung stattfinden, zum Manager hin repliziert werden.

- Wird in der SNMP-Umgebung ein Managed Object erzeugt (beispielsweise wenn ein Agent gestartet wird), so müssen entsprechende Schattenobjekte erzeugt werden. Analoges gilt, wenn Managed Objects gelöscht werden. Durch Exploration der SNMP-Umgebung soll das Gateway Agenten ausfindig machen und sich dynamisch an die Netzstruktur anpassen.
- Eine Ereignismeldung (eine `Trap-PDU`) von einem SNMP-Agent muß auf einen CORBA-Event abgebildet werden.

Hinweis: Da es in der SNMP-Umgebung möglich ist, Tabellenzeilen zu erzeugen und zu löschen (als “ungültig“ zu markieren), muß einem auch einem CORBA-Manager (und nicht nur dem Gateway) die Möglichkeit eingeräumt werden, Tabellenzeile-Objekte zu erzeugen und zu löschen — mit der entsprechenden Wirkung in der SNMP-Umgebung. Schattenobjekte können also sowohl vom Gateway als auch von der Managementanwendung erzeugt oder gelöscht werden. Die semantischen Unterschiede in Bezug auf Delete/Create-Operationen auf Schattenobjekten und deren Effekt in der SNMP-Umgebung sind in der Tabelle 4.1 zusammengefaßt.

Sonstige Forderungen an das Gateway

Insbesondere sind folgende Eigenschaften des Gateways wünschenswert:

- Der Zugriff auf SNMP-Ressourcen sollte möglichst effektiv und effizient sein. Requests an Schattenobjekte sollen quasiparallel bearbeitet werden und sich nicht gegenseitig blockieren, wenn SNMP-Operationen ausgeführt werden. Ein Synchronisationskonzept soll mögliche Verklemmungen verhindern.

Zustand	Aktion	Aufrufer	Wirkung auf Schattenobjekt	Wirkung auf Managementobjekt
M	S erzeugen	Gateway	erzeugt	keine
S, M	S löschen	Gateway	gelöscht	keine
–	S erzeugen	Manager	erzeugt	erzeugt
S, M	S löschen	Manager	gelöscht	gelöscht

S: Schattenobjekt existiert;

M: (entsprechendes) Managementobjekt existiert.

Tabelle 4.1: Delete/Create-Operationen auf Schattenobjekten

- Ein Sicherheitskonzept soll mindestens die von SNMP gewährte Sicherheit bezüglich des unbefugten Zugriffs garantieren.
- Das Gateway soll in Hinsicht seiner Funktionalität (z. B. Mitführen von Statistiken, Schwellwertüberwachung) unabhängig von den Schattenobjekten erweitert werden können.
- Die Funktionalität der Schattenobjekte soll unabhängig von anderen Schattenobjekten erweitert werden können. Hintergedanke ist die schrittweise Verlagerung von Managementaufgaben auf das Gateway/die Schattenobjekte und damit eine fortschreitende Entlastung für den Manager.

4.2 Die Komponenten des Gateways

Es liegt nahe, das Gateway in verschiedene Komponenten, die jeweils eine bestimmte Aufgabe erfüllen, zu zerlegen. Die Moduln des Gateways sind in Abbildung 4.2 dargestellt. Sie werden in der folgenden Aufstellung beschrieben:

Die **Schattenobjekte** sind die Instanzen der IDL-Klassendefinitionen, die zu einer Agenten MIB gehören. Sie liegen auf dem Gateway. Der Manager greift auf die Schattenobjekte zu, indem er die Methoden der Schattenobjekte mittels CORBA-Requests aufruft. Die Schattenobjekte antworten ihrerseits mit CORBA-Responses. Prinzipiell kann der Manager über die statische und die dynamische Aufrufschnittstelle Methoden der Schattenobjekte aufrufen. Die statische Aufrufschnittstelle ist allerdings nicht geeignet, da dazu die Client-Stubs aus der IDL-Beschreibung der Schattenobjekte zu den Client-Programmen der Managementanwendung dazugebunden werden müssen. Jedesmal, wenn neue Schattenobjektklassen hinzukommen und verwendet werden sollen, muß die Managementanwendung also neu übersetzt werden, ganz abgesehen von der Vielzahl der Client-Stubs, die letztendlich zum Manager dazugebunden werden müssen.

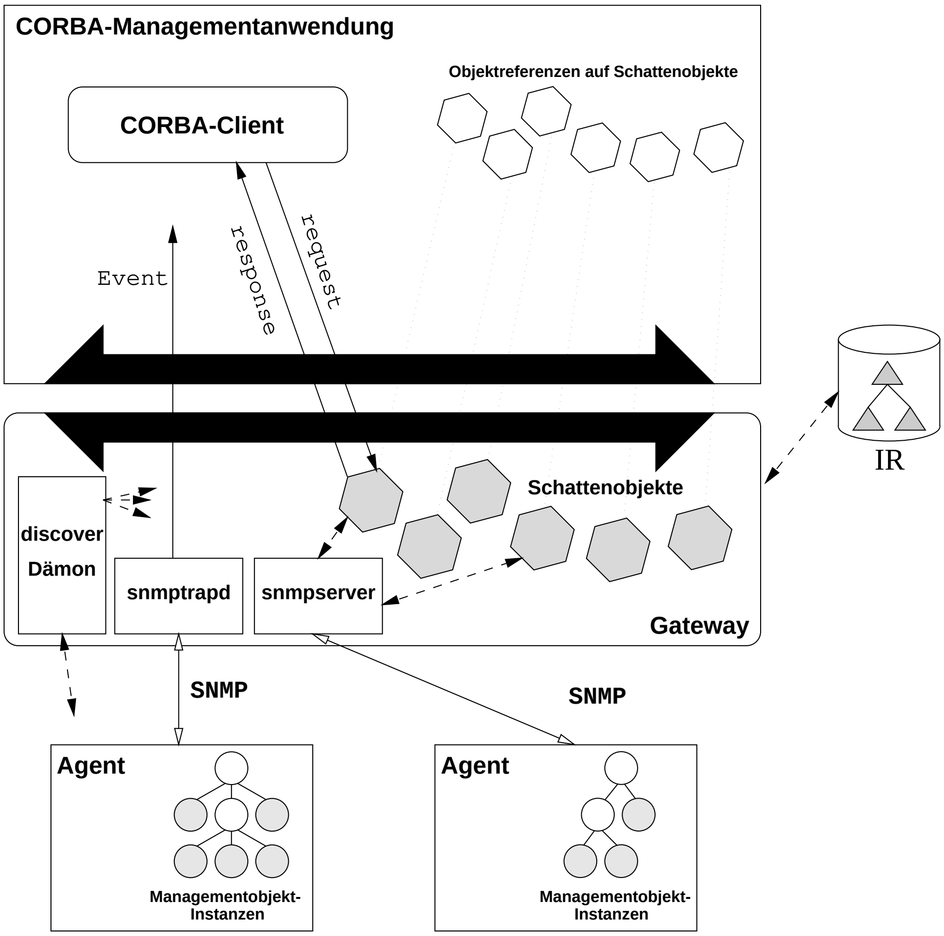

Abbildung 4.2: Ausgangslage

Für die dynamische Aufrufschnittstelle benötigt die Managementanwendung **Objektreferenzen auf die Schattenobjekte**, auf die zugegriffen werden soll. Solche Objektreferenzen werden dem Manager beispielsweise durch den Naming Service oder durch Events (etwa vom Gateway, wenn es ein Schattenobjekt erzeugt hat) zur Verfügung gestellt. Beim Methodenaufruf gibt der Manager in einem Request die Objektreferenz des adressierten Schattenobjektes, die aufzurufende Methode sowie dazugehörige Parameter an. Anhand der Objektreferenz lokalisiert der ORB das Schattenobjekt und ruft dort die gewünschte Methode auf. Die Requests werden immer genau an das Schattenobjekt weitergeleitet, das als Ziel angegeben wurde. Der ORB unterstützt nicht die Möglichkeit, beispielsweise einen Request für Objekt X an ein Objekt Y weiterzugeben, das die Aufgaben von X übernimmt.

Die von einem Sprachcompiler erzeugten IDL-Klassendefinitionen einer Agenten-MIB werden in das **Interface Repository** des ORB eingetragen (Abb. 4.3), also *nicht* direkt in die CORBA-Managementanwendung, wie es bei SNMP-Managern und SNMP-Plattformen notwendig ist. Für jede neu hinzukommende oder geänderte Agenten-MIB wiederholt sich dieser Vorgang. Laut der CORBA-Definition gibt es

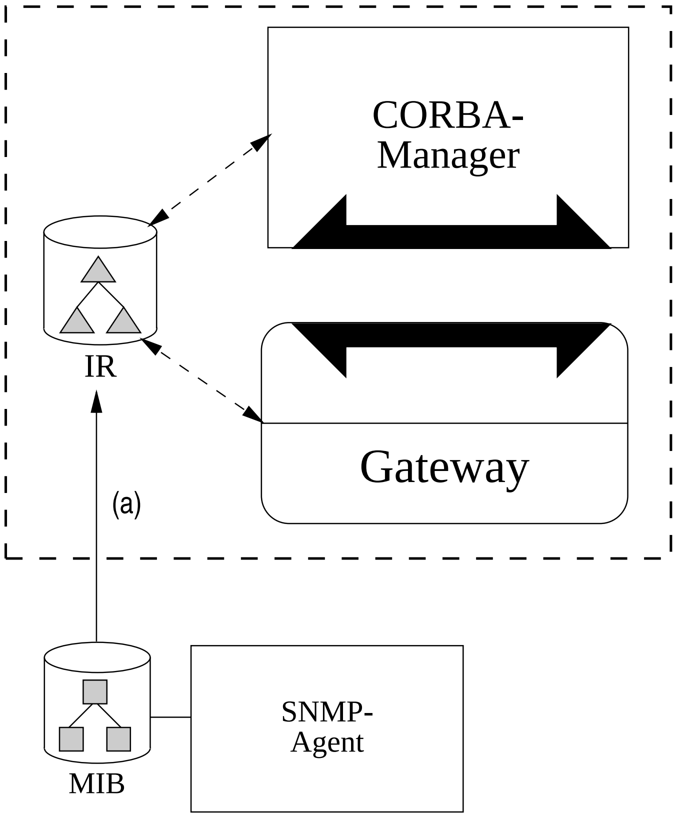

Abbildung 4.3: Einspielen der Agenten-MIB

in einer CORBA-Umgebung nur ein einziges (verteiltes) Interface Repository, auf das *alle* CORBA-Anwendungen Zugriff haben. Mit dem einmaligen Laden einer Agenten-MIB ins Interface Repository steht diese MIB (bzw. die entsprechenden Schnittstellendefinitionen) also automatisch sowohl der Managementanwendung als auch dem Gateway zur Verfügung.

Der **snmpserver** ist die Komponente für die Kommunikation mit der SNMP-Umgebung. Hauptaufgabe des *snmpserver* ist das Senden von SNMP-PDUs und das Empfangen der dazugehörigen Response-PDUs. Außerdem muß der *snmpserver* u. U. Response-PDUs (bzw. deren Inhalt) auf Schattenobjekte verteilen.

Der **snmptrapd** ist jenes Modul des Gateways, welche SNMP-Trap-PDUs von Agenten empfängt. Seine Aufgabe besteht darin, diese Traps auf CORBA-Events abzubilden. Zusammen mit dem *snmpserver* bildet der snmptrapd den in Abbildung 4.1 dargestellten SNMP-Manager. Die Komponente *snmptrapd* wird im Abschnitt 4.5 ausführlich behandelt.

Ein weiteres Gateway-Modul ist der **Discovery-Dämon**. Es soll zuständig sein für die Erfassung der einzelnen SNMP-Agenten im Internet (z. B. durch ein SNMP-Walk). Diese Komponente ist demzufolge auch zuständig für die Erzeugung von Schattenobjekten.

In den nun folgenden Abschnitten wird diskutiert, wie diese Komponenten kombiniert werden können. Die unterschiedlichen Varianten werden in Hinsicht auf Realisierbarkeit und Eignung bewertet, sodaß insgesamt ein Gateway entsteht, das die oben besprochenen Anforderungen erfüllt. In Kapitel 5 wird beschrieben, welche Möglichkeiten und Einschränkungen die Entwicklungsumgebung SOM/DSOM hat, um das entstandene Konzept zu implementieren.

4.2.1 Zustandsloses und zustandsbehaftetes Gateway

Ein Designkriterium für das Gateway ist die Frage, ob es zustandslos oder zustandsbehaftet sein soll. Im Rahmen dieser Diplomarbeit soll folgende Definition gelten:

Definition: Ein Gateway ist *zustandslos* (*stateless*), wenn im Gateway keine Werte von Variableninstanzen der SNMP-Agenten gespeichert werden. Andernfalls bezeichnet man das Gateway als *zustandsbehaftet* oder *stateful*.

Bei einem zustandslosen Gateway werden CORBA-Requests immer unmittelbar auf SNMP-Protokolleinheiten abgebildet. Für jeden Zugriff auf ein Schattenobjekt muß mindestens eine SNMP-PDU erzeugt und abgeschickt werden. Das Gateway hat die Funktionalität eines PDU-Konverters.

Bei einem zustandsbehafteten Gateway hingegen ist der Informationsaustausch zwischen CORBA-Manager und SNMP-Agent teilweise entkoppelt. Die Werte von Variablen werden in den Attributen der Schattenobjekte im Gateway gespeichert und können dem Manager bei einem Request unmittelbar zurückgegeben werden, also ohne daß SNMP-Agenten abgefragt werden müssen. Der Zugriff auf SNMP-Ressourcen wird dem Manager simuliert. Wie auch immer müssen die im Gateway (in den Shadow Objects) replizierten MIB-Inhalte (also die Werte der SNMP-Variablen) konsistent sein mit den tatsächlichen MIB-Werten der SNMP-Agenten.

Vorteile eines zustandsbehafteten Gateways sind kürzere Antwortzeiten aus Sicht des Managers und ein effektiverer Informationsaustausch zwischen dem Gateway und einem SNMP-Agenten dadurch, daß mehrere Managed Objects gleichzeitig und im voraus mit einer PDU abgefragt werden können. Außerdem können durch Vergleich der Information im Gateway und in der Agenten-MIB Änderungen festgestellt werden ([ACH93]). Je dynamischer die Information in den Agenten ist, desto häufiger muß die Information im Gateway allerdings aktualisiert werden. Ändern sich die Werte bestimmter SNMP-Variablen sehr oft (z. B. Zähler), ist es nicht möglich, die im Gateway replizierten Werte konsistent zu halten. Ein Gateway, in dem alle

SNMP-Werte gespeichert werden, ist aus diesem Grund nicht realisierbar. Eine realistische Lösung hingegen ist eine Mischung aus einem zustandsbehafteten und einem zustandslosen Gateway, in dem nur (nahezu) statische Werte repliziert werden (vgl. 4.6).

4.3 Zustandsloses Gateway

4.3.1 Zugriff mittels generischer Klassen

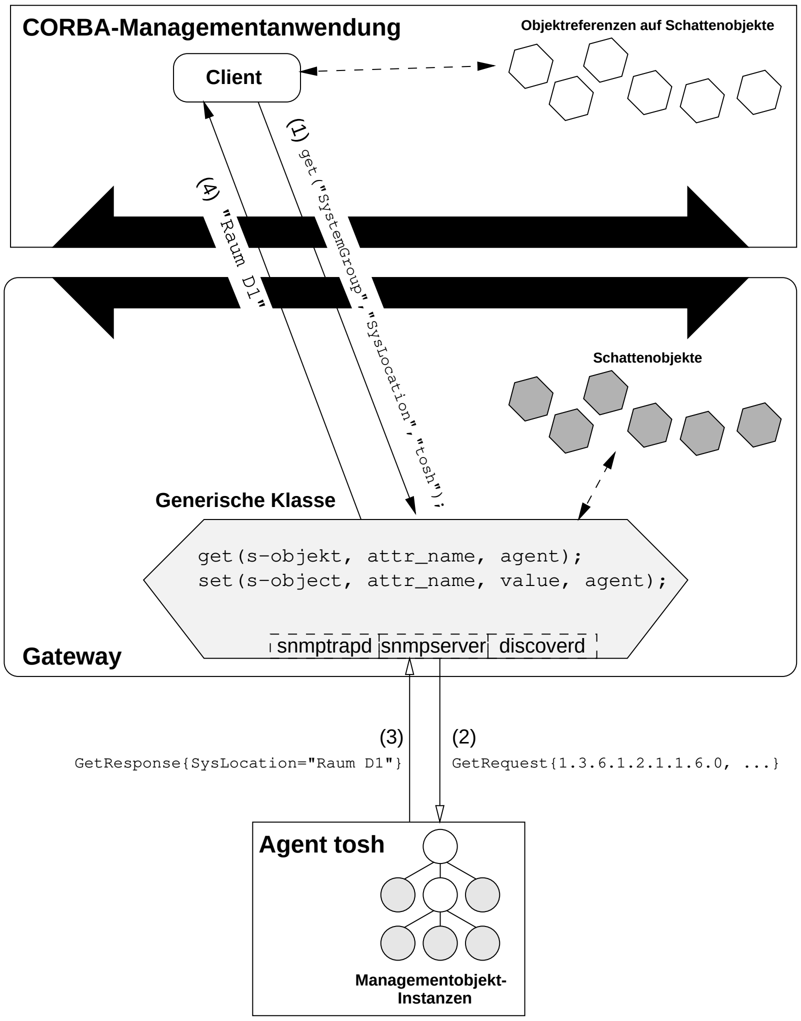

Abbildung 4.4: Zugriff mittels einer generischen Klasse

Im ersten Fall, der besprochen werden soll, erhält der Manager eine Objektreferenz auf eine Instanz einer generischen Klasse mit i. w. zwei Methoden:

- eine Methode `get()` für den lesenden Zugriff auf SNMP-Variablen; als Parameter gibt der Manager die Objektreferenz des Schattenobjektes, das Attribut, das gelesen werden soll, sowie die Adresse des Zielagenten an.
- eine Methode `set()` für den schreibenden Zugriff; Parameter dieser Methode sind die Objektreferenz des Schattenobjektes, das Attribut, der zu setzende, neue Wert und der Zielagent.

Die Schattenobjekte und die dazugehörigen Objektreferenzen werden von diesem Objekt zur Verfügung gestellt. Die `get()`-Methode erzeugt eine `GetRequest-PDU`, die `set()`-Methode eine `SetRequest-PDU`; das Objekt implementiert also gewissermaßen eine Dienstschnittstelle, mit der eine CORBA-Managementanwendung mit SNMP-Agenten kommunizieren kann. Es ist äquivalent mit dem in Abbildung 4.2 gezeigten *snmpserver*. Der Manager verwendet explizit diese Schnittstelle, wenn er auf SNMP-Ressourcen zugreifen will.

In der Abbildung 4.4 ist der Zugriff auf die Variable `SysLocation` des Agenten „tosh“ gezeigt. Der Manager benötigt hierzu den `get()`-„Dienst“. Er wird also die Methode `get()` mit den Parameterwerten “`SystemGroup`“ (eine Objektreferenz auf das Schattenobjekt “SystemGroup“), “`SysLocation`“ (das gewünschte Attribut) und “`tosh`“ (der Zielagent) aufrufen (1). Im Gateway muß daraufhin eine Abbildung des Attributnamens auf die SNMP-Variable stattfinden. Der Objektidentifikator und der Instanzidentifikator sind im adressierten Schattenobjekt gespeichert und werden vom generischen Objekt abgefragt. Schließlich wird eine `GetRequest-PDU` erzeugt und abgeschickt (2). Die angeforderte Information wird der vom Agenten kommenden `GetResponse` (3) entnommen und dem Aufrufer zurückgegeben (4).

Dieser Ansatz ist nicht geeignet, da eine Managementanwendung, statt die Attributzugriffsfunktionen der Schattenobjekte aufzurufen, die Objektreferenzen und Attribute explizit als Parameter dem Gateway übergeben muß. Die Tatsache, daß (über ein Gateway) auf SNMP-Ressourcen zugegriffen wird, ist für den Manager nicht transparent. Zu einem gegebenen Objekt (bzw. Objektreferenz) muß der Manager entscheiden, ob für den Zugriff ein Gateway-Dienst verwendet werden muß oder — bei „normalen“ CORBA-Objekten — direkt die Objektmethoden aufgerufen werden können.

Die zweite Alternative einer generischen Klasse basiert auf der Mehrfachvererbung. Bei der Übersetzung einer Agenten-MIB werden verschiedene Objektklassen erzeugt. Schließlich wird eine Klasse definiert, welche alle Attribute und Methoden der erzeugten IDL-Schnittstellen erbt. Eine Instanz dieser Klasse kann gewissermaßen als CORBA-Repräsentant eines SNMP-Agenten angesehen werden. In der Abbildung 4.5 wird eine solche Instanz — in Anlehnung an die Bezeichnung „Schattenobjekt“ — *Schattenagent* genannt. Ein Schattenagent integriert die Schnittstellen der Schattenobjektklassen und ein Kommunikationsmodul für den Informationsaustausch mit SNMP-Agenten (den *snmpserver*, s. o.). Für jeden SNMP-Agenten wird ein Schat-

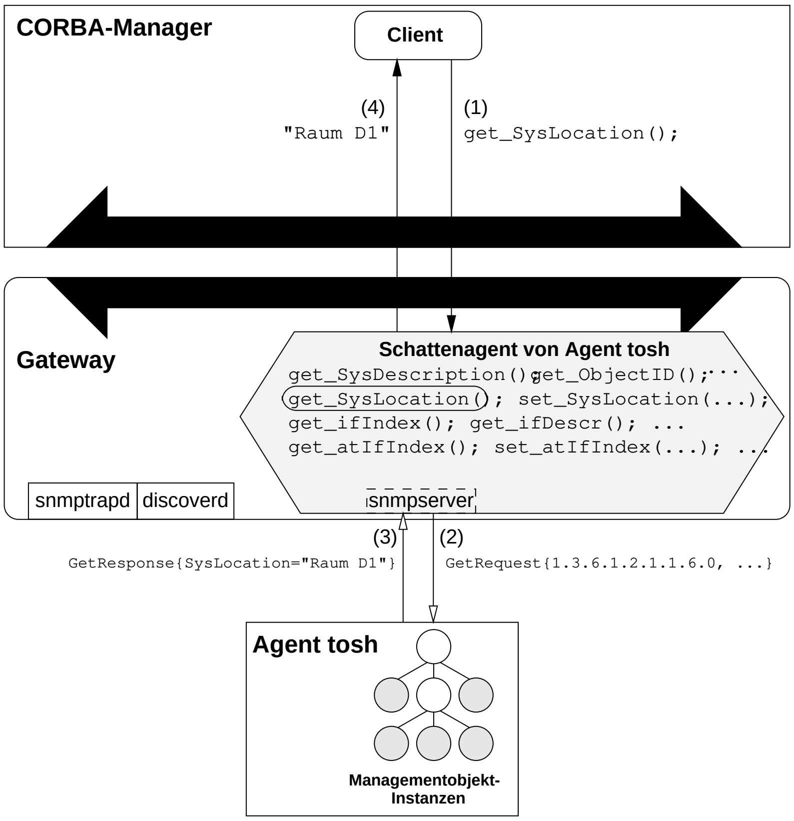

Abbildung 4.5: Zugriff auf einen Schattenagent

tenagent erzeugt, diese Aufgabe übernimmt ein eigener Prozeß im Gateway. Der Manager hat von jedem Schattenagent eine Objektreferenz. Zu jedem Attribut gehören eindeutige Methoden, die den Zugriff auf das Attribut ermöglichen. Ein Zugriff selbst erfolgt also durch den Aufruf dieser Methoden mittels eines CORBA-Requests.

Im Beispiel ruft die Managementanwendung die Methode `get_SysLocation()` auf (1). Der Schattenagent bildet den Attributnamen auf den SNMP-Objektidentifikator ab, identifiziert die zum Attribut gehörende SNMP-Instanz und versendet einen `GetRequest` (2). Mit (3) und (4) gelangt die angeforderte Managementinformation zuerst zum Schattenagent und dann zum Manager.

Der Gewinn im Vergleich zum obigen Verfahren ist, daß für die Managementanwendung nicht erkennbar ist, daß mit SNMP-Agenten kommuniziert wird. Der Ansatz hat aber folgende Nachteile:

- Die kleinste verwaltbare (Objekt)-Einheit aus Sicht des Managers ist ein Schattenagent, ein in der Regel sehr großes Objekt.

- Die Problematik beim Zugriff auf SNMP-Tabellen: Für alle SNMP-Zeileninstanzen derselben Tabellenspalte existiert nur eine einzige get()- bzw. set()-Methode. Allein die Kenntnis der aufgerufenen Attributzugriffsfunktion reicht dem Gateway aber nicht aus, um die SNMP-Instanz zu identifizieren. Der Schattenagent kann beispielsweise bei einem Aufruf der Methode get_ifDescr()[1] nicht feststellen, welche Zeileninstanz der Manager lesen möchte. Umgekehrt besteht für den Manager auch nicht die Möglichkeit, dem Schattenagent die Zeile, auf die zugegriffen werden soll, zu übergeben.

4.3.2 Erzeugung der SNMP-PDUs innerhalb der Schattenobjekte

Ein nächster Ansatz ist, die Schattenobjekte derart zu erweitern, daß jedes *für sich* mit einem Agenten in der SNMP-Umgebung kommunizieren kann. Jedes Schattenobjekt ist also in der Lage, SNMP-PDUs zu verschicken und zu empfangen, und übernimmt damit selbst die Rolle eines (Protokoll-)Gateways. Ein eigener *snmpserver* ist in jedem Schattenobjekt enthalten.

Der Zugriff auf einen Agenten ist in Abbildung 4.6 dargestellt. Im Beispiel wird angenommen, daß der Manager die Variable `SysLocation` eines Agenten lesen möchte. Für das zum Agenten gehörende Schattenobjekt SystemGroup mit dem entsprechenden Attribut besitzt der Manager eine Objektreferenz. Der Manager erzeugt einen Request, in dem er das Schattenobjekt als Zielobjekt und die Attributzugriffsfunktion get_SysLocation() als gewünschte Operation auf diesem Zielobjekt angibt (1). Dieser Request bewirkt einen Aufruf der Methode get_SysLocation des Schattenobjektes. Das Schattenobjekt berechnet daraufhin alle notwendigen Parameter für eine `GetRequest-PDU` (Objekt- und Zugriffsidentifikator des Managementobjektes, Host) und verschickt schließlich eine SNMP-Message, welche die erzeugte PDU enthält (2).

Nachdem die SNMP-Message verschickt wurde, wartet der Schattenobjekt-Prozeß auf die Antwort des Zielagenten. Sofern diese eintrifft (3), untersucht das Schattenobjekt die `GetResponse-PDU` auf Fehler und gibt die benötigte Information dem Aufrufer auf der Managerseite zurück (4). Der schreibende Zugriff läuft analog ab, es wird allerdings kein Wert zurückgegeben.

Für die Erzeugung der Schattenobjekte ist der *Discovery-Dämon* zuständig (s. Abbildung 4.7). Sobald dieser ein Managementobjekt „findet“ (1), sucht er im Interface Repository anhand des Objektidentifikators der gefundenen Variable eine passende Klasse. Wenn das Managed Object noch von keinem Attribut aller schon existierenden Instanzen dieser Klasse repräsentiert wird, wird die Methode Create() der entsprechenden Factory[2] aufgerufen (2), wodurch ein Schattenobjekt erzeugt wird

[1] `ifDescr` ist das zweite Spaltenelement der SNMP-Tabelle `ifTable`

[2] Eine Factory ist ein Objekt mit dem eine Instanz mit einer bestimmten IDL-Schnittstelle

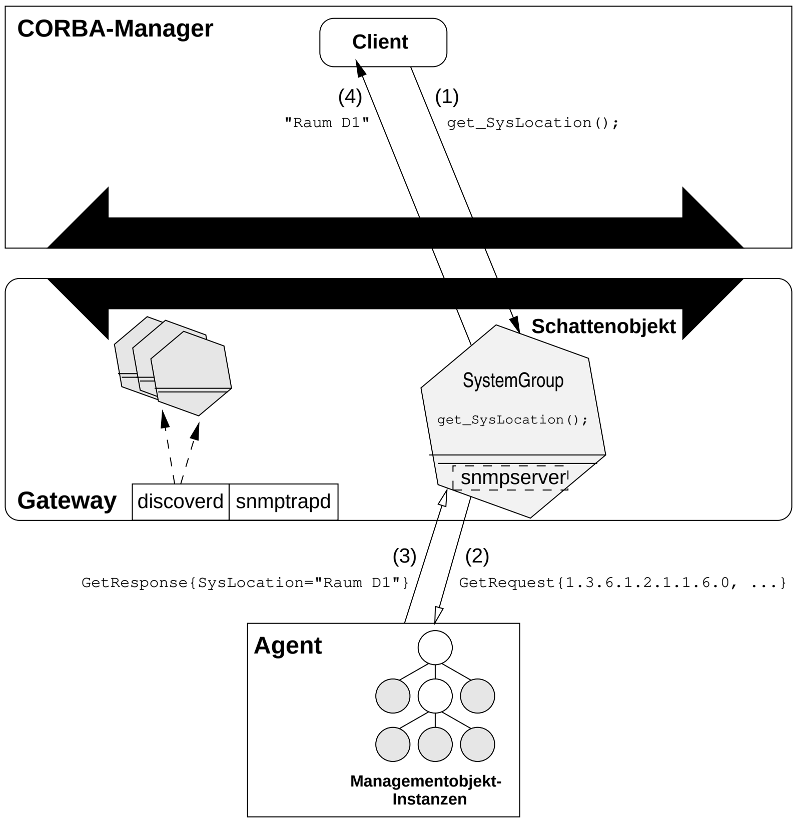

Abbildung 4.6: Schattenobjekte mit Gatewayfunktionalität

(3). Bei der Initialisierung (4) des neuen Schattenobjektes werden die Adresse des Agenten und Zugriffsidentifikator (bei Gruppenobjekten also „.0“, bei Tabellenzeile-Objekten der Identifikator der Zeile, vgl. 4.1). Mit einem Event wird die Erzeugung (und sinnvollerweise gleich die Objektreferenz) des Schattenobjektes dem Manager mitgeteilt (5).

Bewertung

Dieser Ansatz hat folgende Nachteile

- Die Schattenobjekte sind sehr komplex. Jedes Schattenobjekt stellt im Prinzip einen einfachen SNMP-Manager dar, der nur für wenige Managementobjekte eines einzigen Agenten zuständig ist.

erzeugt werden kann. Factories können durch den CORBA LifeCycle Service gefunden werden, s. [OMG96], 2.3.1.

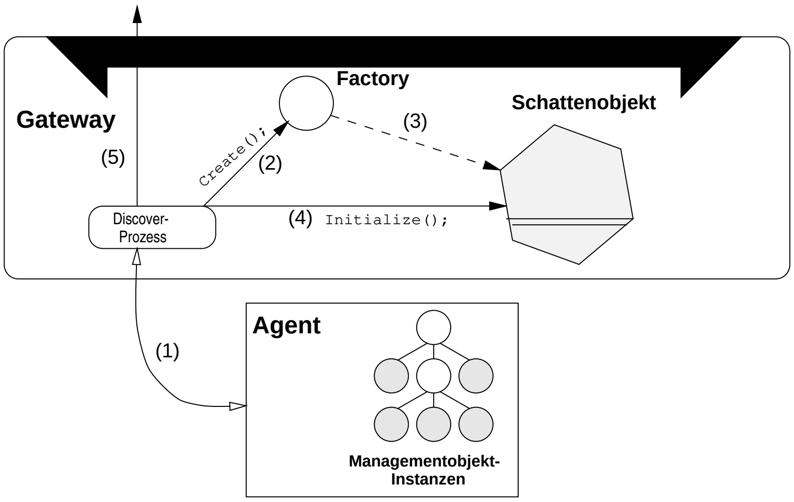

Abbildung 4.7: Erzeugung von Schattenobjekten

- Eine SNMP-GetResponse kann ausschließlich anhand der in ihr enthaltenen `request-id` dem ursprünglichen SNMP-Request zugeordnet werden[3]. Da viele Schattenobjekt-Prozesse unabhängig SNMP-PDUs verschicken, ist die Erzeugung notwendigerweise eindeutiger `request-ids` problematisch.

- Die PDUs `GetNextRequest` und `GetBulk` können von den Schattenobjekten, auch wenn sie von der SNMP-API angeboten werden, nicht sinnvoll verwendet werden, da damit vom Zielagenten unter Umständen Werte von Managed Objects zurückgeschickt werden, die nicht vom Schattenobjekt repräsentiert werden. Während z. B. ein Schattenobjekt, das die *i*-te Zeile einer SNMP-Tabelle darstellt, mit einer `GetRequest-PDU` nur Werte von Tabelleninstanzen der *i*-ten Zeile erhält, werden bei einer `GetNextRequest-PDU` mit den gleichen Parametern die Instanzen der *(i+1)*-ten Tabellenzeile zurückgegeben.

- Die Erweiterung der Schattenobjekte, beispielsweise die Einführung von einer neuen API ist nur durch Verändern (neu Übersetzen) aller betroffenen Schattenobjekte möglich.

- Oftmals ist erwünscht, daß ein Gateway Schnittstellen zu Verfügung stellt, durch die das Gateway selbst überwacht werden kann. Bei diesem Ansatz müßten globale, nicht das Schattenobjekt betreffende Daten für Statistiken, wie etwa die Anzahl der fehlerhaft empfangenen SNMP-Messages, in jedem Schattenobjekt gespeichert und aktualisiert werden.

[3] Die `request-ids` zusammengehörender PDUs stimmen überein (vgl. [CMRW96b]).

4.3.3 Erzeugung der SNMP-PDUs außerhalb der Schattenobjekte

Die Mängel des vorhergehenden Ansatzes haben gezeigt, daß die Komponente, welche für die Kommunikation mit Agenten in der SNMP-Umgebung benötigt wird, von den Schattenobjekten getrennt werden muß. Bei dem nun folgenden Ansatz

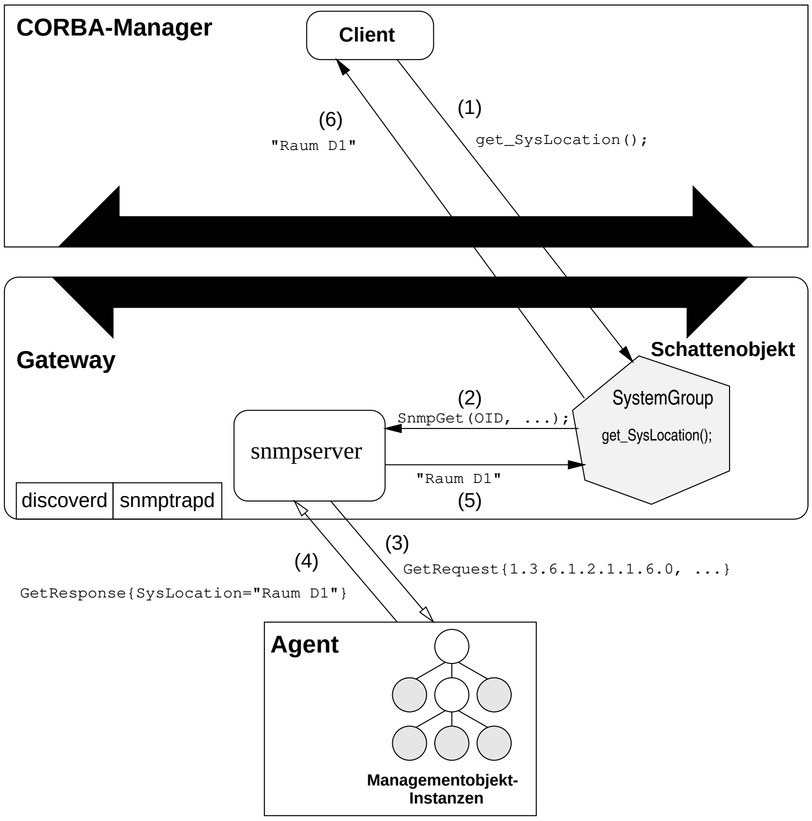

Abbildung 4.8: Informationsaustausch über snmpserver-Prozeß

wird also der *snmpserver* aus den Schattenobjekten herausgelöst und übernimmt als *eigener* Prozeß die Abbildung zwischen den Protokollwelten.

Die Schattenobjekte besitzen nun nicht mehr die Funktionalität eines Gateways. Deren Erzeugung erfolgt analog zur vorherigen Ansatz. Der *snmpserver* bietet den Schattenobjekten (und nicht dem Manager, wie in 4.3.1) zwei Methoden an:

- eine Methode `SnmpSet()`, welche eine `SetRequest-PDU` erzeugt und versendet
- eine Methode `SnmpGet()`, welche (je nach Bedarf) eine `GetRequest-`, `GetNext-Request-` oder eine `GetBulk-PDU` erzeugt und ebenso abschickt.

Der *snmpserver* hat außerdem die Aufgabe, SNMP-Messages bzw. `Response-PDUs` zu empfangen und den aufrufenden Schattenobjekten die angeforderte Information zurückzugeben.

Der Informationsaustausch zwischen Manager und Agent ist in Abbildung 4.8, wieder am Beispiel von der Variable `SysLocation`, dargestellt. Wiederum ruft der Manager mittels der Objektreferenz die Zugriffsmethode Get_SysLocation des Schattenobjektes auf (1). Das Schattenobjekt schickt daraufhin einen CORBA-Request an den *snmpserver* (2). Da der snmpserver nicht feststellen kann, welches Schattenobjekt der Aufrufer ist[4], werden Objekt-, Zugriffsidentifikator des Managementobjektes und die Adresse des Zielagenten als Parameter vom Schattenobjekt dem Gateway übergeben. Die aufgerufene Server-Methode SnmpGet() erzeugt damit eine `GetRequest-PDU` und sendet diese an den Zielagenten. Trifft eine `GetResponse-PDU` von Seiten des Zielagenten beim Gateway ein (4), so kann das Gateway diese anhand der `request-id` der ursprünglichen `GetRequest-PDU` zuordnen und damit dem Schattenobjekt, für das letztere verschickt wurde, zurückgeben (5). Das Schattenobjekt reicht die angeforderte Information schließlich dem Manager weiter (Rückkehr aus dem Methodenaufruf, Abb. 4.8 (6)).

4.4 Zustandsbehaftetes Gateway

Eine weitere Möglichkeit ist, die Kommunikation zwischen dem Manager und den Agenten zu entkoppeln. Während bei den vorherigen Fällen jeder Aufruf einer Attributzugriffsfunktion auf eine SNMP-PDU abgebildet wird, soll nun der Fall betrachtet werden, bei dem die Kommunikation von Manager mit den Schattenobjekten einerseits und die Kommunikation von Gateway mit den Managementobjekten andererseits und zwar *unabhängig* stattfinden.

Bei einem *zustandsbehafteten* Gateway werden Werte der Agenten-MIBs im Gateway repliziert. Gespeichert werden diese Werte in den Attributen der Schattenobjekte und müssen regelmäßig aktualisiert werden, damit sie zum Zeitpunkt des Zugriffs von Seiten des Managers mit den tatsächlichen Werten in den entsprechenden Managed Objects konsistent sind. Diese *virtuelle* Konsistenz setzt die Managementanwendung voraus; mit welcher Strategie diese gewährt wird, ist für die Managementanwendung irrelevant.

Es können zwei Möglichkeiten unterschieden werden:

- Jedes Schattenobjekt sorgt selbst dafür, daß seine Information aktuell ist
- Ein ausgezeichneter Prozeß im Gateway ist für die Konsistenz der Information

[4] Er kann nur die UserID des CORBA-Requests und den Host, von dem dieser kommt feststellen, jedoch nicht den Prozeß und die Routine, die den Request erzeugt haben.

aller Schattenobjekte zuständig.

Im ersten Fall stellt kann man sich jedes Schattenobjekt als einen einzigen Prozeß vorstellen, der regelmäßig aktiv wird, um die eigenen Attributwerte mit den dazugehörigen Werten in der SNMP-Umgebung abzugleichen — auch wenn niemals auf die Attribute des Schattenobjektes zugegriffen wird. Wie oft das „Update" geschieht, hängt von der Implementierung des Schattenobjektes ab. Idealerweise werden bei der Implementierung Informationen über das dynamische Verhalten der SNMP-Managementobjekte berücksichtigt.

Die bisher besprochenen zustandslosen Gateways in 4.3.2 und 4.3.3 können zu einem zustandsbehafteten Gateway ausgebaut werden, indem die Schattenobjekte selbst die Funktionalität der Konsistenzerhaltung übernehmen. Die Schritte (2)–(3) in der Abbildung 4.6 bzw. die Schritte (2)–(5) in Abbildung 4.8 (Informationsaustausch mit SNMP-Agent) folgen dann nicht notwendigerweise unmittelbar bei einem Zugriff auf das Schattenobjekt, sondern möglicherweise zeitlich versetzt.

Die Abbildung 4.9 zeigt den zweiten Fall. Wieder will ein Manager, der den Wert des Attributs SysLocation lesen und ruft deshalb, wie schon beschrieben, die Attributzugriffsfunktion *Get_SysLocation* auf (1). Nun gibt aber das Schattenobjekt den Wert des Attributs (der notwendigerweise im Schattenobjekt gespeichert sein muß) unmittelbar an den Manager zurück (2). Bei dieser Aktion wird *keine* SNMP-PDU erzeugt.

Da die Schattenobjekte die Attributwerte nicht selbst mit den Werten der entsprechenden Managed Objects konsistent halten, müssen diese Werte regelmäßig vom Gateway aktualisiert werden. Zu diesem Zweck fragt das Gateway (im einfachsten Fall in festen Zeitintervallen) die Managementinformation von den Agenten ab (3, 4) und schreibt diese in das zugehörige Attribut des Schattenobjektes (5). Dazu ist eine neue Methode im Schattenobjekt notwendig, da:

- auf die Attribute eines Objektes prinzipiell nur mit objekteigenen Methoden zugegriffen werden kann,

- nicht jedes Attribut eines Schattenobjektes eine Set-Operation zur Verfügung stellt, die statt dieser Methode verwendet werden könnte und,

- je nachdem, ob der Manager oder das Gateway die Set-Operation des Schattenobjektes aufruft, unterschiedlich reagiert werden muß. Der Set-Aufruf des Managers muß auf eine Set-Operation auf das entsprechende Managementobjekt in der SNMP-Umgebung repliziert werden, während beim Gateway das Attribut des Schattenobjektes aktualisiert (und keine PDU verschickt) werden soll. Problem dabei ist, daß das Schattenobjekt nicht feststellen kann, wer der Aufrufer der Set-Operation ist.

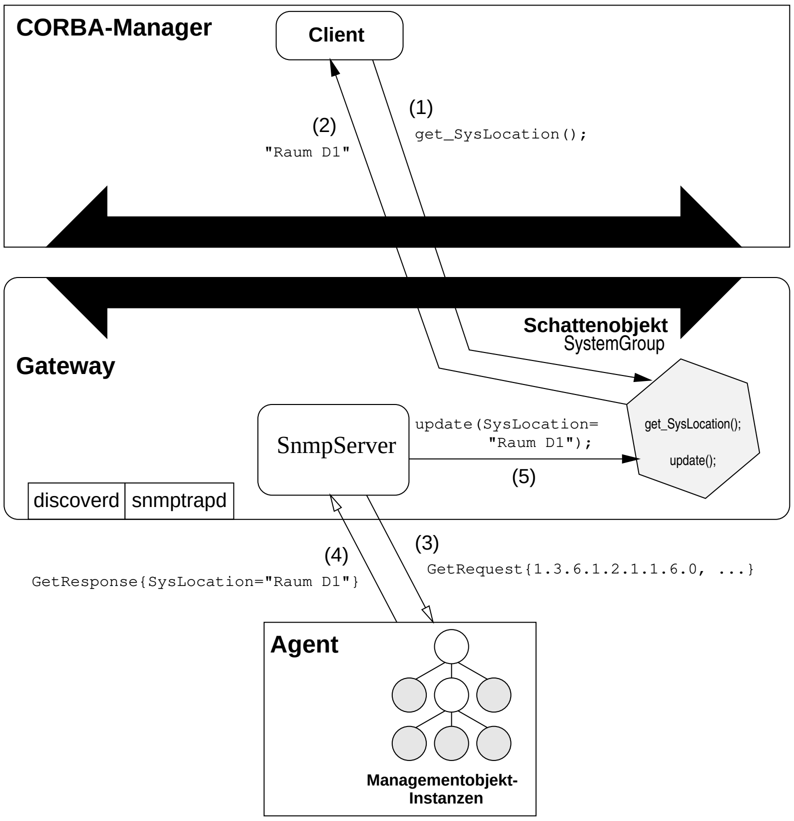

Abbildung 4.9: Entkoppelter Informationsaustausch

Alternativ dazu kann das Gateway ein Schattenobjekt auffordern, sich selbst zu aktualisieren. Auch bei dieser Variante entscheidet das Gateway über den Zeitpunkt des Abgleichs.

Eine weitere Frage ist, wie oft ein derartiges Update erfolgen soll. Je seltener das Gateway die Information in den Schattenobjekten mit der in den Managementobjekten vergleicht, desto größer ist die Gefahr, daß bei einem Zugriff auf Schattenobjekte von Seiten des Managers die Information nicht konsistent ist. Da die Anzahl der Schattenobjekte in der Regel sehr hoch ist, ist andererseits ein häufiges Aktualisieren sehr aufwendig und (deshalb) nur mit einer nach oben begrenzten Frequenz möglich. Wenn sich die Managementinformation in den Agenten kontinuierlich ändert, wie es z. B. bei Zählervariablen der Fall ist, ist ein gewisser Inkonsistenzgrad in den Schattenobjekten nicht vermeidbar.

Umgekehrt kann es sein, daß ein Manager ein Attribut mit einem neuen Wert belegt, beispielsweise das Attribut SysLocation durch einen Aufruf von Set_SysLocation() wie in Abbildung 4.10(1) gezeigt. In diesem Fall muß das entsprechende Managementobjekt aktualisiert werden (s. 4.1, Replikation von Operationen). Da Managementinformation auf der CORBA-Seite ausschließlich in den Schattenobjekten gespeichert wird, kann das Gateway bei einem Vergleich der entsprechenden SNMP-Werte nicht feststellen, welche Information gültig ist. Deshalb ist es notwendig, daß das Schattenobjekt dem Gateway meldet, eines seiner Attribute verändert wurde . Eine Möglichkeit ist, dem Gateway einen Event zu schicken (Abb. 4.10 (2a)). Das Gateway verschickt daraufhin die notwendige SNMP-PDU (3). In Anlehnung an ein zustandsloses Gateway (Abb. 4.8), kann das Schattenobjekt auch selbst die Initiative ergreifen und explizit die Erzeugung einer SNMP-PDU (durch eine Aufruf einer entsprechenden Gateway-Methode, 2b) veranlassen.

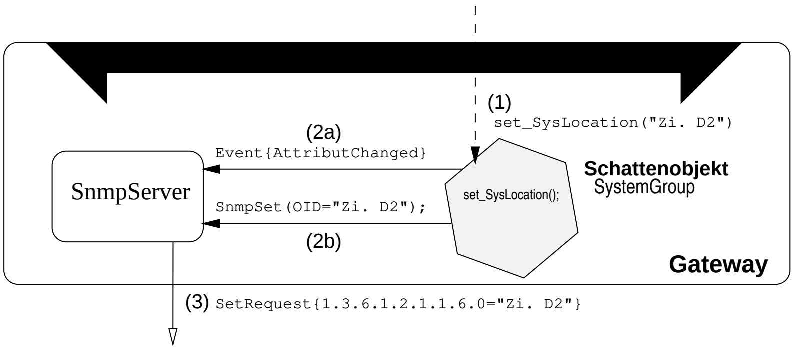

Abbildung 4.10: Schreibender Zugriff auf Attribut

4.5 Abbildung von SNMP-Ereignismeldungen

In diesem Abschnitt werden nun Traps besprochen, welche von Agenten an das Gateway geschickt werden und auf CORBA-Events abgebildet werden müssen. Das Konzept zur Lösung dieser Anforderung beruht auf der Annahme, daß das Erzeugen und Versenden von CORBA-Ereignismeldungen gemäß des CORBA-Event-Service ([OMG96]) erfolgt. Eine Zusammenfassung des Event Service ist 2.3.1 gegeben.

Daß die Komponente *snmptrapd* erst in diesem Abschnitt beleuchtet wird, hat folgende Gründe:

- Technisch bedingt kann immer nur ein Prozeß gleichzeitig an dem Port 162[5] auf SNMP-Traps warten. Dementsprechend erfüllt der snmptrapd-Dämon eine sozusagen einzigartige Aufgabe.

[5] i. e. das well-known Port, an das SNMP-Agenten ihre Ereignismeldungen schicken.

- Andererseits ist denkbar, daß ein Gateway mehrere Dämonen hat, welche jeweils auf einem Rechner SNMP-Traps empfangen. Jeder Dämon kann dann als eine Einheit angesehen werden, welcher von den restlichen Gatewaykomponenten weitgehend unabhängig ist.

Beim Internet-Management entspricht eine Ereignismeldung einer Protokolldateneinheit (SNMP-Trap), die von einem Agenten erzeugt und an einen SNMP-Manager geschickt wird. Im Gegensatz dazu entsprechen CORBA-Events Funktionsaufrufen. Objekte, welche einen Event an ein anderes Objekt schicken wollen, rufen eine bestimmte Methode des Zielobjektes auf. Das Zielobjekt muß diese Methode und damit den Event unterstützen. Das Analog zur Ereignismeldung zwischen einem Agent und einem SNMP-Manager besteht also darin, daß ein „Stellvertreter“ des SNMP-Agenten eine Methode des CORBA-Managers aufruft. Zwischen diesen unterschiedlichen „Ereignismeldungen“ soll nun eine geeignete Abbildung stattfinden.

4.5.1 Abbildung von SNMP-Traps auf generische CORBA-Events

Die einfachste Möglichkeit, wie SNMP-Traps abgearbeitet werden können ist, einen SNMP-Trap unbearbeitet in einen generischen CORBA-Event zu packen und direkt an den Manager zu schicken (siehe Abbildung 4.11). Ein Dämon `snmptrapd` am well-known Port 162 empfängt demzufolge eine Trap-PDU und ruft als PushSupplier die Methode `push()` eines PushEventConsumers des Managers auf (2a). Dabei wird die Trap-PDU als Parameter (Datentyp `any`) übergeben. Beim *pull*-Modell ruft der PullEventConsumer des Managers die Methode `pull()` des Trap-Dämons im Gateway auf. Ihm wird dann die Trap-PDU zurückgegeben. Diese Methode entspricht dem Polling

Im Prinzip werden bei dieser Vorgehensweise Ereignismeldungen „direkt an den Manager“ geschickt. Das Fehlen eines Event Channel Objektes — laut des CORBA-Event-Dienstes durchaus legitim — hat aber den Nachteil, daß es für das Gateway nicht transparent ist, wer der Empfänger der Events ist. Es benötigt nämlich für jeden Manager (Consumer) eine Objektreferenz und muß an jeden Manager jeden Event einzeln verschicken. Für einen Event Channel braucht das Gateway hingegen nur eine Objektreferenz; beliebige Consumer können sich an den Event Channel anschließen, um Ereignismeldungen zu empfangen, ohne daß dies für einen Supplier im allgemeinen und das Gateway im einzelnen Konsequenzen hat.

Abbildung 4.12 stellt eine Lösung mit einem generischen Event Channel dar (*push*-Modell). Das Gateway hat einen Event Channel erzeugt (beispielsweise bei der Initialisierung oder beim Eintreffen des ersten SNMP-Traps). Sobald ein SNMP-Trap empfangen wird, ruft der snmptrapd die Methode `push()` des Event Channels auf und schiebt die Daten der Ereignismeldung in den Event Channel. Nun ist es Aufgabe

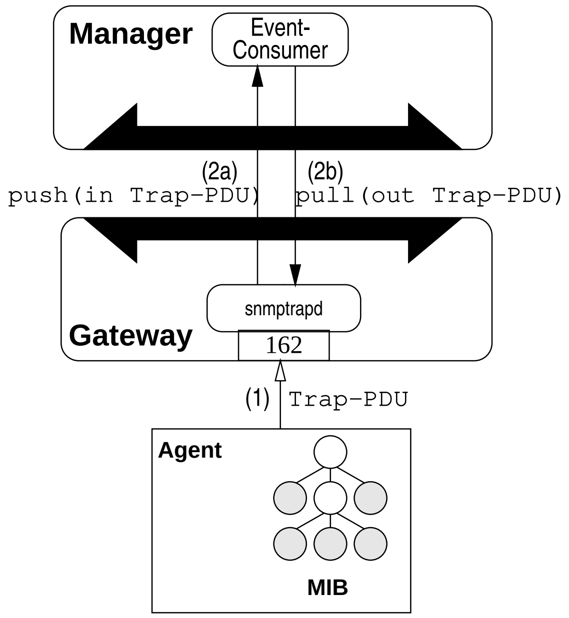

Abbildung 4.11: Traps als generische Events direkt an Manager

dieses Event Channels, die Ereignismeldung an alle Manager (Consumer) weiterzuleiten (3), die bei diesem Event Channel registriert sind.

Generische Events haben allerdings folgende Nachteile, sodaß das Modell der typisierten Kommunikation zwischen Manager und Gateway geeigneter ist:

- Die Methoden push() und pull() besitzen jeweils nur einen Parameter von Datentyp any. Damit muß ein Manager beispielsweise als PushConsumer alle Daten akzeptieren und interpretieren, die ihm beim Aufruf seiner push-Methode übergeben werden.
- Die ursprüngliche Herkunft der Ereignismeldungen kann nicht bestimmt werden. Beispielsweise kann nicht erkannt werden, ob ein Event direkt von einer CORBA-Umgebung oder (eigentlich) von einer SNMP-Domäne stammt.
- Vor der Verarbeitung des Events ist nicht feststellbar, welcher SNMP-Agent den Trap verschickt hat.
- Unterschiedliche Typen von SNMP-Traps (z.B. coldStart, warmStart, authenticationFailure ...) können nicht abgegrenzt werden.

Insgesamt sind die generischen *push*- und *pull*- Methoden nicht differenziert genug, um im Event Channel Ereignismeldungen sinnvoll zu filtern.

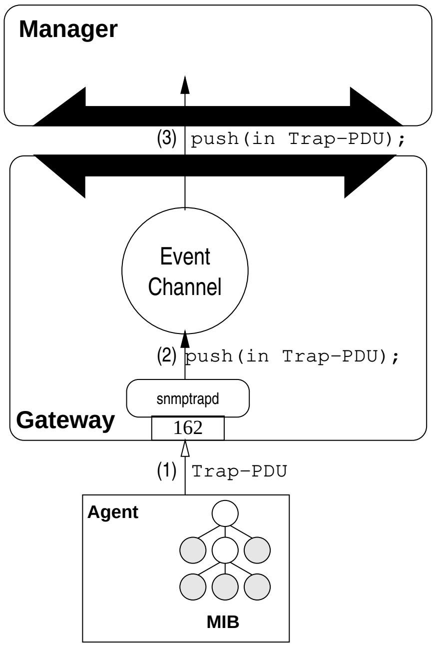

Abbildung 4.12: Traps als generische Events an Event Channel

4.5.2 Abbildung von SNMP-Traps auf typisierte CORBA-Events

Bei der Abbildung von SNMP-Traps auf typisierte CORBA-Events (im folgenden *TypedEvents* genannt), gibt es generell zwei Möglichkeiten:

- Es wird ein einziger TypedEvent definiert, auf den alle SNMP-Traps abgebildet werden;

- Jeder SNMP-Trap wird auf einen eigenen TypedEvent abgebildet.

Eine dritte Form der Abbildung besteht in einer Mischung der beiden angeführten Möglichkeiten. Es ist denkbar, daß eine Menge von SNMP-Traps auf einen einzigen TypedEvent abgebildet wird (z. B. die Menge der „unbekannten" SNMP-Traps), während eine zur ersteren disjunkte Menge von SNMP-Traps auf eine Menge von TypedEvents abgebildet wird. Da sich dieser Fall auf die angesprochen (Teil-)Fälle zurückführen läßt, wird nicht näher darauf eingegangen.

Den folgenden Beschreibungen wird das *push*-Modell zugrundegelegt. Für das *pull*-Modell gelten aber die Ausführungen i. w. genauso.

Definition eines TypedEvents für alle SNMP-Traps

Wenn SNMP-Traps auf einen einzigen TypedEvent abgebildet werden soll, bedeutet dies, daß bei Empfang einer Trap-PDU immer dieselbe, vorher festgelegte Methode aufgerufen wird. Im Modul SNMPMgmt (s. 3.1.1, [JID95]) wird dazu eine Schnittstellendefinition GenericNotification mit der „push"-Methode snmp_notification() zur Verfügung gestellt. Das Gateway meldet sich bei einem TypedEventChannel als Supplier an (1) und erhält eine Objektreferenz auf ein Objekt des Interface GenericNotfication. Trifft nun z. B. eine Trap-PDU mit dem Trap-Typ `linkUp` (OID = "... SnmpTraps.4") ein (3), so muß das Gateway lediglich die Methode snmp_notification aufrufen. In den Parametern dieser Methode werden die Informationen über den SNMP-Trap festgehalten. (Beim generischen Event Channel hatte die *push*-Methode nur einen Parameter vom Typ any.) Der TypedEventChannel sorgt dafür, daß die Ereignismeldung an alle bei ihm registrierten Consumer (insbesondere also Manager) weiterverschickt wird. Er ruft dazu wiederum Methoden der Consumer auf. Wenn ein Manager einen snmp_notification-Event empfangen will, muß er also die Schnittstelle GenericNotification unterstützen und bei der Registrierung dem TypedEventChannel mitteilen.

Dieser Ansatz hat folgende Eigenschaften:

- Die Realisierung ist einfach, das Gateway muß immer nur dieselbe Methode aufrufen.
- Ein Managementanwendung kann feststellen, daß der Event ursprünglich aus einer SNMP-Domäne stammt.
- Nach wie vor kann nur mittelbar (aus dem Inhalt des Events) festgestellt werden, von welchem Agenten der Trap ursprünglich herkommt.
- Die Ereignismeldung ist nicht differenziert genug, um unterschiedliche Typen von SNMP-Traps zu erkennen, so daß diese beispielsweise speziellen Consumern zugeteilt werden kann. Erst bei der Verarbeitung des Events kann entsprechend verschieden reagiert werden.

Definition eines TypedEvents für jeden SNMP-Trap

Bei dieser Variante gibt es für jeden SNMP-Trap eine Methode, d. h. einen eigenen Event. Die Methoden werden bei der Übersetzung einer Agenten-MIB in IDL-Schnittstellen definiert. In jedem IDL-Modul existieren zwei Schnittstellen (SnmpNotification und PullSnmpNotification), deren Methoden die den in der dazugehörigen Agenten-MIB definierten Ereignismeldungen entsprechen (s. 3.1.1, [JID95]). Aufgabe des Gateways ist es, zur Laufzeit von Fall zu Fall zu entscheiden, welche Methode von welcher Schnittstelle für den aktuellen Trap aufgerufen werden muß.

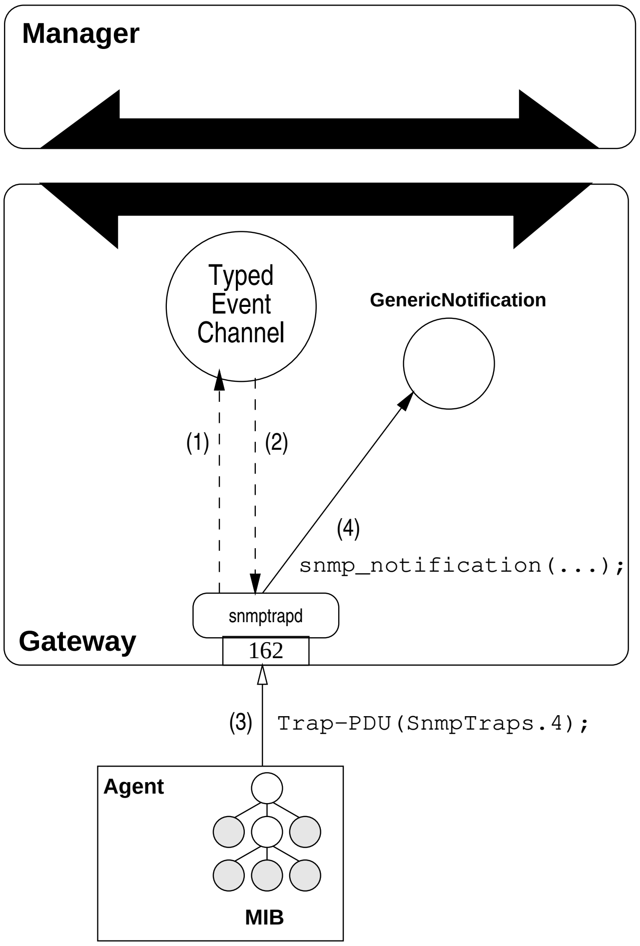

Abbildung 4.13: Alle SNMP-Traps werden auf ein TypedEvent abgebildet

Beispiel: Wie vorher treffe der SNMP-Trap `linkUp` vom SNMP-Agent *tosh* beim snmptrapd des Gateways ein (Abbildung 4.14). In der folgenden resolve-Funktion (2) muß das zur Agenten-MIB von *tosh* gehörende IDL-Modul (im Interface Repository) und innerhalb dieses Moduls die Schnittstelle gefunden werden, welche die Methode (linkup()) zu dem Trap unterstützt. Die Methode resolve liefert dieses Interface zurück. Bei der Anmeldung beim TypedEventChannel ((3) und (4), sofern nicht schon stattgefunden) wird dieses Interface angegeben und vom TypedEventChannel eine Objektreferenz auf ein Objekt dieser Klasse zurückgegeben. Das Gateway ruft anschließend die Methode linkUp() dieses Objektes auf (5). Eine Managementanwendung kann gezielt Events eines SNMP-Agenten empfangen, in dem sie die dazugehörige SnmpNotification-Schnittstelle unterstützt.

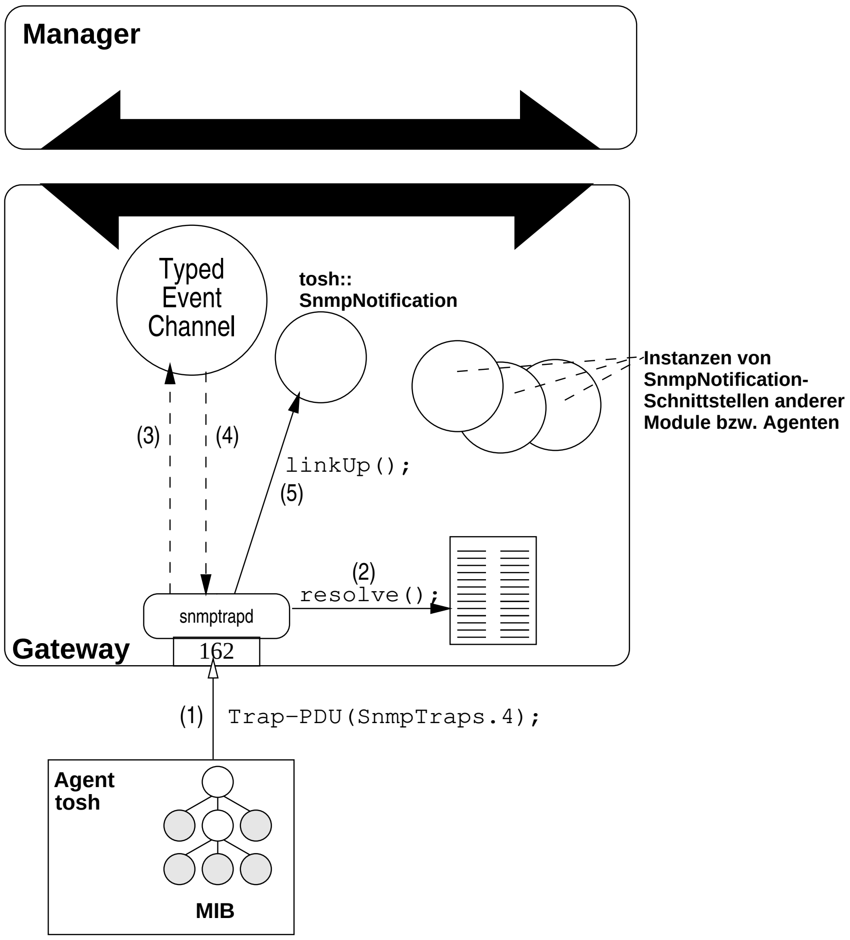

Abbildung 4.14: Jeder SNMP-Trap wird auf einen eigenen TypedEvent abgebildet

4.6 Vorstellung des Konzeptes

In den vorhergehenden Abschnitten wurden verschiedene Ansätze vorgestellt, die nun bewertet werden. Zuerst wird entschieden, ob ein zustandsloses oder ein zustandsbehaftetes Gateway als Basis für das Konzept dienen soll. Sobald dies feststeht, wird zwischen den verschiedenen Lösungsansätzen abgewogen.

Bei der Frage, ob ein zustandsloses oder ein zustandsbehaftetes Gateway besser geeignet ist, muß folgendes betrachtet werden (vgl. auch 4.2.1):

- Der Zugriff auf die Schattenobjekte von Seiten des Managers ist bei einem zustandsbehafteten Gateway sehr schnell, da kein Lookup auf die SNMP-Instanzen erfolgt. Bei einem zustandslosen Gateway muß hingegen bei jedem Zugriff auf ein Schattenobjekt mit einem SNMP-Agent kommuniziert werden.

- Der Zugriff auf SNMP-Ressourcen ist auf effektive und effiziente Weise möglich. Mit `GetBulk` und `GetNext` können mit einem Mal sehr viele SNMP-Variablen abgefragt werden. In einem zustandslosen Gateway hat dies keinen Sinn, da zusätzlich abgefragte Information nicht gespeichert werden kann.
- Im zustandsbehafteten Gateway können die im Gateway gespeicherten Werte mit den Werten der entsprechenden Managed Objects verglichen werden und so Veränderungen der Informationsinhalte festgestellt werden. Veränderungen können durch Events einer Managementanwendung gemeldet werden. Damit kann ein Controlling von SNMP-Ressourcen, welche sonst nur durch Polling verwaltet werden können, simuliert werden.
- Die im Gateway gespeicherten Werte müssen zum Zeitpunkt des Zugriffs mit den entsprechenden Werten in der SNMP-Umgebung konsistent sein. Häufig notwendige Aktualisierung durch SNMP-Requests kann sich negativ auf die Netzlast auswirken und damit den Vorteil des zustandsbehafteten gegenüber dem zustandslosen Gateway relativieren.
- Inkonsistenzen in den Schattenobjekten bei einem zustandsbehafteten Gateway können nicht vollständig ausgeschlossen werden. Um den Grad der Inkonsistenz minimal zu halten, sind komplexe Strategien und Mechanismen notwendig.

Fazit: Für statische Information ist ein zustandsbehaftetes Gateway, für Information, die sich oft ändert, ein zustandsloses Gateway geeigneter. Da in den Agenten sowohl gleichbleibende als auch dynamische Werte vorkommen, kann weder das zustandslose noch das zustandsbehaftete Gateway kategorisch abgelehnt werden. Die beste Lösung ist vielmehr ein Gateway, welches die „Dynamik" der Information in den einzelnen Managed Objects berücksichtigt, und zur Laufzeit entscheidet, ob eine SNMP-PDU benötigt wird. Dazu sind geeignete Kennzahlen und -werte notwendig, die das dynamische Verhalten der Managementobjekte beschreiben ([ACH93], [HAB91]).

Ein wichtiges Kriterium bei der Bewertung der besprochenen Ansätze ist also, in wie weit sich der jeweilige Ansatz für diese Lösung eignet. In dieser Diplomarbeit wurde für den in 4.3.3 auf Seite 51 beschriebene Ansatz entschieden. Abbildung 4.15 zeigt nochmal diesen Ansatz: Jedes Schattenobjekt repräsentiert mit seinen Methoden und Attributen eine oder mehrere SNMP-Managementobjektinstanzen. Eine Managementanwendung kann auf SNMP-Instanzen zugreifen, indem sie mit CORBA-Requests die (Attributzugriffs-)Methoden entsprechender Schattenobjekte aufruft. Für den Informationsaustausch mit SNMP-Agenten ist die Komponente snmpserver im Gateway zuständig. Die Funktionalität dieser Komponente besteht darin, SNMP-PDUs zu erzeugen, zu senden und zu empfangen[6]. Die Schattenobjekte

[6] SNMP-Trap-PDUs werden nicht behandelt, dafür ist der Trap-Dämon zuständig

rufen die Methoden dieses Gatewayobjektes auf und übergeben dabei als Parameter die Adresse des Zielagenten, an den die SNMP-PDU geschickt, und den Identifikator der SNMP-Instanz, auf die zugegriffen werden soll. Diese Werte werden einmalig in jedem Schattenobjekt bei dessen Erzeugung gesetzt. Dadurch erfolgt die Zuord-

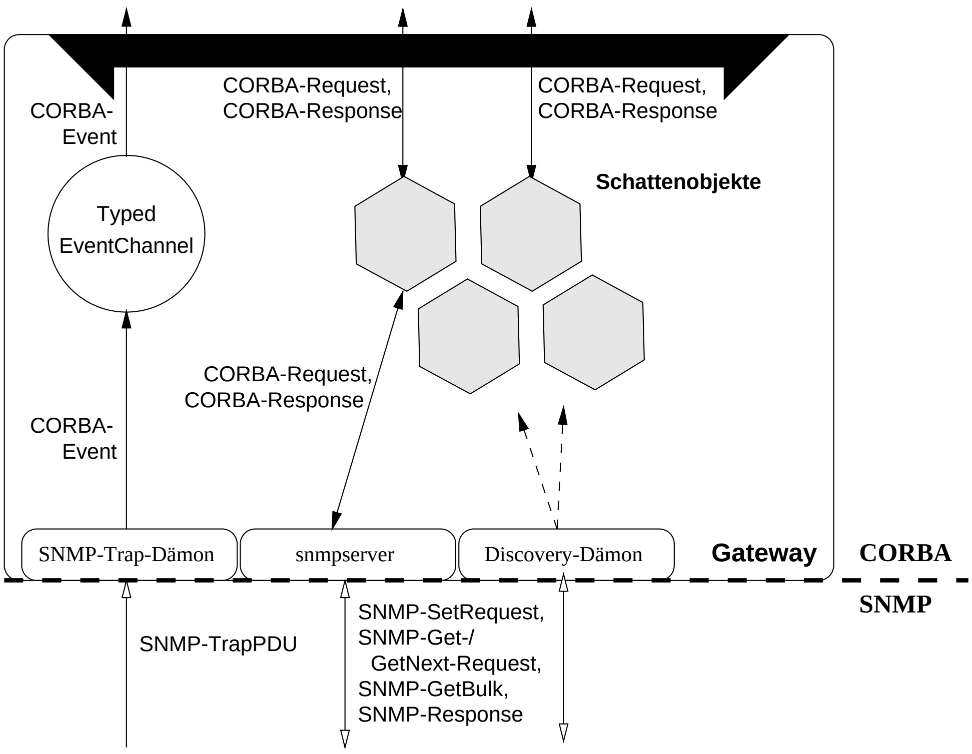

Abbildung 4.15: Gewähltes Gateway-Konzept

nung von einem Schattenobjekt (bzw. seiner Attribute) zu bestimmten, existierenden SNMP-Managementobjektinstanzen. Eine weitere Komponente im Gateway ist der Discovery-Dämon, dessen Aufgabe es ist, die SNMP-Umgebung zu explorieren und zu gefundenen SNMP-Agenten entsprechende Schattenobjekte zu erzeugen. Voraussetzung dafür ist, daß die Agenten-MIBs als IDL-Schnittstellendefinitionen im Interface Repository vorliegen. Ein SNMP-Trap-Dämon schließlich wartet an Port 162 des Gateway-Rechners auf ankommende SNMP-Trap-PDUs. Je nach Trap erzeugt er unterschiedliche CORBA-Events und schiebt sie in einen (typisierten) EventChannel. Dieser leitet die Events an alle bei ihm registrierten EventConsumer, beispielsweise an eine Managementanwendung weiter.

Dieser Ansatz hat folgende Vorteile:

- Die Schattenobjekte sind einfach, deren Implementierung ist dementsprechend einfach automatisierbar. Damit können sehr viele SNMP-Agenten schnell (d. h.

mit wenig Aufwand) einer CORBA-Managementanwendung zur Verfügung gestellt werden.

- Es besteht die Möglichkeit, die Schattenobjekte individuell zu implementieren; damit kann das Gateway stufenweise zu einem zustandsbehafteten Gateway erweitert werden.

- Die IDL-Schnittstellendefinitionen der Schattenobjekte müssen nicht um Methoden ergänzt werden, die das Gateway zur Aktualisierung der Attribute nutzen kann. Dies ist notwendig, wenn die Schattenobjekte nicht selbst die von ihnen gespeicherten Werte konsistent halten.

- Im Gegensatz zum Gateway, bei dem jedes Schattenobjekt selbst SNMP-PDUs erzeugt (4.3.2 auf Seite 48), kann bei diesem Ansatz die Komponente, welche die Kommunikation mit der SNMP-Umgebung übernimmt, ausgetauscht, erweitert und verbessert werden, ohne daß die Schattenobjekte verändert werden müssen (vorausgesetzt die Schnittstelle dieser Komponente bleibt gleich).

- In derselben Komponente können Informationen verwaltet werden. Beispielsweise könnten ein Community-String und/oder die Adresse eines bestimmten Agenten für SNMP-PDUs dort zentral gespeichert (und muß gegebenenfalls nur dort verändert) werden; Ebenso können statistische Daten, beispielsweise die Anzahl der verschickten SNMP-PDUs verwaltet und abgefragt werden.

- Sicherheitstechnisch hat dieser Ansatz den Vorteil, daß nur an einer oder wenigen Stellen der Übergang in die SNMP-Welt (im Sinne des Informationsaustausches) erfolgt, statt an beliebig vielen, wie es der Fall wäre, wenn jedes Schattenobjekt SNMP-PDUs erzeugt.

Es muß aber bedacht werden, daß es bei diesem Ansatz jedes Schattenobjekt mit einer Gatewaykomponente kommuniziert (mit Requests an diese Komponente), bevor eine SNMP-PDU erzeugt wird. Dieser zusätzliche Kommunikationsaufwand (innerhalb der CORBA-Umgebung) kann zu Performanceeinbußen führen. Umgekehrt kann die Komponente, welche SNMP-PDUs erzeugt, zum Engpaß werden. Dennoch wurde dieser Ansatz weiterverfolgt. Die im folgenden Abschnitt besprochenen Aspekte beschreiben das Gatewaykonzept genauer.

Im Bezug auf die Abbildung von SNMP-Trap-PDUs auf CORBA-Events wurde der ereignisgesteuerte Ansatz, also das *push*-Modell gewählt. Des weiteren wurde die typisierte Ereigniskommunikation der generischen vorgezogen. Die Entscheidunggründe wurden bereits im vorherigen Unterkapitel angeführt.

4.6.1 Konzeptspezifische Lösungsaspekte

Abbildung der Schattenobjekte auf Managementobjektinstanzen

Ein Schattenobjekt repräsentiert mit seinen Attributen und Methoden ein oder mehrere Managementobjektinstanzen in der SNMP-Umgebung. Bei einem Zugriff auf dieses Schattenobjekt muß also die entsprechenden SNMP-Zielinstanz bestimmt werden. Für die Identifikation einer Variableninstanz werden die Adresse des Agenten, dem diese angehört, der ihr Objektidentifikator und ihr Zugriffsidentifikator benötigt (s. 4.1).

Aus Gründen der Transparenz findet die entsprechende Abbildung innerhalb des Gateways statt. Zwei Möglichkeiten sind denkbar:

- Ein bestimmtes Gatewayobjekt übernimmt diese Aufgabe für alle Schattenobjekte,
- Jedes Schattenobjekt führt die Abbildung für alle eigenen Attribute bzw. Methoden aus.

Der erste Fall wird abgelehnt, da im Prinzip beliebig viele Namen (der Schattenobjekte) im Gateway verwaltet werden müssen. Die Suche nach Variableninstanzen zu einem Schattenobjekt ist zu aufwendig. Der zweite Fall ist wesentlich einfacher und schneller, da jedem Schattenobjekt nur eine kleine Menge von Managementobjektinstanzen zugeordnet ist, aus der die Zielinstanz bestimmt werden muß. Die Objektidentifikatoren der Managementobjekttypen, deren Instanzen von einem Schattenobjekt dargestellt werden können, stehen in der IDL-Beschreibung der Schattenobjektschnittstelle im Interface Repository. Sie können bei der Implementierung im Code des Schattenobjektes „fest verdrahtet“ werden. Wie auch immer können die Adresse des Agenten sowie der Zugriffsidentifikator der SNMP-Instanz erst zur Laufzeit ermittelt werden. Sie werden im Schattenobjekt in eigenen Attributen gespeichert. Bei der Erzeugung bzw. Initialisierung des Schattenobjektes müssen diese Attribute belegt werden. Ein Schattenobjekt kann damit mit konstantem Aufwand selbst den Identifikator der SNMP-Variableninstanz bestimmen und (notwendigerweise) dem Gateway mitteilen.

Erzeugung von Schattenobjekten

Damit eine Managementanwendung auf Schattenobjekte und damit auf SNMP-Ressourcen zugreifen kann, müssen diese Schattenobjekte notwendigerweise existieren. Da es die Aufgabe des Gateways ist, einer Managementanwendung SNMP-Ressourcen zugänglich zu machen, ist das Gateway konsequenterweise dafür zuständig, die SNMP-Umgebung zu erkunden und geeignete Schattenobjekte zu erzeugen.

Dabei sind nach Möglichkeit die in 4.1 aufgestellten Bedingungen einzuhalten. Prinzipiell sind zwei Vorgangsweisen möglich:

- Das Gateway sucht im Interface Repository nach Schnittstellen, die Agenten-MIBs beschreiben, und erzeugt a priori, d. h. ohne Kenntnis über den tatsächlichen Zustand der SNMP-Welt, die Schattenobjekte. Den Schattenobjekten können dadurch aber keine (mit Sicherheit existierenden) Managementobjekte zugeordnet werden.

- Die bessere Methode besteht darin, daß das Gateway die SNMP-Welt exploriert und zu jedem gefundenen Agenten Schattenobjekte erzeugt, sofern dessen MIB in IDL-Notation im Interface Repository gefunden wird[7].

Die Schattenobjekte müssen bei ihrer Erzeugung richtig initialisiert werden. Konkret bedeutet dies, daß die Schattenobjektattribute für den Zugriffsidentifikator der SNMP-Instanzen und die Adresse des entsprechenden SNMP-Agenten belegt werden müssen. Andernfalls kann die im vorherigen Abschnitt beschriebene Abbildung nicht stattfinden. Besonderes Augenmerk gilt den Zugriffsidentifikatoren:

- Bei Schattenobjekten, deren Attribute Managed Objects repräsentieren, die nicht Element einer SNMP-Tabelle sind (und deshalb auf einem Agenten nicht mehrmals vorkommen dürfen), kann das Attribut für den Zugriffsidentifikator automatisch mit ".0" belegt werden.

- Bei Schattenobjekten, deren Attribute Managed Objects repräsentieren, die Spaltenelement einer SNMP-Tabellenzeile sind, muß dasselbe Attribut mit dem Indexwert dieser Tabellenzeile belegt werden. Dieser kann mit Hilfe von `GetNextRequest-PDUs` Zeile für Zeile herausgefunden werden.

Hinweis: Auch der Managementanwendung ist es erlaubt, Schattenobjekte zu erzeugen, etwa wenn in einer Tabelle eines SNMP-Agenten eine neue Zeile erzeugt werden soll. In diesem Fall ist die Managementanwendung selbst verantwortlich, daß das Schattenobjekt richtig initialisiert wird. Nach der Erzeugung und Initialisierung des Schattenobjektes muß die Managementanwendung außerdem zuallererst eine Set-Operation auf diesem Schattenobjekt ausführen. Erst dann wird im Agenten eine entsprechende Zeile erzeugt. Diese Bedingung ist aber (aufgrund der Logik) durchaus akzeptabel: ein Manager wird eine von ihm erzeugte Tabellenzeile zuerst belegen, bevor er ihren Inhalt abfragt.

[7] Es sollte allerdings möglich sein, diese Exploration von außerhalb (etwa durch spezielle Gatewaymethoden) zu steuern, z. B. zu unterbrechen.

Löschen von Schattenobjekten

Schattenobjekte werden in folgenden Situationen gelöscht:

- Die Managementanwendung löscht ein Schattenobjekt und die entsprechenden SNMP-Managementobjektinstanzen konnten ebenso „gelöscht" werden;

- Die SNMP-Managementobjektinstanzen, die von einem bestimmten Schattenobjekt repräsentiert werden, existieren nicht mehr (z. B. wegen des Ausfalls des Agenten). In diesem Falle ist das Schattenobjekt ungültig.

Der erste Fall besagt, daß ein Schattenobjekt im Gateway *nicht* gelöscht wird, wenn die entsprechenden Managementobjektinstanzen in der SNMP-Umgebung nicht entfernt werden können[8]. Damit wird gewährleistet, daß das Gateway mindestens die Managementinformation abbildet, die in der SNMP-Umgebung tatsächlich zugreifbar ist.

In einer zweiten Situation müssen Schattenobjekte gelöscht werden, damit das Gateway nicht mehr abbildet als den tatsächlichen Status der SNMP-Managementinformation. Beispielsweise können SNMP-Agenten ausfallen; die Schattenobjekte, die die Managementinformation dieses Agenten repräsentieren sind damit ungültig und werden gelöscht. Im Gateway wird der Ausfall des Agenten i. a. erst bekannt, wenn versucht wird, über ein Schattenobjekt auf die nicht mehr existierenden Managed Objects in der SNMP-Umgebung zuzugreifen. Dieses Problem kann im Grunde nur gelöst werden, indem das Gateway durch Polling die Gültigkeit der Schattenobjekte überprüft.

Fehlermeldungen

Die Ausnahmebehandlung bei Fehlern erfolgt in der CORBA-Umgebung durch Exceptions. Im CORBA-Standard sind eine Reihe von Standard-Exceptions definiert. Methoden können außerdem anwendungsspezifische Exceptions auslösen, was in IDL entsprechend definiert werden muß. Bei Attributzugriffsoperationen allerdings gilt dies nicht, da diese nicht explizit in der IDL-Schnittstellenbeschreibung definiert werden. Fehler, die beim Aufruf von Attributzugriffsfunktionen auftreten, können deshalb nur durch die CORBA-Standard-Exceptions beschrieben werden.

Aus Gründen der Transparenz müssen dem Manager SNMP-Fehlermeldungen in Form von Exceptions zurückgegeben werden. Außerdem sollen soviele SNMP-Errors wie möglich vom Gateway abgefangen werden. Dies betrifft vor allem protokollspezifische Fehlermeldungen. Beispielsweise muß der Fehler `tooBig` (SNMP-Message

[8] Es sein nochmals darauf hingewiesen, daß das eigentliche „Löschen" von SNMP-Instanzen nicht möglich ist. Eine SNMP-Instanz wird „gelöscht", indem sie als ungültig markiert wird. Dies ist außerdem nur für manche SNMP-Tabellenzeilen erlaubt.

zu groß) vom Gateway behandelt werden, das ja die Verantwortung für korrekte SNMP-Message trägt.

Andere Fehler wiederum können durch die vorgenommene Abbildung der Informationsmodelle nicht mehr oder nur noch selten auftreten. So wird der SNMP-Fehler `notWritable` zurückgegeben, wenn versucht wird, auf ein Managed Object, das nur gelesen werden darf, schreibend zuzugreifen. Für ein derartiges Managed Object wird bei der ASN.1/IDL-Sprachabbildung ein Attribut mit dem Prädikat readonly definiert. Damit entfällt aber die implizite Definition einer set-Operation für dieses Attribut. Eine Managementanwendung kann also niemals schreibend auf dieses Attribut (und damit auf das entsprechende Managed Object) zugreifen.

Manche SNMP-Fehlermeldungen müssen aber auf CORBA-Standard-Exceptions abgebildet werden. So kann etwa bei dem SNMP-Fehler `authorizationError` die Exception NO_PERMISSION ausgelöst werden. SNMP-Fehler, für die es keine passende Exception gibt werden — wie in [JID95] vorgeschlagen — auf die Exception NO_IMPLEMENT abgebildet.

Sicherheitsaspekte

Wenn eine Managementanwendung befugt ist, auf ein Schattenobjekt ohne Einschränkung zuzugreifen, bedeutet das nicht, daß sie auf die entsprechenden Managed Objects bzw. den entsprechenden Agenten ebenso uneingeschränkt zugreifen kann. Der vollständige Zugriff auf SNMP-Agenten wird nur gewährt, wenn der für den konkreten Zugriff erforderliche Community-String des Agenten bekannt ist. Der Community-String ist Bestandteil eines einfachen Paßwort-Mechanismus', der Zugriff auf SNMP-Agenten durch Unbefugte verhindern soll. Ein Agent verwirft eine SNMP-Message oder gewährt nur eingeschränkten Zugriff (z. B. nur lesenden), wenn der darin enthaltene Community-String nicht mit dem des Agenten übereinstimmt.

Letztendlich kann der Community-String eines Agenten nur durch die Managementanwendung selbst mitgeteilt werden. Folgende Lösungen sind möglich:

- Bei jedem Zugriff auf ein Schattenobjekt muß die Managementanwendung den Community-String des dazugehörigen Agenten mit übergeben (als Parameter der Zugriffsmethode). Auch wenn diese Lösung die „sicherste“ ist, geht die Transparenz des Zugriffs auf SNMP-Ressourcen verloren und scheidet deshalb aus.

- In jedem Schattenobjekt wird bei seiner Initialisierung der Community-String des entsprechenden Agenten gespeichert. Das Gateway verwendet dazu einen „Standard“-Community-String; der Manager hat die Möglichkeit, den Community-String eines Schattenobjektes zu verändern, um mehr Zugriffsrechte zu erlangen. übergibt den Community-String der Gatewaykomponente, die

die SNMP-PDU erzeugt. Diese Lösung hat den Nachteil, daß der Manager (einmal) in allen Schattenobjekten, die einen Agenten repräsentieren, den Community-String setzen, damit er auf die gesamte Managementinformation dieses Agenten zugreifen kann.

- Die Community-Strings der Agenten, zu denen Schattenobjekte existieren, werden an zentraler Stelle im Gateway verwaltet. Eine Managementanwendung hat die Möglichkeit, Community-String im Gateway einzutragen bzw. zu ändern. Anhand der Adresse des Agenten, die vom Schattenobjekte übergeben wird, kann entschieden werden, welcher Community-String für die aktuelle SNMP-Message verwendet werden muß. Damit muß der Community-String eines Agenten nur einmal gesetzt werden (und gilt dann für alle Schattenobjekte, die diesen Agenten darstellen).

In den beiden letzten Fällen wird der (eventuell konfigurierte) Community-String eines Agenten automatisch in der SNMP-Message verwendet. Damit besteht die Gefahr, daß auf SNMP-Ressourcen unbefugt zugegriffen wird, da die Attributzugriffsfunktionen eines Schattenobjektes im Prinzip von jedem Objekt aufgerufen werden können. Es ist deshalb notwendig, den Zugriff auf das CORBA/SNMP-Gateway und auf die Schattenobjekte zu kontrollieren. Der CORBA Security Service unterstützt dazu notwendige Sicherheitsmechanismen.

4.6.2 Einsatz der CORBA-Services

Da das Gateway selbst eine CORBA-Anwendung ist, können die CORBA-Dienste in Anspruch genommen werden. Die folgende Aufzählung beschreibt, an welchen Stellen im Gateway dies vorgesehen ist. Eine kurze allgemeine Einführung in die CORBA-Services findet man in 2.3.1.

LifeCycle Service: Im Gateway müssen i. d. R. sehr viele Schattenobjekte verwaltet werden. Eine Managementanwendung muß die Möglichkeit haben, Schattenobjekte zu erzeugen und zu löschen. Der CORBA LifeCycle Service ist deshalb ein wichtiges Hilfsmittel. Beim Internet-Management gibt es bis auf eine Ausnahme keine dem CORBA LifeCycle vergleichbare Mechanismen. Es wird ausschließlich die Möglichkeit unterstützt, SNMP-Tabellenzeilen zu erzeugen und zu löschen. Von Managementobjekttypen, die nicht Element einer SNMP-Tabelle sind, kann es nur eine Instanz in einem Agent geben. Weitere Managed Objects desselben Typs können weder erzeugt noch gelöscht werden. Managed Objects können auch nicht über Adreßräume verschoben werden. Operationen auf Schattenobjekten, die mit dem LifeCycle Service zusammenhängen, können also nicht in die SNMP-Umgebung repliziert werden. Ausnahme ist, wenn ein CORBA-Manager ein Schattenobjekt erzeugt, das eine SNMP-Tabellenzeile repräsentiert. In diesem Fall muß der entsprechenden

Tabelle eines Agenten eine neue Zeile hinzugefügt werden (mit einer Set-PDU). Analoges gilt für das Löschen von derartigen Schattenobjekten.

Naming Service: Der Naming Service wird im Gateway an folgenden Stellen eingesetzt:

- Zu einem Managed Object in der SNMP-Umgebung muß eine Factory gefunden werden, die ein geeignetes Schattenobjekt erzeugt. Da eine Factory auch ein Objekt ist, kann ihr der (zusammengesetzte) Name des Managed Objects, für das es ein Schattenobjekt erzeugen kann, in einem Naming Graph zugewiesen werden. Ein Factoryobjekt kann dadurch schnell gefunden werden.
- Erzeugte Schattenobjekte werden in einem Naming Graph eingetragen. Sowohl das Gateway als auch die Managementanwendung können diesen Naming Graph verwenden, um Objektreferenzen auf die Schattenobjekte zu erhalten. Ähnlich wie Managementobjektinstanzen in einem MIB-Baum eines SNMP-Agenten registriert und identifiziert werden, können Schattenobjekte aus einem Naming Graph ausgewählt werden.

Event Service: Der Einsatz des *Event Service* ([OMG96]) wurde schon in 4.5 besprochen und soll hier nicht noch einmal behandelt werden.

Concurrency Control Service: Ein einfacher Object Adapter reicht alle an ihn gerichteten Requests in der Reihenfolge an die (Schatten)objekte weiter, in der sie beim ihm eintreffen. Erst wenn ein Request abgearbeitet wurde, kann der nächste Request behandelt werden. Aus Effizienzgründen ist es wünschenswert, daß mehrere Requests gleichzeitig weitergereicht werden. Dies kann beispielsweise durch den Einsatz von Threads erreicht werden. In einem solchen Fall müssen Inkonsistenzen, die durch den simultanen Aufruf derselben Schattenobjektmethoden von zwei Threads entstehen können, ausgeschlossen werden. Dazu kann der Concurrency Control Service verwendet werden. In der SNMP-Umgebung werden Zugriffskonflikte auf Managed Objects von vornherein durch das Protokoll ausgeschlossen: Die PDUs treffen hintereinander beim Agenten ein und werden von diesem entsprechend sequentiell beantwortet.

Externalization Service: Bei einem zustandslosen Gateway werden in den Schattenobjekten keine Attributwerte gespeichert. Der „Zustand“ des Schattenobjektes entspricht den Werten der Managed Objects, die es repräsentiert. Bei der Externalisierung muß dieser Zustand jedesmal aus der SNMP-Umgebung geholt werden. Die Schnittstelle, welche die Schattenobjekte zur Nutzung des Externalization Service unterstützen müssen, ist dementsprechend aufwendig. Entsprechende SNMP-Dienste für SNMP-Managementobjekte stehen nicht zur Verfügung.

Relationship Service: Für eine Managementanwendung ist es oft vorteilhaft, bestimmte Beziehungen zwischen CORBA-Managementobjekten (Schattenobjekten) zu definieren. Eine Abhandlung über derartige Relationen ist nicht Thema dieser Diplomarbeit. Beim Internet-Management gibt es keine Entsprechung zum Relationship Service. Eine Abbildung auf SNMP-Dienste kann also nicht stattfinden. Aus diesem Grund wurde der Relationship Service nicht berücksichtigt.

Persistence Object Service: Der persistente Zustand eines Objektes bleibt auch nach dem Beenden des Prozesses, dem das Objekt zugeordnet war, erhalten. Ein Grund für den Einsatz des Persistence Object Service in einem Gateway ist also sofort ersichtlich: Nach einem Ausfall kann der Zustand des Gateway vor dem Ausfall wiederhergestellt werden. Dies ist allerdings nur für den Teil des Gatewayzustandes sinnvoll, der Steuerinformationen des Gateways betrifft. Managementinformationsinhalte, die im Gateway repliziert werden, müssen nach einem Ausfall neu aus der SNMP-Umgebung geholt werden. Persistent replizierte Werte werden zum Zeitpunkt des Neustarts in der Regel nicht mehr aktuell sein.

Der Persistence Object Service kann aber auch verwendet werden, um Arbeitsspeicher zu sparen (z.B. in dem Attributwerte, die sich selten ändern, auf einer Festplatte gespeichert werden). Für Schattenobjekte, in denen keine Managementinformationswerte gespeichert werden, erübrigt sich allerdings der Einsatz des Persistence Object Service (die Schattenobjekte haben keinen „eigenen“ Zustand, s. o.).

Transaction Service: Transaktionen werden vom SNMP-Protokoll nicht unterstützt, weshalb Zugriffe auf SNMP-Agenten mittels Transaktionen nicht realisierbar sind.

Query Service: Einen vergleichbaren Dienst von SNMP gibt es nicht. Trotzdem kann eine Managementanwendung Queries definieren Ein denkbarer Query auf Schattenobjekten wäre die Abfrage des Attributs sysName von allen Schattenobjekten, die dieses Attribut besitzen. Ein derartiger Query hätte zur Folge, daß auf allen diesen Schattenobjekten die Methode get_sysName aufgerufen wird. Dadurch werden die entsprechenden Managed Objects mittels einer SNMP-PDU abgefragt. Im Gateway wird also ein Query Service für SNMP-Managementobjekte (automatisch) simuliert. Wie beim Relationship Service müssen die Schattenobjekte dazu keine spezielle Schnittstelle unterstützen. Für das Gatewaykonzept muß der Query Service aus diesen Gründen nicht näher betrachtet werden.

Property Service: Der Property Service wurde noch nicht standardisiert. Davon abgesehen existiert in der SNMP-Umgebung kein vergleichbarer Mechanismus für Managementobjekte. In dieser Diplomarbeit wurde der Property Service deshalb nicht berücksichtigt.

Insgesamt bieten die CORBA Services zusammen mit den Common Management Facilities (s. 2.3.3) ein mächtiges Framework für die Entwicklung von Managementanwendungen. Als CORBA-Objekte können die Schattenobjekte unter Verwendung aller Dienste verwaltet werden: Sie können z. B. über Adreßräume verteilt, in unterschiedlichen Verfahrensklassen gruppiert oder in Relation zueinander gesetzt werden. Da aber die wenigsten Services eine Entsprechung in einem oder mehreren SNMP-Diensten haben, bleiben derartige Aktionen meist ohne Wirkung auf die SNMP-Umgebung.

In diesem Zusammenhang zeigt sich auch ein weiterer Vorteil, den die Integration von SNMP in die CORBA-Welt durch ein Managementgateway mit sich bringt. Alle Dienste, die in der CORBA-Architektur ein komfortables Management erlauben und die von der Internet-Architektur meist nicht einmal ansatzweise gegeben sind, sind durch die Integration auch für das Management von SNMP Managed Objects einsetzbar. Eine CORBA-Managementanwendung arbeitet bequem mit Schattenobjekten, während das Managementgateway alle notwendigen Abbildungen zwischen den beiden Welten durchführt.

Kapitel 5

Implementierung des Gateways

5.1 IBM's SOMbjects als Grundlage der Implementierung

Als Entwicklungsumgebung stand das *SOMobjects* Toolkit ([IBM96a, IBM96b]) von IBM zur Verfügung. *SOM (System Object Model)* ist eine objektorientierte Technologie um binäre Klassenbibliotheken zu erstellen. Durch das sprach-neutrale Objektmodell läßt sich dabei eine Reihe von Problemen heutiger objektorientierter Programmiersprachen lösen, wie beispielsweise die Inkompatibilität von Softwarekomponenten, welche in verschiedenen Sprachen implementiert oder von unterschiedlichen Compilern übersetzt wurden. Die Kompatibilität von Klassenbibliotheken wird erreicht, in dem Beschreibungen von Klassen und deren Implementierung entkoppelt werden. Außerdem enthält SOMobjects mit dem *Distributed SOM (DSOM)* einen ORB, der die Interoperabilität von Objekten und Anwendungen ermöglicht.

Die aktuelle Version 2.1 ist derzeit für AIX, OS/2, Windows und MVS verfügbar. Daneben liegt zur Zeit auch eine neue Version 3.0, allerdings erst als Beta-Software und nur für eine OS/2 Plattform vor. SOMobjects Version 3.0 enthält wie Version 2.1 nur einen CORBA-1.1-konformen[1] ORB, dafür sind aber alle Object Services Implementierungen der OMG CORBAservices ([OMG96]). Aus diesem Grund wird im folgenden die Version 3.0 vorausgesetzt.

Das SOMobjects Entwicklungstool umfaßt einen SOM-Compiler, eine SOM-Laufzeitumgebung sowie mehrere Frameworks (Klassenbibliotheken), die die Entwicklung von objektorientierten Anwendungen unterstützen: *Distributed SOM (DSOM)*, das *Interface Repository Framework*, das *Emitter Framework* und das *Metaclass*

[1] So steht es in den Handbüchern von SOMobjects; bestimmte Systemdateien lassen aber darauf schließen, daß der ORB in Wirklichkeit sogar CORBA-2.0-konform ist, d. h., daß es möglich ist, den ORB des Toolkits an andere (CORBA-2.0-konforme) ORBs anzuschließen. In Pilotversuchen ist dies bereits gelungen.

Framework. Außerdem können eine Reihe von Object Services verwendet werden.

5.1.1 Der SOM-Compiler

Die Schnittstellen einer Klasse werden bei der Entwicklung mit SOM in der Beschreibungssprache IDL (Interface Definition Language) festgelegt. Die Schnittstellen werden dadurch von ihrer (später erfolgenden) Implementierung getrennt. Die SOM IDL entspricht der OMG IDL im CORBA-1.1-Standard; es werden alle CORBA-Datentypen unterstützt.

Der SOM-Compiler ist zuständig für die Sprachabbildung von IDL auf eine konkrete Programmiersprache. Die Sprachabbildung von IDL auf C ist CORBA-1.1-konform, ein Language Mapping für C++ wird ebenso angeboten. Zur Zeit kann C oder C++ ausgewählt werden.

5.1.2 Die SOM-Laufzeitumgebung

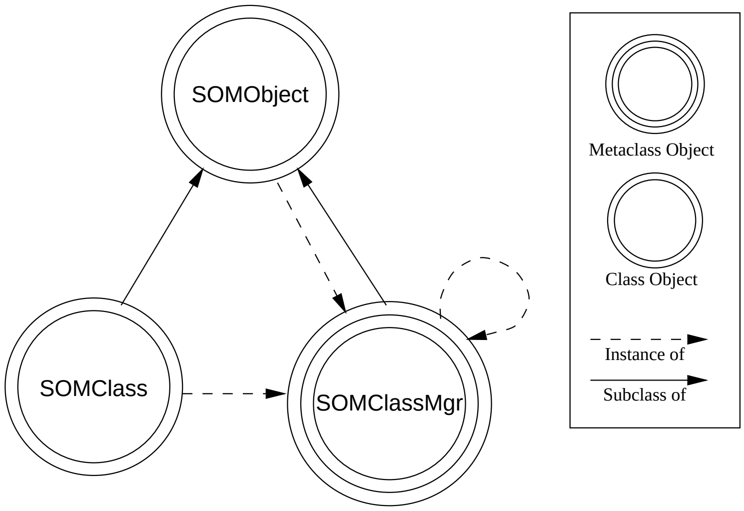

Abbildung 5.1: Die Klassen der SOM-Laufzeitumgebung

Die SOM-Laufzeitumgebung stellt eine Menge von Klassen, Methoden und Prozeduren zur Verfügung, die als Ausgangspunkt für die Entwicklung von Objektklassen und zur Verwaltung von Objekten benötigt werden. Sie besteht aus drei Klassen:

SOMObject: Von dieser Klasse sind alle Klassen abgeleitet. Sie definiert das grundlegende Verhalten aller Objekte.

SOMClass: In SOM ist zur Laufzeit jede Klasse wiederum ein Objekt (in dem Fall auch Klassenobjekt genannt). Deshalb muß auch SOMObject eine Instanz einer Klasse sein. Diese Klasse ist SOMClass, die Wurzel aller *Metaklassen*. Über Methoden und Attribute der Metaklassen können zur Laufzeit Informationen über Klassen gewonnen werden oder eigene Methoden für die Objekte einer Klasse definiert werden.

SOMClassMgr: Ein Objekt dieser Klasse wird automatisch bei der Initialisierung der SOM-Laufzeitumgebung erzeugt und dient der Registrierung aller existierender Klassen sowie dem Laden der Klassenbibliotheken und DLLs (Dynamic Linked Libraries).

Die Abbildung 5.1 zeigt die Beziehung zwischen diesen drei Basisklassen der SOM-Laufzeitumgebung.

5.1.3 Die CORBA-Implementierung DSOM

Distributed SOM (DSOM) ist eine Erweiterung von SOM. Sie erlaubt letztendlich Anwendungen, auf Objekten in verschiedenen Adreßräumen und auf unterschiedlichen Rechnern Operationen aufzurufen. Damit wird die Entwicklung einer verteilten Anwendung mit SOMobjects überhaupt erst ermöglicht. Die wichtigsten Unterschiede zwischen SOMobjects 2.1 und SOMobjects 3.0 betreffen Änderungen/Ergänzungen bezüglich dieses Frameworks.

Eine DSOM-Laufzeitumgebung ist in Abbildung 5.2 dargestellt. Sie besteht mindestens aus vier Komponenten:

- Einem Clientprogramm;
- Einem Serverprogramm, welches Client-Requests bedient. DSOM stellt ein „generischen“ Serverprogramm (somdsvr) und einen Server (somossvr), der benutzt werden kann, wenn SOMobject Object Services eingesetzt werden sollen, zur Verfügung. Auch applikationsspezifische Serverprogramme können entwickelt und eingesetzt werden.
- Dem DSOM-Dämonprozeß somdd, der das Serverprogramm startet und die Verbindung zwischen Client und Server herstellt; Der somdd läuft auf demselben Rechner, auf dem auch das Serverprogramm läuft.
- Einem Name Server, mit dessen Hilfe ein Client Objekte (insbesondere Factories) finden kann. Läuft der Name Server auf einem anderen Rechner als das Serverprogramm, so muß auch dort ein somdd gestartet worden sein.

Das Serverprogramm enthält, neben den Objekten der eigentlichen Anwendung, drei Laufzeitobjekte:

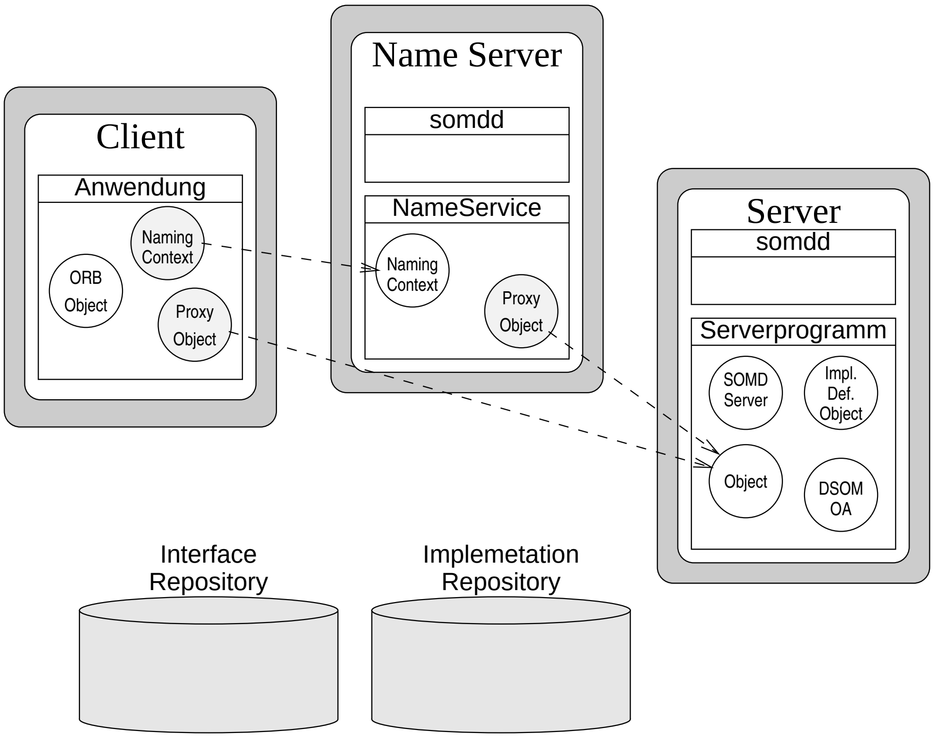

Abbildung 5.2: Die DSOM-Laufzeitumgebung

- Ein ImplementationDefObjekt, beschreibt das Programm, welches den Server implementiert (Aktivierung, Pfadname, Server-ID). Einerseits identifiziert ein ImplementationDefObject eindeutig einen Server in der DSOM-Laufzeitumgebung, andererseits benötigt das Serverprogramm selbst dieses Objekt, um das Verhalten seines Server Objects (s. u.) beeinflussen zu können. Ein Serverprogramm kann „sein" ImplementationDefObject aus dem Implementation Repository holen.

- Das Server Object ist eine Instanz der Klasse SOMDServer. Es stellt den Clients über seine Schnittstelle Dienste zum Erzeugen und Löschen von Objekten im Serverprozeß zu Verfügung. Der Object Adapter (s. u.) benutzt Methoden des Server Objects, um Objektreferenzen aufzulösen (um Objekte innerhalb des Serverprozesses zu identifizieren). Außerdem kann mit der Methode somdRefFromSOMObj eine Objektreferenz zu einem SOMObject erzeugt und damit dieses Objekt als Zielobjekt für Aufrufe von Clients beim Object Adapter angemeldet werden. Diesen Vorgang nennt man das Exportieren eines Objektes. Das Serverobjekt ist also auch an der Weiterleitung von Methodenaufrufen an Zielobjekte beteiligt.

- Der SOM Object Adapter ist die wichtigste Schnittstelle zwischen dem Serverprogramm und der DSOM-Laufzeitumgebung. Die Klasse des SOM Object Adapter ist von der abstrakten Klasse BOA (Basic Object Adapter, der Standard-CORBA Object Adapter), abgeleitet[2]. Die Hauptaufgabe des Object Adapter ist, Requests zu empfangen, an Zielobjekte weiterzureichen und Responses an die aufrufenden Clients zurückzugeben.

Innerhalb eines Clientprogramms muß ein ORB-Object vorhanden sein. Es handelt sich dabei um eine Instanz der DSOM ORB-Klasse, die die CORBA-konforme ORB-Schnittstelle implementiert. Bei der Initialisierung wird eine solche Instanz automatisch erzeugt. Es stellt dem Client Methoden zur Verfügung stellt, entfernte Objekte zu finden, zu erzeugen, zu löschen. Insbesondere erhält ein Client mittels des ORBs Objektreferenzen auf Objekte, welche Basisdienste anbieten (Interface Repository, Name Service). Außerdem bietet es Methoden an, die für die dynamische Aufrufschnittstelle benötigt werden. Auch im Serverprogramm kann der ORB genutzt werden.

Proxy-Objekte

Innerhalb des ORBs werden registrierte Objekte mittels Objektreferenzen identifiziert. Eine gültige Objektreferenz verweist auf genau ein Objekt. In DSOM werden Objektreferenzen auf verschiedene Weisen repräsentiert:

- als Instanz der Klasse SOMDObject,
- als Proxy-Objekt (eine Instanz der Klasse SOMDClientProxy).

Daneben gibt es noch ein Binärformat für den Transport der Objektreferenz über das Netz, und eine String-Form, um eine Objektreferenz abspeichern zu können.

In einem Server werden die Objekte durch Instanzen der Klasse SOMDObject identifiziert. Eine Objektreferenz eines Objektes im Server wird vom Object Adapter erzeugt. Solche Objektreferenzen sind (ebenso) Objekte, die benötigt werden, um Requests an das Zielobjekt weiterzuleiten und Proxy-Objekte desselben Zielobjektes in den Clients zu erzeugen. Ein Proxy-Objekt ist ein Stellvertreter eines Objektes. Es wird zur Laufzeit von SOM im Adreßraum eines Clients erzeugt und unterstützt die Schnittstelle des Objektes, das es vertritt. Der Client ruft Operationen lokal auf diesem Proxy-Objekt auf. Der Aufruf wird aber — für den Client nicht erkennbar — zum Server bzw. dem Zielobjekt weitergeleitet. Die Ortstransparenz der Objekte wird in DSOM also erreicht, indem dem Client vorgetäuscht wird, das gewünschte Zielobjekt liege in seinem lokalen Adreßraum.

[2] In der Abbildung 5.2 ist das Objekt DSOMOA innerhalb des Serverprogramms eine Instanz dieser Klasse.

Das Dynamic Invocation Interface von CORBA wurde ebenso durch Proxy-Objekte realisiert. Die Proxy-Objekte besitzen neben den von ihrem jeweiligen Zielobjekt geerbten Methoden weitere Operationen, mit denen explizit ein Request erzeugt werden kann. Es ist dabei gleichgültig, ob sich das Zielobjekt des Proxy-Objektes im lokalen oder in einem entfernten Adreßraum befindet. Die DSOM-Laufzeitumgebung sorgt für den korrekten Transport des Requests an das adressierte Zielobjekt.

5.1.4 SOMobjects Object Services

Die Schnittstellen der OMG CORBAservices ([OMG96]) werden bei SOMobjects als abstrakte Basisklassen definiert. Die konkret implementierten Schnittstellen der SOM Object Services sind von diesen Basisklassen abgeleitet. In der folgenden Tabelle wird gezeigt, inwieweit CORBAservices in SOM/DSOM realisiert wurden.

CORBAservice	SOM Object Service	Konformität
Naming	ja	ja
Event	ja	es wird nur generische Eventkommunikation unterstützt.
LifeCycle	ja	ja
Persistence	ja	ja
Externalization	ja	Methoden, die mit dem Relationship Service zusammenhängen, wurden nicht implementiert.
Transaction	ja	ja
Security	ja	nein
Concurrency	ja	ja
	Factory	-
	Object Identity	-

Mit Ausnahme des Event, Externalization und Security Service sind alle SOM Object Services mit dem entsprechenden OMG-Standard konform. Im SOMobjects Event Service sind sowohl das *pull-* als auch das *push*-Modell für die Ereigniskommunikation möglich. Wie beim CORBA Event Service kann ein Event Channel gleichzeitig ein *push-supplier* und ein *pull-supplier* sein. Allerdings wird die typisierte Eventkommunikation bei SOMobjects nicht unterstützt.

Ebenso unvollständig (im Bezug auf den entsprechenden CORBA-Service) wurde der SOM Externalization Service implementiert. Dieser Service enthält Klassen- und Methodendefinitionen, um Objekte als sog. *streams* (Datenströme) darzustellen. Damit werden die Daten eines Objektes „transportierbar“, d. h. sie können beispielsweise über das Netz verschickt werden. Wichtig ist, daß der Externalization

Service dabei die Konvertierung zwischen maschinenarchitekturspezifischen Formaten (z. B. big/little-endian) übernimmt. In DSOM wird dieser Service benötigt, um Methodenaufrufe mit Objekten als *pass_by_value*-Parameter zu realisieren. Der SOM Externalization Service variiert nur geringfügig von der CORBA-Spezifikation des CORBA Externalization Service. So wurden beispielsweise die Methoden write_graph und read_graph in der Klasse CosStream::SteamIO nicht implementiert, da ein Relationship-Service, mit dem diese Funktionen im Zusammenhang stehen, nicht vorhanden ist.

Daß der Security-Service von SOMobjects nicht CORBA-konform ist, liegt daran, daß ein Standard für den Security Service der OMG noch nicht zur Verfügung stand. Der SOM Security Service von SOMobjects 3.0 ist allerdings auch in Hinsicht der gebotenen Funktionalität noch unvollständig: derzeit wird nur die Authentifizierung von Clients unterstützt. Mit dem Security Service können beispielsweise keine Passwörter zur Authorisierung der Aufrufer verwaltet werden.

Mit dem Factory Service bietet DSOM Mechanismen an, mit denen Applikationen Factories zum Erzeugen von Objekten auffinden können. Er ist gewissermaßen eine Verfeinerung des LifeCycle Service, der mit zusätzlicher Funktionalität das Finden von Factories und das Erzeugen von Objekten vereinfacht. Die Schnittstellen des Factory Service sind allerdings nicht CORBA-konform. Mit dem SOM Object Identity Service ist es möglich, Objekte untereinander auf Gleichheit zu überprüfen. Er wird benötigt, da das Vergleichen von Proxy-Objekten nicht ausreicht, um die Gleichheit der jeweiligen Zielobjekte festzustellen. Verschiedene Proxy-Objekte können nämlich auf dasselbe Objekt zeigen. Einen entsprechenden CORBAservice gibt es nicht.

5.1.5 Die SOM-Frameworks

Neben DSOM enthält SOMobjects mehrere Frameworks, die die Implementierung von objektorientierten Anwendungen in wichtigen Punkten unterstützen. Es sind Klassenbibliotheken, die Objekte für bestimmte Aufgaben zur Verfügung stellen. Die Frameworks werden in der folgenden Aufzählung beschrieben:

Metaclass Framework: Dieses Framework kann eingesetzt werden, um Anwendungsklassen oder Metaklassen zu definieren. Dabei kann auf folgenden Metaklassen aufgebaut werden:

- *SOMMBeforAfter metaclass:* Oftmals ist es erwünscht, daß vor und/oder nach einem Methodenaufruf auf einem Objekt automatisch bestimmte Methoden ausgeführt werden. Mit der Metaklasse *SOMMBeforAfter* kann eine Klasse dieses Objektes (zusammen mit den automatisch aufzurufenden Methoden) definiert werden.

- *SOMMSingleInstance metaclass:* Klassen, die von dieser Klasse abgeleitet werden, können nur einmal instantiiert werden.
- *SOMMTraced metaclass:* Methoden von Klassenobjekten dieser Metaklasse geben vor und nach ihrer Ausführung Meldungen mit den Parametern bzw. den Rückgabewerten (auf die Standardausgabe) aus. Damit lassen sich Methodenaufrufe zurückverfolgen.
- In SOMobjects 3.0 neu hinzugekommen ist die Metaklasse *SOMMProxyFor*. Die Erzeugung von Proxy-Objekten und Proxy-Objektklassen erfolgt bei SOM/DSOM automatisch. Es ist aber oft notwendig, zu anwendungsspezifischen Objekten spezielle Proxy-Objekte bzw. -klassen zu definieren. Proxy-Objektklassen werden von der Metaklasse *SOMMOProxyFor* abgeleitet.

Collection Class Framework: Diese Klassenbibliothek stellt dem Anwendungsprogrammierer viele hilfreiche Klassen zur Verfügung. Es handelt sich um häufig benötigte Datenstrukturen wie Hash-Tabellen, Dictionaries, Mengen, Linked Lists, Sortierte Listen, Warteschlangen, und Priority-Queues.

Event Management Framework: Dieses Framework definiert Klassen für das Versenden und Empfangen von asynchronen Ereignismeldungen. Mit ihm können Ereignismeldungen zentral verwaltet werden. Im Gegensatz zum Event Service sind Ereignismeldungen dabei eigene Objekte. Es gibt *Client Events* (anwendungsspezifische Events), *Timer Events*, *Sink Events* (Meldungen über Ein- und Ausgabeaktivitäten auf Event Sinks wie z. B. Sockets oder Dateideskriptoren) und *WorkProc Events*[3]. Die Weiterverarbeitung der Events erfolgt durch *Callback*-Prozeduren. Eine Anwendung muß für jede Eventart, die sie empfangen will, einen Callback definieren. Für den Empfang von Ereignismeldungen kann sich eine Anwendung bei einem Event-Manager-Object anmelden (und auch abmelden). Auch wenn diesbezüglich eine Ähnlichkeit zu einem Event Channel besteht, so ist das Event Management Framework nicht CORBA-konform.

Emitter Framework: In Zusammenhang mit dem SOM Compiler stehen sogenannte *Emitter*, welche die Ausgabe des Compilers in einem bestimmten Format abspeichern. Von SOMobjects mitgeliefert werden zwei Emitter, der C- und der C++-Emitter, welche Header-Dateien und die Programmskelette in C bzw. C++ erzeugen. Mit dem Emitter Framework können spezielle Emitter entwickelt werden, z. B. solche, welche für die Erzeugung von Dokumentationsdateien verwendet werden können.

Interface Repository Framework: Bei SOM wird das Interface Repository — eigentlich ein integrales Teil der CORBA-Architektur — als eigenes Framework

[3] Diese Art von „Events“ sind Hintergrundprozesse, die ausgeführt werden, wenn gerade keine Ereignismeldungen vorliegen.

betrachtet. Dies ändert nichts an der Tatsache, daß das SOM Interface Repository CORBA-konform ist. Zusätzlich können im SOM Interface Repository allerdings SOM-IDL-spezifische Informationen abgelegt werden (in Form von *Modifikatoren, modifiers*). Beispiele hierfür sind Versionnummern und Kurzbeschreibungen von Klassen oder Compilerdirektiven für die Erzeugung der Programmskelette.

Unterschiede zu SOMobjects 2.1

Das Replication Framework sowie das Persistence Framework aus Version 2.1 sind in SOMobjects 3.0 nicht mehr vorhanden. Letzteres Framework wurde durch den Persistence Object Service ersetzt, die Funktionalität des Replication Framework ist Teil der Funktionalität des LifeCycle Service. Weitere Änderungen sind u. a.:

- Neue Modifikatoren für die SOM IDL Dateien, mit denen die Erzeugung von C- oder C++-Sourcecode gesteuert werden kann.
- SOMobjects 3.0 stellt die Möglichkeit zur Verfügung, Objekte zu „casten“.
- Das Implementation Repository wird nur noch auf Rechnern benötigt, auf denen Server laufen. In Version 2.1 wurde das Implementation Repository von Client verwendet, um Informationen über DSOM-Server zu erhalten. Außerdem wurden weitere Schnittstellen realisiert, die die Eigenschaften des Implementation Repository unterstützen.
- Die Ortstransparenz für Objekte wurde verbessert. Methoden, die nur auf Proxy-Objekten (Objektreferezen auf Objekte) aufgerufen werden konnten, können in Version 3.0 auch auf lokalen Objekten aufgerufen werden.
- Es wurden neue Schnittstellen und Methoden realisiert, die die Arbeit mit den ebenso neuen Object Services vereinfachen.

5.2 Realisierung des Gateways mit SOM/DSOM

Die Abbildung 5.3 zeigt das CORBA/SNMP-Gateway als DSOM-Serverprogramm. Das Gateway kann entweder automatisch vom somdd oder manuell von der Kommandozeile oder von einer Anwendung aus gestartet werden. Dabei wird ein Object Adapter Objekt erzeugt, welches wiederum ein Server Objekt (in der Abbildung SOMOSServer, da Object Services genutzt werden) instantiiert. Anschließend wird das Gatewayobjekt (snmpserver erzeugt und exportiert. Die Methoden dieses Objektes werden hauptsächlich von den Schattenobjekten benötigt. Wenn eine Managementanwendung auf das snmpserver-Objekt zugreifen will (z. B. zu Konfigurationszwecken), benötigt sie ein Proxy-Objekt zu diesem Objekt (siehe Abb. 5.3).

Nach der Initialisierung ist das Gateway prinzipiell bereit, Requests von Clients zu empfangen und zu verarbeiten. Noch existieren im Gateway allerdings noch keine Schattenobjekte, die noch erzeugt werden müssen.

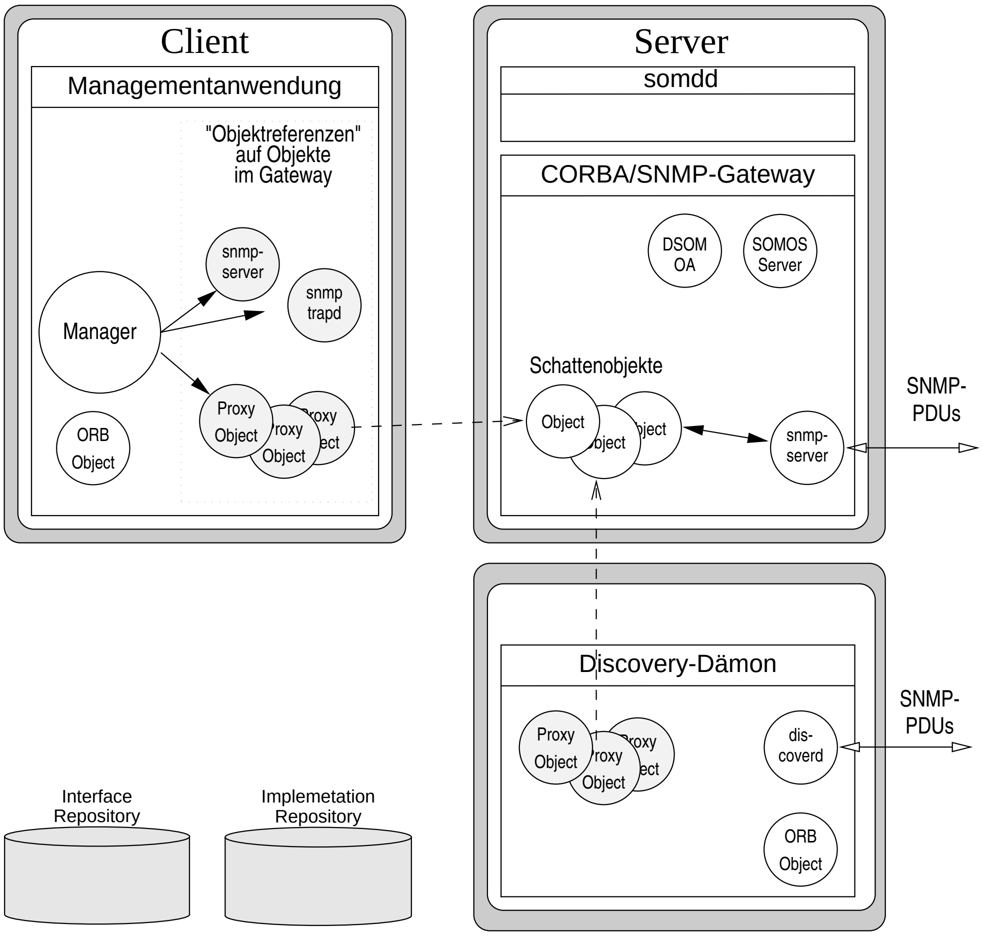

Abbildung 5.3: Das Gateway als DSOM-Serverprogramm

Verwaltung der Schattenobjekte im Gateway

Für das Erzeugen von Schattenobjekten im Gateway ist der Dämon discoverd zuständig. Er macht SNMP-Agenten und SNMP-Managementobjekte ausfindig und erzeugt entsprechende Schattenobjekte im Gateway. Wie schon erwähnt, benötigt es dazu das Interface Repository (Zugang über das ORB-Object), um geeignete Schnittstellen zu finden und entsprechende Factories. Aus folgenden Gründen ist es besser, den Discovery-Dämon nicht als Teil des Gateways zu implementieren:

- Der Dämon kann als eigener Prozeß implementiert werden. Die Aufgabe des Gateways und die des Dämons sind voneinander unabhängig. Das Gateway arbeitet Requests ab, der Dämon hingegen erkundet die SNMP-Umgebung, d. h. sucht SNMP-Agenten. Deshalb sollte die Funktionalität auch getrennt implementiert werden, was zudem den Parallelitätsgrad insgesamt erhöht.

- Der Gatewayserver tritt nach der Initialisierung in eine Endlosschleife ein, um Requests abzuarbeiten. Schattenobjekte, die nach dem Eintritt in diese Endlosschleife von einem Objekt innerhalb des Gateways erzeugt werden, können nur mit sehr hohen Programmieraufwand exportiert werden. Wenn der Discovery-Dämon hingegen durch einen Request an den Gatewayserver die Erzeugung eines Schattenobjektes anfordert, geschieht dies gewissermaßen automatisch. Statt dem erzeugten Schattenobjekt oder einem Zeiger auf dieses Objekt (der im Adreßraum des Clients ungültig wäre), gibt der Object Adapter ein entsprechende Proxy-Objekt zurück. Die Generierung eines Proxy-Objektes (mit der Methode somdRefFromSOMObj des Objekt Servers) entspricht aber genau der Exportierung des dazugehörigen Objektes ([IBM96a]).

Wenn ein Manager ein Schattenobjekt im Gateway erzeugen will (Tabellenzeilen!), benötigt er ein Proxy-Objekt einer passenden Factory im Gateway. Beim Erzeugen des Schattenobjektes wird ihm ein Proxy-Objekt dieses Objektes zurückgegeben. Alle weiteren Aufrufe erfolgen dann lokal auf dieses Proxy-Objekt. Die DSOM-Laufzeitumgebung sorgt für die transparente Weiterleitung des Aufrufs an das eigentliche Zielobjekt im Gateway.

Schattenobjekte werden in einen Nameserver eingetragen[4] und auf diese Weise einer Managementanwendung zugänglich gemacht. Damit eine Managementanwendung den Nameserver nicht andauernd nach neu hinzugekommenen Schattenobjekten pollen muß, erzeugt jedes Schattenobjekt bei seiner Initialisierung eine Ereignismeldung.

Das Löschen von Schattenobjekten im Gateway vom Manager aus erfolgt durch einen entsprechenden Methodenaufruf auf dem dazugehörigen Proxy-Objekt, der an das Schattenobjekt weitergegeben wird. Eine Fehlermeldung muß zurückgegeben werden, wenn die Operation nicht in die SNMP-Umgebung repliziert werden kann. (Wird stattdessen ein Schattenobjekt gelöscht, das SNMP-Managementobjekte repräsentiert, die nicht gelöscht werden können, so ist die Abbildung der SNMP-Ressourcen in die CORBA-Umgebung nicht mehr konsistent.) Alternativ kann die Managementanwendung nur die Proxy-Objekte löschen, sodaß die Schattenobjekte bestehen bleiben. Dazu ruft sie die Methode release des Proxy-Objektes auf. Die Entscheidung, ob das Proxy-Objekt oder das Schattenobjekt gelöscht werden soll, muß von der Managementanwendung getroffen werden.

[4] In der Abbildung 5.3 ist der Nameserver der Übersichtlichkeit halber nicht dargestellt.

Kommunikation mit SNMP-Agenten

Die Schattenobjekte nutzen die Methoden des Gatewayobjekts snmpserver, um mit SNMP-Agenten zu kommunizieren. Diese Methoden können auf zwei Weisen aufgerufen werden:

- direkt, d. h. ohne den ORB zu nutzen.
- mit einem Request.

Für die erste Variante benötigt das Schattenobjekt nur einen Pointer (im C- bzw. C++-Sinne) auf das Objekt snmpserver. Der Aufruf einer Methode dieses Objektes erfolgt ohne Kommunikation über den ORB, beispielsweise mit der Code-Zeile `response = snmpserver_instance→snmp_get();`. Auf diese Weise ist ein Aufruf zwar sehr schnell, die Ortstransparenz geht aber verloren, da der Zeiger nur im Adreßraum des Gateway-Prozesses gültig ist. Ein Schattenobjekt kann z. B. nicht auf einen anderen Rechner verschoben werden. Gerade diese Ortstransparenz ist aber der entscheidende Vorteil, den der Einsatz von CORBA mit sich bringt. Trotz des Overheads wird deshalb die zweite Variante — also der Aufruf der snmpserver-Methoden mittels eines Requests — vorgezogen.

Behandlung von SNMP-Traps

Aus ähnlichen Gründen wie beim Discovery-Dämon soll der SNMP-Trap-Dämon ebenfalls als eigener Prozeß implementiert werden (Abbildung 5.4). Der SOM Event Service unterstützt nur die generische Ereigniskommunikation. Um einige Nachteile der generischen Eventkommunikation zu umgehen, wird folgender Vorschlag gemacht. Mit dieser Lösung wird ermöglicht, daß eine Managementanwendung gezielt bestimmte „SNMP-Events“ empfangen kann:

- Für jeden SNMP-Trap-Typ (coldStart, warmStart, linkUp, ...) erzeugt der Trap-Dämon snmptrapd (zuständig für das Empfangen von SNMP-TrapPDUs) ein spezielles EventChannel-Objekt. (In der Abbildung 5.4 sind zwei dieser EventChannels eingezeichnet).
- Je nach Trap-PDU ruft der Trap-Dämon die push-Methode des entsprechenden EventChannels auf.
- Die EventChannels werden in einen Nameserver eingetragen, sodaß sie der Managementanwendung zur Verfügung stehen.
- Zum Empfang eines bestimmten Traps muß sich die Managementanwendung beim entsprechenden EventChannel registrieren. In der Abbildung 5.4 hat sich ein PushConsumer-Objekt der Managementanwendung bei einem der dargestellten EventChannel angemeldet.

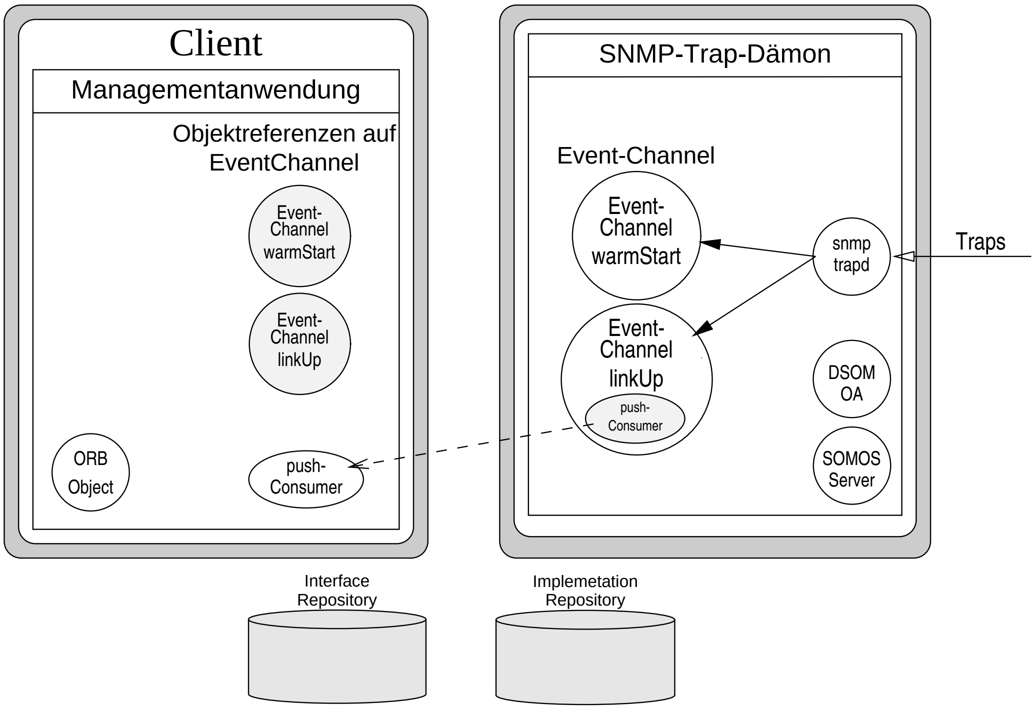

Abbildung 5.4: Behandlung von SNMP-Traps

5.3 Überblick über das Gateway

In diesem Abschnitt wird das bisher Besprochene zusammengefaßt, um einen kurzen Überblick über das Gateway zu geben. Im folgenden Abschnitt wird dann anhand einiger Beispiele gezeigt, wie dieses Gateway funktioniert.

Das Gateway besteht aus drei Komponenten: der *Discovery-Dämon*, der *SNMP-Trap-Dämon* und das eigentliche *Gatewayprogramm*. Da ihre Aufgaben weitgehend unabhängig sind, kann der Parallelitätsgrad des Gateways gesteigert werden, indem einzelne Prozesse diese Aufgaben übernehmen. Dabei ist es vorteilhaft und einfacher, die Komponenten jeweils in einem eigenen DSOM-Serverprogramm zu implementieren anstatt als Unterprozesse eines einzigen Serverprogramms.

Der Discovery-Dämon

Der *Discovery-Dämon* sucht in der SNMP-Umgebung nach SNMP-Agenten und dazugehörigen Managementobjektinstanzen. Beispielsweise können mit einem Subnetz-Autodiscovery verschiedene IP-Adressen von Rechnern ausfindig gemacht werden. Mit einem SNMP-Walk auf jede dieser Adressen kann festgestellt werden, ob auf dem entsprechenden Rechner SNMP-Agenten laufen. Dabei kann etwa die Tatsache ausgenutzt werden, daß auf SNMP-Agenten eine Inkarnation der MIB-II existieren muß. Im nächsten Schritt werden alle Objektidentifikatoren der Managed Objects einer Agenten-MIB gesucht. Hier zeigt sich das Problem, daß diese Agenten-MIB nicht nur die MIB-II beinhalten kann, sondern beispielsweise auch eine Private-MIB. Ein SNMP-Walk muß also „hoch genug" im Internet-Registrierungsbaum angesetzt werden.

Der Discovery-Dämon muß schließlich zu jeder OID einer SNMP-Variable aus einer SNMP-Gruppe oder SNMP-Tabelle das entsprechende Attribut (bzw. die der SNMP-Gruppe/Tabelle entsprechende IDL-Schnittstelle) finden. Dazu sucht er im Interface Repository nach einer Konstanten[5], die mit einer bestimmten OID übereinstimmt. Der Name der Konstanten entspricht dem Namen des gesuchten Attributes.

Der skizzierte Ablauf stellt keinesfalls die beste und einzige Lösung dar. In jedem Fall ist das gesamte Verfahren jedoch sehr komplex und rechenintensiv. Es ist auch unwahrscheinlich, daß eine derartige Suche vollautomatisch abläuft. Es ist deshalb notwendig, daß der Discovery-Dämon von einer Managementanwendung gesteuert werden kann, beispielsweise um den Suchbereich einzuschränken oder einen Suchvorgang abzubrechen.

Wenn die IDL-Schnittstellendefinitionen zu einer MIB eines gefundenen Agenten im Interface Repository abgelegt sind, erzeugt der Discovery-Dämon alle Instanzen dieser Schnittstellen (die Schattenobjekte), die notwendig sind, um die gesamte Managementinformation dieses Agenten darzustellen. So werden etwa zu einer SNMP-Tabelle mit fünf Zeilen genau fünf Instanzen der (einer Zeile) entsprechenden IDL-Klasse gebildet.

Die Schattenobjekte werden im Gatewayprogramm erzeugt, da auf diese Weise garantiert ist, daß Schattenobjekte sofort bei ihrer Erzeugung exportiert werden und der Object Adapter des Gatewayprogramms dadurch Requests an diese Objekte weiterreichen kann (vgl. 5.2).

Das Gatewayprogramm

Das *Gatewayprogramm* beinhaltet Schattenobjekte und ein Objekt `snmpserver`, das Methoden zum Versenden von SNMP-PDUs anbietet:

- Jedes Schattenobjekt repräsentiert eine oder mehrere SNMP-Instanzen eines Agenten. Sobald zur Laufzeit auf ein Attribut eines Schattenobjektes zugegriffen wird, muß die entsprechende Managementobjektinstanz identifiziert werden. Jedes Schattenobjekt hat dazu zwei zusätzliche Attribute, in denen die Adresse des Agenten, auf dem sich die SNMP-Instanz befindet, und der

[5] Bei der Übersetzung von ASN.1-Makros in IDL-Schnittstellendefinitionen werden die OIDs aller Managed Objects als Konstante übernommen, vgl. 3.1.1.

Instanzidentifikator gespeichert werden. Diese Attribute werden bei der Initialisierung durch den Discovery-Dämon einmal belegt. Jedes Schattenobjekt „kennt“ damit die Managementobjektinstanzen, die es repräsentiert. Auf diese Weise ist die Abbildung von Attribut auf SNMP-Managementobjektinstanz in konstanter Zeit möglich. Außerdem kann mit dem Zugriff auf das Attribut, das die Adresse des Agenten speichert, festgestellt werden, zu welchem SNMP-Agenten ein Schattenobjekt „gehört“.

Es ist abhängig von der Implementierung der Attributzugriffsmethoden des jeweiligen Schattenobjektes, wie auf einen Request reagiert wird. Im wesentlichen wird entweder eine snmpserver-Methode aufgerufen — und damit eine SNMP-PDU erzeugt — oder der Wert des gewünschten Attributs sofort zurückgegeben. Im Abschnitt 5.4 wird der Zugriff einer Managementanwendung auf Schattenobjekte an Beispielen genauer beschrieben.

- Das snmpserver-Objekt bildet den eigentlichen Zugangspunkt in die SNMP-Umgebung. Die Schattenobjekte rufen Methoden dieses Objektes auf, welches daraufhin SNMP-PDUs erzeugt und versendet. Die Adresse des Zielagenten und der Identifikator der SNMP-Instanz, die gelesen oder gesetzt werden soll, werden dabei vom aufrufenden Schattenobjekt übergeben. Für den Zugriff auf einen Agenten benötigt das snmpserver-Objekt den Community-String. Zu jedem Agenten verwaltet das Objekt deshalb eine Liste oder Tabelle von SNMP-Agenten und ihren Community-Strings. Eine andere Möglichkeit wäre, daß die Schattenobjekte wie die Adresse auch den Community-String des Zielagenten dem snmpserver als Methodenparameter übergeben. Erstere Lösung hat aber folgenden Vorteil: Ein Community-String eines Agenten muß gegebenenfalls nur an einer Stelle geändert werden und nicht in allen betroffenen Schattenobjekten. Es muß allerdings beachtet werden, daß Unbefugte dadurch einfach Zugriff auf SNMP-Ressourcen bekommen können. Es ist deshalb notwendig sowohl den Zugriff auf Schattenobjekte als auch den Zugriff auf das snmpserver-Objekt zu sichern.

Aus Performacegründen sind der snmpserver und die Schattenobjekte im Gatewayprogramm zusammengefaßt. Es ist aber prinzipiell möglich, Schattenobjekte in verschiedene Adreßräume zu verschieben.

Der SNMP-Trap-Dämon

Der *SNMP-Trap-Dämon* empfängt SNMP-Trap-PDUs. Für jeden SNMP-Trap wird ein eigener Event Channel erzeugt. Eine Managementanwendung kann sich bei einem oder mehreren dieser Event Channel registrieren, um bestimmte Arten von SNMP-Traps (in Form von CORBA-konformen Events) zu empfangen. Typisierte Event Channels können nicht eingesetzt werden, da in SOMobjects typisierte Eventkommunikation nicht unterstützt wird. In Abschnitt 5.4 wird an einem Beispiel die Behandlung eines SNMP-Traps gezeigt.

Einsatz der SOM Object Services

Folgende SOM Object Services werden eingesetzt:

- der *Naming Service*, um alle relevanten Laufzeitobjekte des Gateways (Schattenobjekte, Event-Channel) der Managementanwendung zur Verfügung zu stellen. Eine Managementanwendung kann ein Schattenobjekt über dessen Namen oder dessen Objektreferenz im Naming Graph (s. 2.3.1) adressieren.
- der *LifeCycle Service*; Alle Schattenobjekte unterstützen den LifeCycle Service. Die Schattenobjekte werden mittels Factories erzeugt, können innerhalb des ORB verschoben und gelöscht werden.
- der *Event Service* für Ereignismeldungen.

5.4 Funktionsweise des Gateways

Die Abbildung 5.5 zeigt eine Beispielkonfiguration des Gateways zur Laufzeit. In der SNMP-Umgebung läuft ein SNMP-Agent auf dem Rechner „tosh“. Von der MIB-II des Agenten sind die `system-Group` und die Routing-Tabelle `ipRouteTable` der `ip-Group` (beide aus der `mibII`, [MR91]) abgebildet. Der Wert der Variable `sysDesc` ist “SunOS 4.0.1“; die Routing-Tabelle besteht aus zwei Zeilen. Eine Zeile (bzw. die SNMP-Instanzen, die zu dieser Zeile gehören,) wird mit dem Wert des Spaltenobjektes `ipRouteDest` eindeutig identifiziert. Der Agent kann über die IP-Adresse (129.188.215.16) und die Portnummer (161) angesprochen werden.

Im Gatewayprogramm (s. o.) wurden durch den Discovery-Dämon Schattenobjekte erzeugt, die den dargestellten Auszug aus der Agenten-MIB entsprechen. Zu der SNMP-Gruppe `system` wurde eine Instanz der IDL-Schnittstelle systemGroup generiert. In dieser Schnittstelle wurde für jedes Managed Object der SNMP-Gruppe `system` ein Attribut definiert (s. 3.1.1). Auf die Instanz eines Managed Object kann über das gleichnamige Attribut des Schattenobjektes zugegriffen werden. Für das Managed Object `sysDescr` beispielsweise (s. Abb. 5.5) stehen dazu zwei Methoden, get_sysDescr und set_sysDescr, zur Verfügung. Bei einem Aufruf dieser Methoden werden die Werte zweier zusätzlicher Attribute benötigt, um die richtige SNMP-Instanz zu identifizieren. Das Attribut destination des Schattenobjektes der Klasse systemGroup trägt die IP-Adresse des Agenten. Das Attribut indexinfo ist mit dem gemeinsamen Zugriffsidentifikator (“.0“) der SNMP-Variablen belegt, die

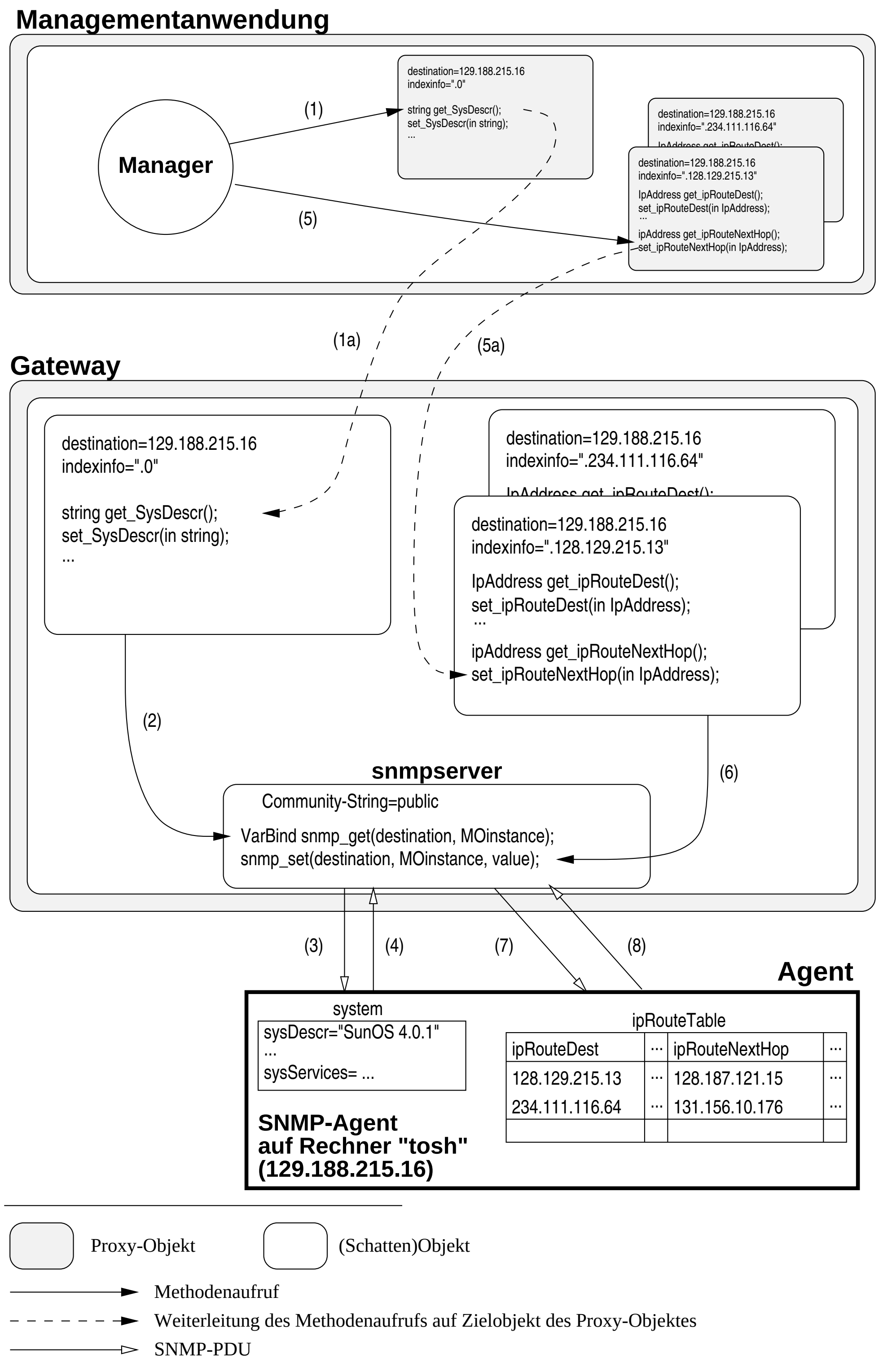

Abbildung 5.5: Beispielkonfiguration des Gateways

das Schattenobjekt repräsentiert[6]. Zur Erinnerung: Die Objektidentifikatoren der SNMP-Variablen sind als Konstante im Interface Repository abgelegt.

Im Beispiel wurde für jede Zeile der Routing-Tabelle eine Instanz der IDL-Klasse ipRouteTableEntry erzeugt. Bei beiden Schattenobjekten ist das Attribut destination mit der IP-Adresse des SNMP-Agenten belegt. Die Attribute indexinfo sind hingegen mit dem *Wert* desjenigen Spaltenobjektes belegt, das einen Zeileneintrag der Routing-Tabelle eindeutig identifiziert (i. e. der Wert des Indextyps `ipRouteDest`). Ein Schattenobjekt trägt den „Indexwert" (".128.129.215.13") der ersten Tabellenzeile, das andere den der zweiten Zeile (".234.111.116.64"). Dadurch sind die Schattenobjekte eindeutig einer Zeile der Routing-Table zugeordnet. Die Elemente einer Zeile können mit den Attributzugriffsfunktionen des entsprechenden Schattenobjektes abgefragt und/oder gesetzt werden.

Im Gateway befindet sich neben den Schattenobjekten auch noch das snmpserver-Objekt. Die Methoden snmp_get() und snmp_set() erzeugen, versenden und empfangen SNMP-PDUs (allerdings keine SNMP-Trap-PDUs, diese werden vom Trap-Dämon verarbeitet). Der Einfachheit halber wird für alle SNMP-Messages derselbe „public"-Community-String verwendet.

Ein Client-Objekt der Managementanwendung (in der Abb. 5.5 der „Manager") greift nicht direkt auf die Schattenobjekte zu, sondern ruft die Methoden ihrer Proxy-Objekte auf. Die DSOM-Laufzeitumgebung sorgt dafür, daß ein Aufruf einer Proxy-Objekt-Methode umgeleitet wird auf das Zielobjekt des Proxy-Objektes. Dies ist für das aufrufende Objekt nicht erkennbar. Die Proxy-Objekte können in der Managementanwendung beliebig angeordnet werden. Die Schnittstellen der Objekte (bzw. Proxy-Objekte) können durch Dienste des ORBs festgestellt werden.

5.4.1 Einfaches Get/Set

Zunächst sei angenommen, daß der Manager vom Schattenobjekt der Klasse systemGroup den Wert des Attributs SysDescr („sichtbar" durch die Methoden get_SysDescr und set_SysDescr) lesen will. Er ruft also die Methode get_SysDescr auf (Abb. 5.5(1)). Es folgt einer Reihe von Aktionen, die für den Manager transparent sind[7]:

- Der Aufruf wird von der DSOM-Laufzeitumgebung auf das Zielobjekt des Proxy-Objektes weitergeleitet (Abb. 5.5(1a)).
- Innerhalb der aufgerufenen Methode erfolgt ein Aufruf der Methode snmp_get des snmpserver-Objektes (2), von dem jedes Schattenobjekt eine Objektrefe-

[6] Der Zugriffsidentifikator ist für alle SNMP-Objektinstanzen, die — wie die der SNMP-Gruppe `system` — nicht Element einer Tabelle sind, gleich.

[7] der Manager ist nämlich nur am Wert des Attributs interessiert und nicht daran, wie und woher dieser Wert geholt wird

renz speichert. Der Parameter MOinstance dieser Methode setzt sich zusammen aus:

- dem Objektidentifikator des Managementobjekttyps `sysDescr` und
- dem Wert des Attributs indexinfo.

Der Objektidentifikator eines SNMP-Objekttyps steht im Interface Repository und wird aus Effizienzgründen im Schattenobjekt gespeichert. An ihn wird der Zugriffsidentifikator der SNMP-Variable angehängt. Konkret ergibt sich für den Parameter MOinstance: "1.3.6.1.2.1.1.1.0".

- Das snmpserver-Objekt erzeugt daraufhin die `Variable Binding List` für die GetRequest-PDU. Diese packt er in eine SNMP-Message, in der er den vorkonfigurierten Community-String verwendet.
- Die SNMP-Message wird verschickt (Abb. 5.5(3)).
- Aus der zurückkommenden `GetResponse-PDU` (4) extrahiert das snmpserver-Objekt den Wert der SNMP-Variable `sysDescr`, nämlich "SunOS 4.0.1", und gibt ihn dem Schattenobjekt zurück (Rückkehr aus der Methode snmp_get).
- Das Schattenobjekt überprüft, ob irgendwelche Exceptions vom snmpserver-Objekt ausgelöst wurden. Wenn nicht kann das Ergebnis an den Manager zurückgegeben werden.

Für ein zweites Beispiel sei angenommen, daß der Eintrag für `ipRouteNextHop` der ersten Zeile der Routing-Tabelle geändert werden soll. Pakete mit der Zieladresse "128.129.215.13" sollen fortan nicht mehr an die Adresse "128.187.121.15" sondern an die Adresse "131.156.10.176" weitergeleitet werden. Diesen neuen Wert für das Attribut[8] gibt er beim Aufruf der Methode set_IpRouteNextHop (Abb. 5.5(5))des Proxy-Objektes an. (Um herauszufinden, auf welchem Objekt er die Methode aufrufen muß, hat der Manager vorher die Attribute ipRouteDest abgefragt.) Wiederum sorgt die DSOM-Laufzeitumgebung dafür, daß die entsprechende Methode des Schattenobjektes, auf das das Proxy-Objekt zeigt, aufgearbeitet wird (5a). Der Parameter MOinstance für den Aufruf von snmp_set wird auf dieselbe Weise wie oben berechnet:

$$\underbrace{1.3.6.1.2.1.4.21.1.7}_{\text{OID aus dem IR,}}.\underbrace{128.129.215.13}_{\text{aus indexinfo}}$$

Der Parameter value hingegen enthält die neue IP-Adresse "131.156.10.176". Diesmal wird eine vom snmpserver-Objekt eine `SetRequest-PDU` erzeugt und in einer SNMP-Message verschickt (7). Die vom SNMP-Agent zurückgeschickte Response-PDU (8)

[8] Aus Sicht des Managers wird der Attributwert verändert, nicht der Wert einer SNMP-Variable

wird verwendet, um Fehler festzustellen. Angenommen, die SNMP-Variable konnte nicht gesetzt werden, da der SNMP-Agent den schreibenden Zugriff nicht erlaubt hat. In der Response-PDU wird dies durch den Fehler `authorizationError` angezeigt. Das snmpserver-Objekt löst die der Fehlermeldung entsprechende Standard-Exception NO_PERMISSION aus. Dasselbe tut das Schattenobjekt, sobald aus der Methode snmp_set zurückgekehrt wird.

5.4.2 Verwendung von GetNextRequest und GetBulk

Für den schreibenden Zugriff auf SNMP-Agenten kann beim Aufruf der Methode snmp_set() nur eine `SetRequest-PDU` an den Agenten gesendet werden. Für den lesenden Zugriff hingegen stehen neben der `GetRequest-PDU` zwei weitere Protokolldateneinheiten zur Verfügung: `GetNextRequest` und `GetBulk`. Mit ihnen kann effektiver auf SNMP-Ressourcen zugegriffen werden. Der Manager kann beispielsweise alle Attributwerte der Tabellenzeile-Objekte (Abb. 5.5) gleichzeitig abfragen. Um entsprechend den gesamten Inhalt der Tabelle vom SNMP-Agenten zu holen, wird theoretisch nur eine einzige GetBulk-PDU benötigt. Die Schattenobjekte werden aber voneinander unabhängig die Methoden des snmpserver-Objektes aufrufen. Dieses kann aber weder feststellen, daß zwischen angeforderten SNMP-Variablen ein Zusammenhang — alle sind Elemente der Routing-Tabelle — besteht, noch kann es wissen, wie viele von diesen SNMP-Variablen, die es in einer GetBulk-PDU zusammenfassen könnte, von den Schattenobjekten noch angefordert werden. Jeder Methodenaufruf wird einzeln abgearbeitet, was bedeutet, daß mit einer GetRequest-PDU immer nur eine einzige SNMP-Variable abgefragt wird. Folgende Maßnahmen müssen getroffen werden, um den SNMP-Verkehr effektiver zu gestalten:

- Erweiterung der Schattenobjekte, sodaß mehrere (unterschiedliche) Attributzugriffsfunktionen gleichzeitig aufgerufen werden können[9].
- Modifizieren der Parameter der Methode snmp_get() so, daß Schattenobjekte mehr als eine SNMP-Variable übergeben können.
- Im snmpserver-Objekt werden Anforderungen von Schattenobjekten nach Zielagent sortiert „gesammelt“ und mehrere Variablen gleichzeitig in einer `GetRequest-PDU` oder einer `GetBulk-PDU` abgefragt. Es wird also versucht, *angeforderte* SNMP-Variablenwerte mit möglichst wenigen SNMP-PDUs zu erhalten.

Hinweis: Ein `GetNextRequest` kann nur eingeschränkt verwendet werden: Durch eine `GetNextRequest-PDU` werden Informationen vom Agenten zurückgegeben, die nicht dem Schattenobjekt, das die Methode snmp_get() aufgerufen hat, zugeordnet

[9] Bis jetzt war ein Schattenobjekt erst nach der Rückkehr aus einer Attributzugriffsfunktion wieder bereit, Methoden abzuarbeiten.

werden darf. Wenn beispielsweise die Methode get_ipRouteNextHop() des Schattenobjektes, das die erste Zeile der Routing-Tabelle repräsentiert (in Abb. 5.5 das Schattenobjekt mit indexinfo=".128.129.215.13"), aufgerufen wird und daraufhin eine `GetNextRequest-PDU` verschickt wird, so wird der entsprechende Spalteneintrag der nächsten Tabellenzeile, also "131.156.10.176" vom Agenten zurückgegeben. Da mit der `GetNextRequest-PDU` Tabellen traversiert werden können, von denen die Zeilenindizes nicht bekannt sind, wird diese PDU im Gateway nur vom Discovery-Dämon eingesetzt.

5.4.3 Behandlung eines SNMP-Traps

Die Abbildung 5.6 zeigt eine Beispielkonfiguration des Trap-Dämons des Gateway. In diesem DSOM-Serverprogramm befinden sich eine Reihe von Event Channel, von denen jeder für einen bestimmten SNMP-Trap verwendet wird. Die Event Channel werden in einen Naming Graph eingetragen (das ist möglich, da Event Channel selbst Objekte sind), wodurch eine Managementanwendung die Event Channel finden und sich bei diesen zum Empfang von Ereignismeldungen registrieren kann (s. u.).

Das snmptrapd-Objekt, das an Port 162 SNMP-Trap-PDUs empfängt, muß sich bei jedem dieser Event Channel als Supplier anmelden. In Abbildung 5.6 erfolgt dies exemplarisch am Event Channel „coldStart":

- als Supplier ruft das snmptrapd-Objekt die Methode for_suppliers des Event Channels auf (1). Es wird ein SupplierAdmin-Objekt zurückgegeben. Mit diesem Objekt kann sich ein Supplier bei einem Event Channel anmelden (s. 2.3.1, [OMG96]).
- mit dem Aufruf der Methode obtain_push_consumer() auf dem SupplierAdmin-Objekt (2) erhält das snmptrapd-Objekt einen ProxyPushConsumer. Über die push-Methode dieses Objektes werden Ereignismeldungen in den Event Channel geschrieben.

Damit ein Eventconsumer in der Managementanwendung von dem Event Channel „coldStart" Events empfangen kann, muß er sich als Consumer bei diesem registrieren. Die erfolgt analog zur Anmeldung als Supplier:

- der Eventconsumer ruft die Methode for_consumers des Event Channels[10] auf (3); es wird ihm ein ConsumerAdmin-Objekt zurückgegeben.
- Die Methode obtain_push_supplier des ConsumerAdmin-Objekts, die der Eventconsumer daraufhin aufruft (4), gibt einen ProxyPushSupplier zurück.

[10] die Objektreferenz findet er in einem Naming Graph

- Mit dem Aufruf der Methode `connect_push_consumer` dieses ProxyPushSuppliers (5) registriert sich der Eventconsumer beim Event Channel, indem er dem Event Channel eine Objektreferenz auf sich selbst übergibt.

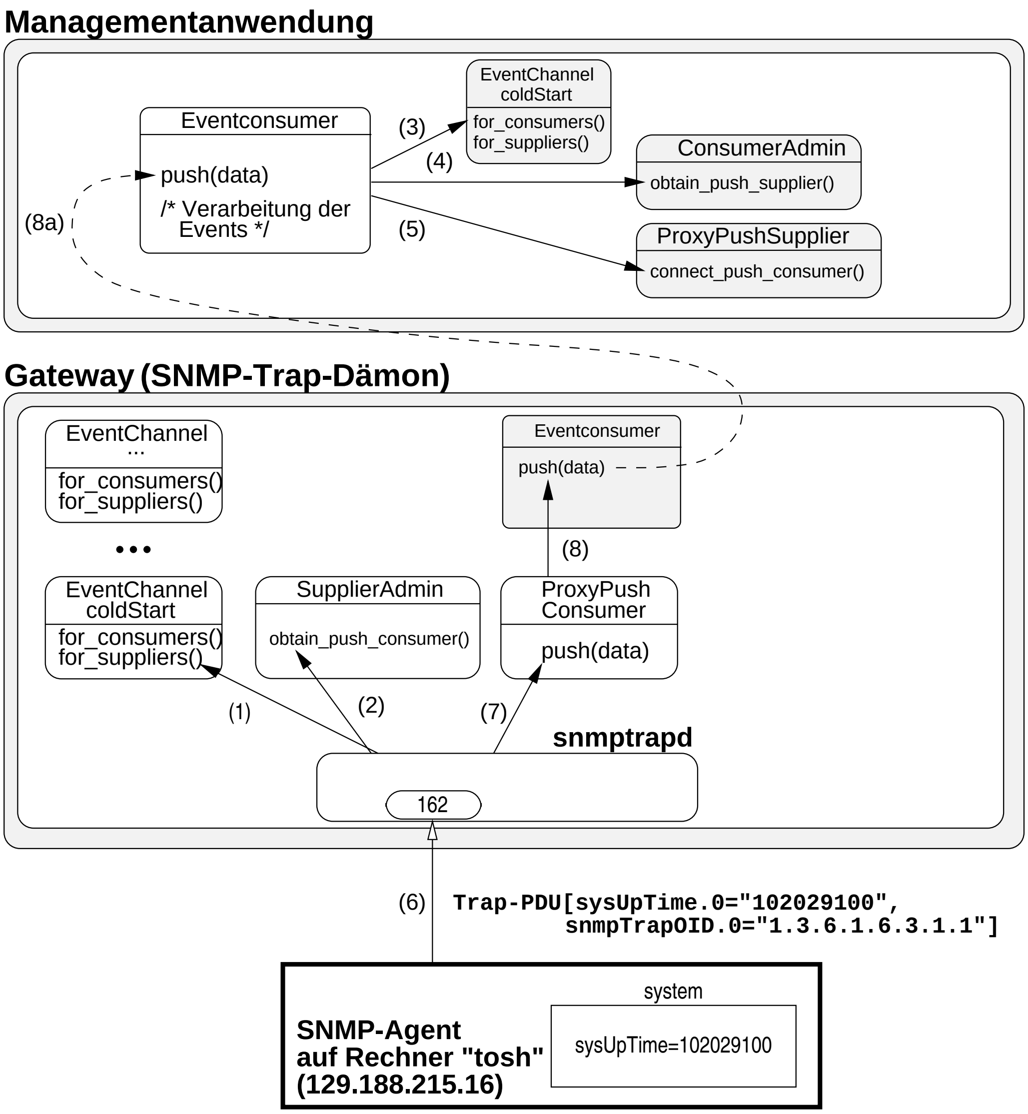

Abbildung 5.6: Beispielkonfiguration des Gateways (Trap-Dämon)

Mit den Schritten (1)–(5) (Abb. 5.6) kann eine Managementanwendung nun den well-known Trap `coldStart` als CORBA-Ereigismeldung empfangen: Das Objekt `snmptrapd` kann anhand der Information in einem ankommenden SNMP-Trap-PDU einen solchen Trap erkennen. Der entsprechende (CORBA-)Event, der für den Beispieltrap „erzeugt" werden muß, besteht im Aufruf der Methode `push` des ProxyPushConsumers des entsprechenden Event Channels (7). Der Parameter data (Datentyp any) enthält dabei Informationen bezüglich des Traps wie z. B. den Zeitpunkt, zu dem der SNMP-Trap erzeugt wurde (sysUpTime.0 = "102029100" [Timeticks]). Das ProxyPushConsumer-Objekt ruft anschließend die Methode push des Proxy-Objektes des Eventconsumers (erhalten bei Registierung des Eventconsumers!) auf (8). Von der DSOM-Laufzeitumgebung wird dieser Aufruf zum eigentlichen Eventconsumer in der Managementanwendung weitergeleitet, welcher die Ereignismeldung verarbeitet. Der Unterschied zu „normalen" Methodenaufrufen ist, daß diese Aufrufe asynchron sind und nichts zurückgegeben wird.

5.5 Der Entwicklungsprozeß

Der gesamte Entwicklungsprozeß ist in Abbildung 5.7 dargestellt. Er besteht aus folgenden Schritten:

1. Übersetzung der ASN.1-Makros in IDL Schnittstellendefinitionen (s. Kap. 3);
2. Manuelle Ergänzung der IDL-Objektdefinitionen um gewünschte oder notwendige Methoden/Attribute. In diesem Schritt können beispielsweise Methoden für die „Pushbutton"-Variablen definiert werden;
3. Definition der Objekte bzw. der Komponenten des Gateways und der benötigten Schnittstellen, ebenso mit IDL;
4. Generierung von Client- und Server-Programmskeletten (in Form von Header-Dateien) in einer gewählten Programmiersprache mittels des SOM IDL-Sprachcompilers;
5. Implementierung der Schnittstellenmethoden.
6. Übersetzen des Sourcecodes mit einem Compiler für die angewandte Implementierungssprache.

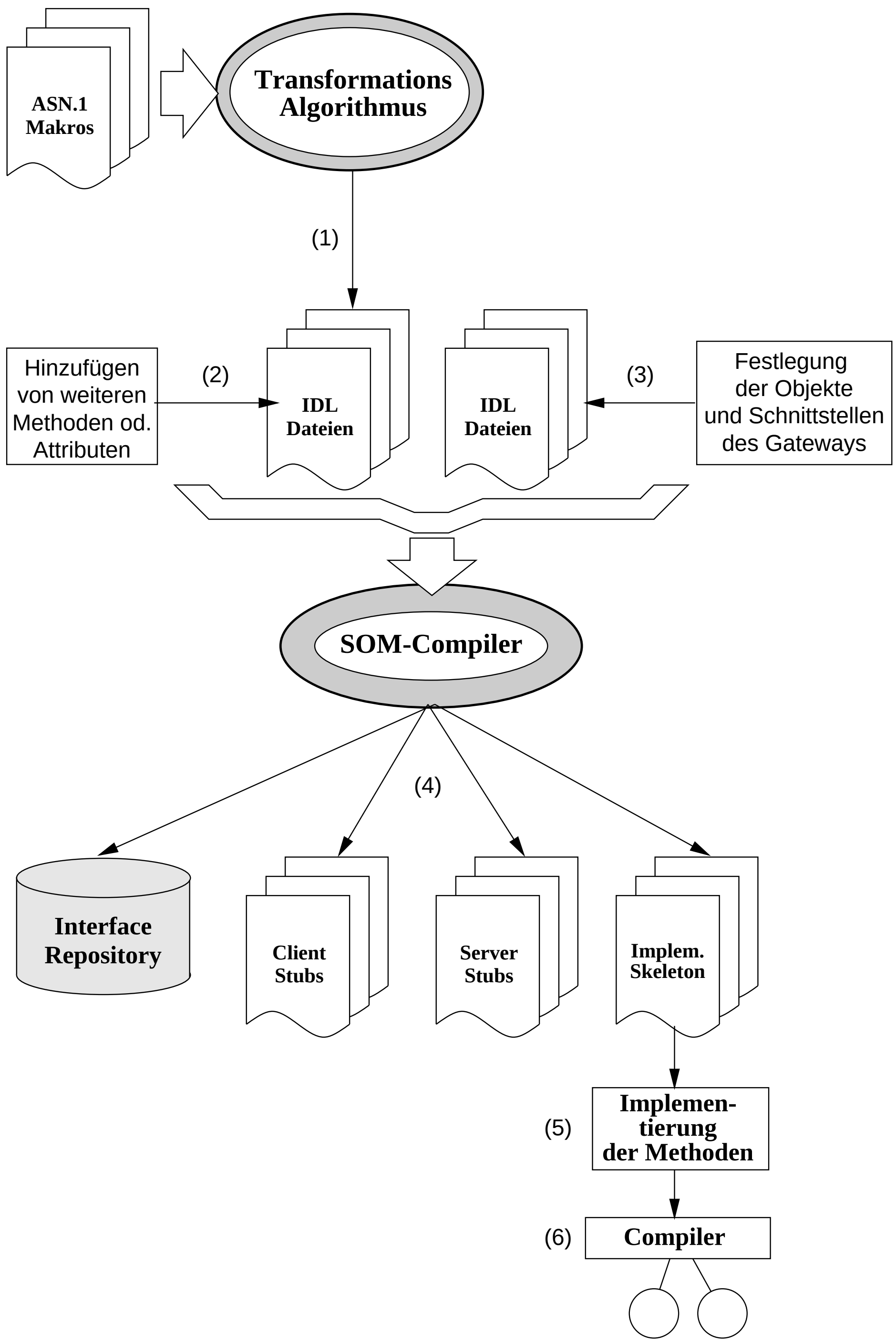

Abbildung 5.7: Der Entwicklungsprozeß

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Auf dem Gebiet des Managements von Netzen und Systemen sind die Managementarchitekturen von OSI und der IETF heutzutage am weitesten verbreitet. Daneben ist mehr und mehr die Tendenz feststellbar, den neuen Ansatz der OMG, die Object Management Architecture (OMA), für das Netz- und Systemmanagement einzusetzen. Zentrales Element der OMA ist die Common Object Request Broker Architecture (CORBA), die Dienste für eine synchrone Kommunikation zwischen Objekten zur Verfügung stellt. CORBA ermöglicht die Interoperabilität von verteilten Anwendungen und wurde von der OMG in der Version 2.0 standardisiert ([OMG95a]).

Alle namhaften Hersteller von Managementsoftware wollen demnächst mit ihren Produkten CORBA-konform sein. Es zeigt sich außerdem an den vielen Forschungs- und Standardisierungsaktivitäten, daß CORBA neben den „traditionellen" Managementarchitekturen vor allem auf dem Gebiet des System- und Anwendungsmanagement möglicherweise eine wichtige Rolle spielen wird. Trotzdem wird sich CORBA — zumindest mittelfristig — kaum vollständig gegenüber dem OSI- und Internet-Management durchsetzen können. Beispielsweise hat sich das Internet-Management aufgrund seiner einfachen und implementierungsfreundlichen Managementlösungen bereits zum De-facto-Standard etabliert und kann aus finanziellen und organisatorischen Gründen nicht einfach kurzfristig „ausgewechselt" werden.

Die Folge ist ein Nebeneinander dieser Managementarchitekturen in einem System. Um dieses System mit vertretbarem Aufwand verwalten zu können, ist es notwendig, Übergänge zwischen den Architekturen zu schaffen, die die Interoperabilität dieser Architekturen gewährleisten. Bei der Einführung eines neuen Ansatzes wie CORBA kann mit derartigen Übergängen eine bestehende Managementarchitektur integriert — und eine bestehende Informationsbasis damit weiterhin verwendet werden. Eine Möglichkeit eines solchen Übergangs ist ein Management-Gateway. Ein Management-Gateway ermöglicht die Einbettung einer Managementarchitektur in eine andere, ohne daß die eingesetzte (CORBA-)Managementanwendung oder existierende (SNMP-)Agenten dafür geändert werden müssen. Ein weiterer Vorteil hat sich gezeigt: Alle CORBA-Dienste zur Verwaltung von Objekten, die in der CORBA-Architektur ein komfortables Management erlauben und die von der Internet-Architektur nicht zur Verfügung stehen, sind auch für das Management von SNMP Managed Objects einsetzbar. Eine CORBA-Managementanwendung arbeitet bequem mit Schattenobjekten, während das Managementgateway alle notwendigen Abbildungen zwischen den beiden Welten durchführt.

In dieser Diplomarbeit wurde ein Konzept für ein CORBA/SNMP-Gateway entworfen. Ein CORBA/SNMP-Gateway ermöglicht es, SNMP-Ressourcen weiterhin, aber von einer CORBA-Umgebung aus, zu verwalten. Es liegt zwischen einer CORBA-Managementanwendung und den Netzressourcen, an der Grenze zwischen den beiden Protokollwelten. Es integriert die Internet-Architektur in die CORBA-Welt, indem es SNMP-Managementobjekte auf CORBA-Objekte abbildet und dadurch einem CORBA-Manager zugänglich macht. Der erste Schritt ist dabei die Übersetzung der Beschreibungssprache ASN.1 in IDL. Der in Kapitel 3 vorgestellte Algorithmus erstellt aus den Objektbeschreibungen einer Agenten-MIB entsprechende IDL-Schnittstellen. Zur Laufzeit erzeugt das Gateway Instanzen dieser Klassen, die die SNMP-Managementobjektinstanzen in CORBA-konformer Form repräsentieren.

Am Anfang des Hauptteils dieser Diplomarbeit, dem 4. Kapitel, wurden die Anforderungen an das Gateway ausgearbeitet. Anschließend wurden verschiedene Interaktionsmöglichkeiten zwischen CORBA-Manager, dem Gateway und den SNMP-Agenten besprochen und bewertet. Auf diese Weise ergab sich das Konzept für das CORBA/SNMP-Gateway. In Kapitel 4 wird außerdem behandelt, inwieweit die Object Services von CORBA innerhalb des Gateway eingesetzt werden können.

Kapitel 5 beschreibt das Entwicklungstoolkit SOMobjects von IBM. Es wird gezeigt, inwieweit die Werkzeuge dieser Entwicklungsumgebung CORBA-konform sind und wie die Realisierung des Gateways damit prinzipiell vor sich geht. Eine Implementierung des Konzeptes wurde aus zeitlichen Gründen jedoch nicht vorgenommen.

6.2 Ausblick

Ein wichtiger nächster Schritt ist es festzustellen, wie sich das Konzept im praktischen Einsatz tatsächlich bewährt. Dazu muß auf dieser Basis zunächst ein Prototyp des Gateways implementiert werden. Daran läßt sich das Verhalten des Gateways beobachten. Die in der Praxis gewonnenen Erkenntnisse können in die fortschreitende Realisierung wieder einfließen.

Auch wenn das vorgeschlagene Konzept bereits eine gute Grundlage für eine Realisierung bietet, kann und muß es an manchen Stellen erweitert werden. Beispielsweise wurden Sicherheitsaspekte und Fehlerbehandlung im Gateway nur ansatzweise besprochen. Die Kommunikation zwischen dem Gateway und SNMP-Managern bedarf ebenso genauerer Betrachtung.

Weitere interessante Fragestellungen betreffen die Erweiterung der Funktionalität der Gatewaykomponenten und der Schattenobjekte. Welche Tätigkeiten können beispielsweise im Gateway oder in den Schattenobjekten erledigt werden, um die Managementanwendung zu entlasten? Wie können SNMP-Agenten, von denen keine IDL-Klassen definiert wurden, dynamisch in die CORBA-Umgebung abgebildet werden? Ist es beispielsweise möglich, für Agenten-MIBs, welche nicht auf IDL-Schnittstellendefinitionen abgebildet wurden, zur Laufzeit generische Schattenobjekte zu instantiieren? Wie müssen die Klassen für diese Schattenobjekte definiert werden? Diese Fragen sollen nur Anregungen geben, in welche Richtung das Gateway vertieft und erweitert werden kann. Sie können eventuell in weiteren Diplomarbeiten bearbeitet werden.

Literaturverzeichnis

[ACH93] S. Abeck, A. Clemm, and U. Hollberg. Simply Open Network Management: An Approach for the Integration of SNMP into OSI Management Concepts. *IFIP, Integrated Network Management III (C-12)*, 1993. H.-G. Hegering and Y. Yemini (Editors).

[CMRW96a] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework. Request for Comments (Standards Track) RFC 1908, Internet Engineering Task Force, January 1996.

[CMRW96b] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2). Request for Comments (Standards Track) RFC 1905, Internet Engineering Task Force, January 1996.

[CMRW96c] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2). Request for Comments (Standards Track) RFC 1902, Internet Engineering Task Force, January 1996.

[Gei92] K. Geihs. OMG Object Request Broker. *PIK 15*, April 1992. K. G. Saur Verlag.

[HA93] H.-G. Hegering and S. Abeck. *Integriertes Netz- und Systemmanagement.* Addison–Wesley Publ., 1993.

[HAB91] H.-G. Hegering, Sebastian Abeck, and Thorsten Böhnke. Converting MIB Descriptions into MIB Implementations. *IFIP/IEEE, International Workshop on Distributed Systems: Operations and Management*, October 1991.

[IBM96a] IBM. *SOMobjects Developer Toolkit Programmer's Guide, Volume I: SOM and DSOM*, first edition, March 1996. SOMobjects Version 3.0.

[IBM96b] IBM. *SOMobjects Developer Toolkit Programmer's Guide, Volume II: Object Services*, first edition, March 1996. SOMobjects Version 3.0.

[JID95] JIDM. Inter-Domain Management Specifications: Specification Translation. Preliminary specification, X/Open Joint Inter-Domain Management Group, August 1995. Sanity Check Draft.

[KS93] Pramod Kalyanasundaram and Adarshpal S. Sethi. An Application Gateway Design for OSI-Internet Management. *IFIP, Integrated Network Management III (C-12)*, 1993. H.-G. Hegering and Y. Yemini (Editors).

[MBL93] Subrata Mazumdar, Stephen Brady, and David W. Levine. Design of Protocol Independent Management Agent to Support SNMP and CMIP Queries. *Third International Symposium on Integrated Network Management*, May 1993.

[MR90] K. McCloghrie and M. Rose. Structure and Identification of Management Information for TCP/IP-based Internets. Request for Comments (Standard) STD 17, RFC 1155, Internet Engineering Task Force, May 1990.

[MR91] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based Internets: MIB-II. Request for Comments (Standard) STD 17, RFC 1213, Internet Engineering Task Force, March 1991. (Obsoletes RFC1158).

[OMG92] Object Management Group OMG. *Object Management Architekture Guide*. John Wiley & Sons, Inc., second edition, September 1992. OMG TC Document 92-11-1.

[OMG95a] Object Management Group OMG. *CORBA 2.0/Interoperability - Universal Networked Objects*. OMG, March 1995. OMG TC Document 95.3.xx.

[OMG95b] Object Management Group OMG. Object Services RFP 3. Request for proposal, OMG, February 1995. OMG TC Document 94-7-1.

[OMG95c] Object Management Group OMG. Object Services RFP 4. Request for proposal, OMG, February 1995. OMG TC Document 94-4-18.

[OMG96] Object Management Group OMG. *CORBAservices: Common Object Services Specification (COSS)*. OMG, second edition, March 1996.

[Rö93] M. Rösch. Object Request Broker - Funktionsweise und erste Erfahrungen. *unix/mail*, November 1993. Carl Hanser Verlag.

[Sou94] Nader Soukouti. Using CORBA to manage OSI. *Draft 1.0*, November 1994.

[Sou95] Nader Soukouti. Preliminary CORBA/SNMP Interaction Translation Architecture. *JIDM Task Force*, September 1995.

[Vog96] Thorsten Vogs. Entwurf und Implementierung eines Konzeptes zur Anbindung von Objekt Request Brokern an eine Netzmanagementplattform. Master's thesis, Technische Universität München, February 1996.

[Wal95] S. Waldbusser. Remote Network Monitoring Management Information Base (RMON-MIB). Request for Comments (Draft Standard) RFC 1757, Internet Engineering Task Force, February 1995. (Obsoletes RFC1271).

[X/O95] X/Open. Systems Management: Common Management Facilities. Preliminary Specification (for OMG RFC Review), X/Open Company Ltd., December 1995.