Proceedings of the *GI-Fachtagung Kommunikation in Verteilten Systemen (KiVS) 97*,
Braunschweig, Germany, February 1997

Ein werkzeugunterstützter Ansatz zur Gewinnung CORBA-konformer Managementagenten aus modularen SNMP-Implementierungen

Alexander Keller*
Fakultät für Informatik, Technische Universität München
c/o Institut für Informatik der
Ludwig Maximilians Universität München
Oettingenstr. 67, 80538 München
E-Mail: keller@informatik.uni-muenchen.de

Zusammenfassung

Mit der zunehmenden Verbreitung von CORBA für die Implementierung verteilter Anwendungen ergibt sich die Möglichkeit, diese objektorientierte Kommunikationsarchitektur auch zum Management dieser Applikationen und der Systeme, auf denen die verteilten Anwendungen laufen, einzusetzen. Es ist daher sinnvoll, CORBA-konforme Managementagenten bereitzustellen, die die Überwachung und Steuerung dieser Systeme auf effiziente Art gewährleisten. Während einerseits gegenwärtig keine solchen CORBA-konformen Agenten existieren, gibt es andererseits SNMP-Agenten, die diesen Zweck erfüllen. Gesucht ist daher ein Verfahren zur Gewinnung CORBA-konformer Managementagenten aus bestehenden SNMP-Implementierungen. Der Beitrag beschreibt ein Vorgehensmodell zur Migration bestehenden modularen SNMP-Agentencodes in eine CORBA-Umgebung und schildert dessen konkrete Anwendung anhand eines praxisnahen Beispiels. Es handelt sich hierbei um einen Agenten für das Management von UNIX-Endsystemen, der am Lehrstuhl entwickelt wurde und in der Praxis zum Einsatz kommt. Weiterhin wird skizziert, wie der Transformationsaufwand durch die Abstützung auf standardisierte Architekturen und Verfahren sowie am Markt erhältliche Werkzeuge in akzeptablen Grenzen gehalten werden kann.

1 Einführung

Durch die zunehmende Verbreitung vernetzter arbeitsteiliger Systeme sehen sich die Betreiber großer verteilter IV-Infrastrukturen steigenden Anforderungen an die Güte der von ihnen angebotenen Dienste ausgesetzt. Diese Situation wird durch das Vorhandensein stark heterogener Endbenutzersysteme sowie den in ihnen enthaltenen Objekten wie Anwendungsprogrammen, Betriebssystemen und Kommunikationsprotokollen noch weiter verschärft. Demgegenüber resultiert aus der hohen Konkurrenzsituation zwischen den Betreibern und dem dadurch entstehenden Kostendruck der Zwang, die Anzahl hochqualifizierten Personals zur Überwachung und Steuerung dieser interagierenden Komponenten zu reduzieren.
Der Brisanz der Situation, in der sich die Betreiber befinden, kann jedoch geeignet entgegengetreten werden, indem ihnen Werkzeuge zur Verfügung gestellt werden, die die zugrundeliegende Heterogenität verschatten und es ihnen somit ermöglichen, eine einheitliche Sicht auf die verteilten Systeme zu erhalten. Standardisierte Managementarchitekturen sind ein geeignetes Mittel, um dieses Ziel zu erreichen. Beispiele hierfür sind das ISO/OSI–Management Framework oder die Internet–Managementarchitektur (häufig auch als SNMP–Management bezeichnet), die jedoch per se nicht interoperabel sind.

*Diese Arbeiten wurden gefördert durch das IBM European Networking Center, Heidelberg

Beide Architekturen haben allerdings Gemeinsamkeiten, die unter anderem der Grund dafür sind, daß sich bis zum heutigen Tage keine dieser beiden Alternativen vollständig am Markt durchsetzen konnte: für die Beschreibung der zu steuernden Ressourcen wird ein eigenständiges Informationsmodell und damit eine spezielle Beschreibungssprache verwendet; zur Kommunikation zwischen Managementsystem und der zu überwachenden Infrastruktur existieren dedizierte Managementprotokolle. Die Handhabung dieser durch das Management zusätzlich eingeführten Heterogenität erweist sich im praktischen Betrieb jedoch oft als hinderlich und führt insbesondere dazu, daß die Hersteller einer verteilten Anwendung und der entsprechenden Managementapplikation nur in den seltensten Fällen identisch sind. Die Anforderungen der Betreiber an Managementsysteme werden daher in marktfähigen Produkten oft nur unzureichend berücksichtigt.
Auch im Bereich der Entwicklung von Managementsystemen macht sich die Fixierung auf eine spezielle Beschreibungsmethodik und ein eigenständiges Managementprotokoll bemerkbar: Allgemein verfügbare Modellierungs- und Implementierungswerkzeuge können entweder nicht oder nur mit hohem Anpassungsaufwand eingesetzt werden, da die Notation zur Beschreibung von Managementinformation oft nicht von den Herstellern dieser Werkzeuge unterstützt wird.
Demgegenüber verfolgt die im Rahmen der *Object Management Architecture (OMA)* von der *Object Management Group (OMG)* standardisierte *Common Object Request Broker Architecture (CORBA)* ([6]), die ursprünglich für verteilte objektorientierte Programmierung konzipiert wurde, einen anderen Ansatz: man fordert, daß *eine einzige* Architektur sowohl für die Entwicklung als auch für das Management einer verteilten Anwendung verwendet wird. Dies bedeutet, daß nicht nur die Beschreibung sowohl der Management- als auch der Anwendungsobjekte in einer identischen Notation (OMG IDL[1]) erfolgt, sondern ebenfalls Nutz- und Managementdaten einer Applikation mit demselben Kommunikationsmittel (einem sogenannten *Object Request Broker (ORB)*) übertragen werden. Management wird somit zu einem (zweifellos wichtigen) Teilaspekt verteilter Anwendungen. Somit kann das gesamte Spektrum verfügbarer Werkzeuge zur Softwareentwicklung genutzt werden, da Nutz- und Managementdaten identisch modelliert und implementiert werden.
Konzeptionelle Untersuchungen bezüglich der Mächtigkeit des CORBA-Objektmodells im Vergleich zu den Informationsmodellen etablierter Managementarchitekturen haben die prinzipielle Eignung von CORBA für das Management von Endsystemen und der auf ihnen ablaufenden Anwendungen nachgewiesen (siehe [12]). Einerseits sind aufgrund des geringen Alters von CORBA[2] momentan gerade erste Implementierungen auf dem Markt erhältlich; im Bereich des Managements sind derzeit noch keinerlei CORBA-konforme Managementsysteme und -agenten bekannt. Auf Seiten der Hersteller ist jedoch zunehmend die Tendenz erkennbar, solche Werkzeuge zu entwickeln ([15], [7]). Andererseits existiert gegenwärtig eine große Zahl an SNMP-fähigen Managementagenten und es stellt sich daher die Frage, wie bestehender SNMP-Agentencode nahtlos in die CORBA-Welt migriert werden kann. Ein so entstehender CORBA-Managementagent besteht aus verteilten kooperativen Managementobjekten, die mit Hilfe des Object Request Brokers interagieren.
Der Beitrag präsentiert ein Vorgehensmodell zur Migration bestehenden modularen SNMP–Agentencodes in eine CORBA–Umgebung und schildert die Schritte dessen konkreter Anwendung anhand eines praxisnahen Beispiels. Seine Struktur orientiert sich an dem in Abbildung 1 skizzierten Vorgehensmodell: Abschnitt 2 beschreibt die Eigenschaften des am Lehrstuhl entwickelten SNMP-Agenten zum Management von UNIX-Workstations und den JIDM-Algorithmus (siehe 2.2) zur Umsetzung des Internet-Informationsmodells in CORBA-Objektbeschreibungen. Das Ergebnis ist ein algorithmisch erzeugtes Objektmodell, das jedoch noch verbessert werden muß, um den Anforderungen gerecht zu werden. Die Gründe dafür sowie die Schritte und Werkzeuge zur Optimierung des vorhandenen Objektmodells sind in

[1]Die Interface Definition Language (IDL) ist die von der OMG standardisierte Beschreibungssprache für die Schnittstellen von CORBA-Objekten

[2]Der aktuelle Standard [2], der in der Version 2.0 vorliegt, wurde im Dezember 1994 verabschiedet

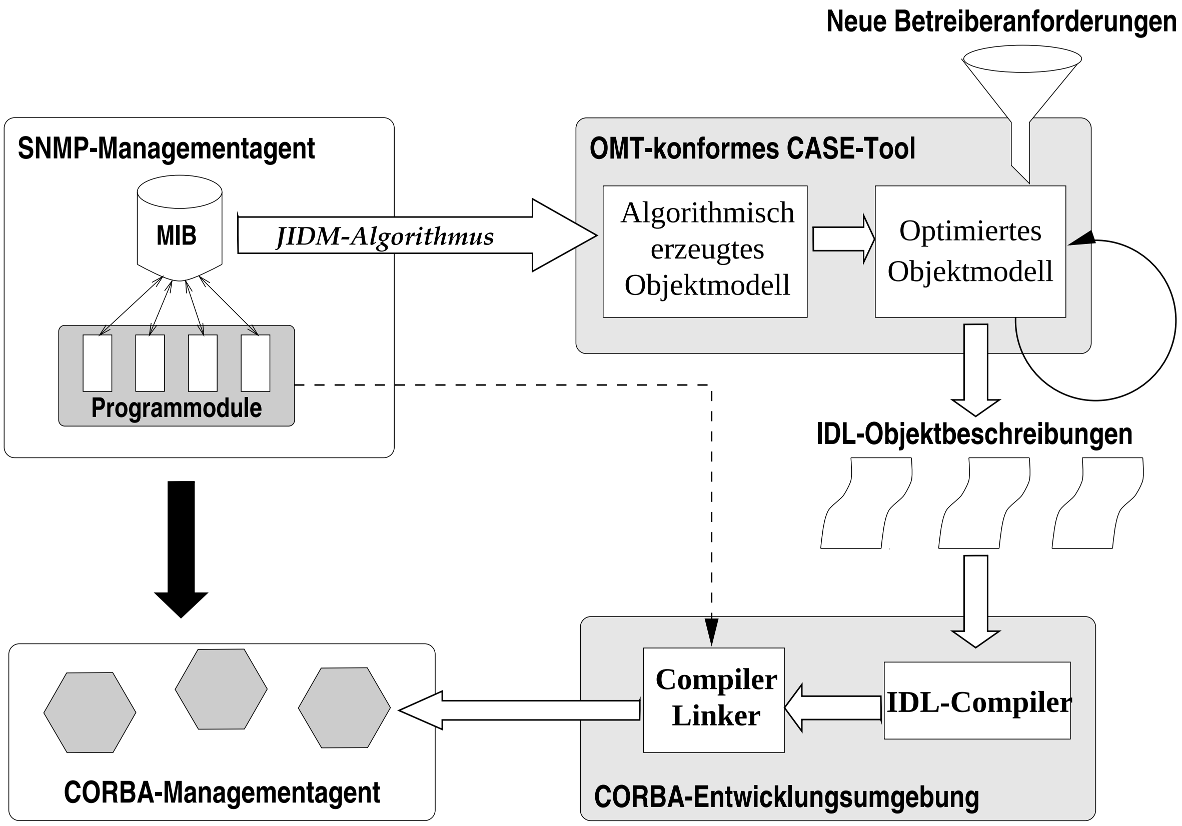

Abbildung 1: Vorgehensweise bei der Portierung des Agenten

Abschnitt 3 beschrieben. Hierbei ist es von großer Bedeutung, auf Werkzeuge wie CASE-Tools zurückgreifen zu können, die den zyklischen Prozeß der objektorientierten Analyse und des Designs geeignet unterstützen. Sie schaffen auch die Grundlage, um das bestehende Modell um neue Anforderungen von Seiten der Betreiber nahtlos zu erweitern. Nachdem die Modellierung abgeschlossen ist, kann die Implementierung erfolgen, die in Abschnitt 4 dargestellt wird. Hierbei hat sich bei unseren Arbeiten herausgestellt, daß die Schnittstellen zwischen den einzelnen Werkzeugen gegenwärtig noch verbesserungsbedürftig sind und man von einem reibungslosen Zusammenspiel aller am Entwicklungsprozeß beteiligten Komponenten noch etwas entfernt ist.

2 Ausgangssituation

2.1 Ein SNMP-Agent für das Management von UNIX-Workstations

Zum heutigen Zeitpunkt muß trotz zahlreicher Bemühungen der Hersteller die Problemstellung, integriertes Management von UNIX-Workstations sicherzustellen, noch immer als ungelöst betrachtet werden: Es existieren zwar Werkzeuge, die eine Vielzahl von Parametern eines Endsystems erfassen und mit Hilfe eines standardisierten Managementprotokolls an die Managementplattform weiterleiten ([5]); aktive Eingriffsmöglichkeiten durch den Administrator fehlen jedoch völlig. Andererseits existieren Produkte, die zwar Eingriffe zur Steuerung des Systems erlauben; die Kommunikation mit der Managementplattform geschieht jedoch lediglich auf der Grundlage eines herstellerspezifischen, proprietären Protokolls, dessen Schnittstellen nicht offengelegt sind ([8]). Zusätzlich sind beide Ansätze von einer Bottom-Up-Vorgehensweise geprägt, die nur selten die tatsächlichen Bedürfnisse der Netzbetreiber berücksichtigt. Von einer umfassenden, integrierten Managementlösung ist man daher noch weit entfernt.

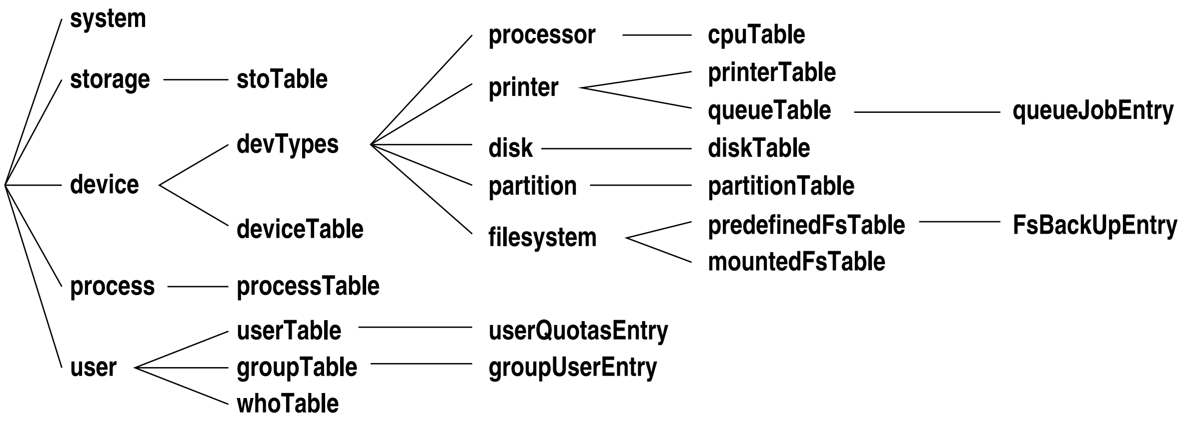

Abbildung 2: Eine MIB für das Management von UNIX-Workstations (nach [3])

Im Gegensatz zu kommerziell erhältlichen Werkzeugen haben wir in unseren Arbeiten [3] einen Top-Down-Ansatz verfolgt, der das typische Aufgabenspektrum eines UNIX-Systemadministrators abdeckt. Es wurden daher Möglichkeiten für das Einrichten und Löschen von Benutzerkennungen und -gruppen sowie deren Quoten für den Zugriff auf Betriebsmittel (Plattenplatz, Druckseiten) ebenso vorgesehen, wie Funktionen zum Erfassen und ggf. Stoppen der momentan aktiven Prozesse oder zum Mounten/Unmounten von Dateisystemen. Es ist somit nicht nur passives Monitoring von UNIX-Workstations möglich, sondern auch das aktive Eingreifen in den Betrieb des Systems. Als Managementprotokoll wird SNMP in der Version 2 verwendet.

Im Rahmen dieser Arbeiten wurde eine Systemmanagement-MIB entwickelt und der dazugehörige Agent auf verschiedenen Betriebssystemplattformen (IBM AIX, HP-UX, SunOS, Solaris) implementiert. Diese MIB ist in Abbildung 2 schematisch dargestellt; sie umfaßt 195 MIB-Variablen und 15 Tabellen und stellt (unter anderem) ein Modell folgender Komponenten eines UNIX-Systems zur Verfügung:

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

Im Falle der Benutzer- und Gruppenverwaltung tritt jedoch durch eine Einschränkung des Internet-Informationsmodells unweigerlich folgendes Problem auf: in der Regel sind auf einem UNIX-System mehrere Benutzer eingetragen, die ihrerseits wieder zu mehreren Gruppen gehören. Tabellen innerhalb von Tabellen sind jedoch explizit durch den entsprechenden Internet-Standard [1] untersagt. „Zu den Internet-Standards konforme“ Managementsysteme akzeptieren jedoch MIBs, die diese Anomalie aufweisen, klaglos.

Als sehr vorteilhaft für die Kapselung des SNMP-Agentencodes hat sich der modulare Aufbau des SNMP-Agenten erwiesen: Zum Zugriff auf die MIB-Variablen wurde jeweils eine Prozedur verwendet. Dies bedeutet, daß jede MIB-Variable in einem eigenen Modul implementiert worden ist und über eine bzw. zwei Schnittstellen verfügt, je nachdem, ob auf die Variable nur lesend oder auch schreibend zugegriffen werden kann. Die ausschließlich lesbare MIB-Variable `sysName` wird durch Aufruf der Prozedur get_sysName ermittelt; eine schreib- und lesbare Variable wie `sysLocation` wird durch die Prozeduren get_sysLocation und set_sysLocation implementiert. Der SNMP-Agent besteht daher insgesamt aus 195 get- und 150 set-Prozeduren. Der ursprüngliche Grund für diese Modularisierung lag in der besseren Erreichbarkeit von

Plattformunabhängigkeit, da man zwischen Betriebssystem-spezifischen Systemaufrufen und Aufrufen, die für alle unterstützten Betriebssysteme identisch sind, besser unterscheiden konnte.

2.2 Algorithmus zur Umsetzung von SNMP-MIBs in das CORBA-Objektmodell

Die Abbildung bestehender Objektbeschreibungen in das CORBA–Objektmodell bedingt die algorithmische Überführung der Spezifikationssprachen, in denen die Managementobjekte beschrieben sind. In unserem Fall handelt es sich um eine Übersetzung der in der Internet–Managementarchitektur verwendeten ASN.1–Templatesprache in die OMG *Interface Definition Language (IDL)*. Für diesen Zweck existiert ein Algorithmus [16], der im Rahmen der Aktivitäten der *Joint X/Open NM-Forum Inter-Domain Management Task Force (JIDM)* entworfen wurde und sich zur Zeit in der Standardisierungsphase befindet. Wichtigster Gesichtspunkt bei der Überführung von Agenten einer Managementarchitektur in eine andere Architektur ist die unterschiedliche Mächtigkeit der jeweiligen Informationsmodelle. Während das Internet-Informationsmodell sämtliche Aspekte eines Agenten (Parameter und Aktionen) in Form von skalaren Datentypen bzw. Tabellen beschreibt und – im Gegensatz zu CORBA – keine Mechanismen wie Vererbung und Polymorphie kennt, werden im CORBA-Objektmodell Agenten in Form von Objektklassen sowie deren zugehörigen Attributen und Methoden definiert. Die Transformation der in SNMPv2 vorhandenen Datentypen, Makros und asynchronen Ereignismeldungen in die CORBA–Entsprechungen geschieht folgendermaßen:

- Für jede Gruppe einer SNMP–MIB wird eine Objektklasse erzeugt; die darin enthaltenen einfachen skalaren Datentypen werden zu Attributen der Objektklasse. So werden zum Beispiel `Integer32` bzw. `TimeTicks` auf die IDL–Datentypen `long` bzw. `unsigned long` abgebildet; `DisplayString` und `IpAddress` werden einer Sequenz von Octets zugeordnet.
- SNMP–Tabellen werden durch den Algorithmus ebenfalls zu Objektklassen transformiert: Jede Zeile einer Tabelle wird damit zu einer Instanz einer Objektklasse, die durch eine IDL–Schnittstelle festgelegt ist. Ein Beispiel soll dies erläutern: Enthält ein System drei Festplatten, so wird dies auf der SNMP–Seite durch das Anlegen von drei Zeilen der Tabelle `stoTable` dargestellt. CORBA–seitig würden stattdessen drei Instanzen der Objektklasse `StorageDevice` erzeugt. Das CORBA-Objektmodell ist hierbei dem intuitiven Verständnis näher als das entsprechende Internet-Informationsmodell.
- Variablen, die einzelne Felder der Tabelle spezifizieren, werden im Falle einfacher Datentypen zu Attributen der entsprechenden Objektklasse.
- SNMP-Traps werden in CORBA-Events, asynchrone Ereignismeldungen, transformiert, die durch den *Object Event Service* ([9]) bereitgestellt werden.

Obwohl mit dem Übersetzungsalgorithmus dem Entwickler ein mächtiges Werkzeug zur syntaktischen Überführung von SNMP–Objektbeschreibungen in das CORBA–Objektmodell zur Verfügung steht, sind manuelle Eingriffe erforderlich, da Managementsemantik im Internet-Informationsmodell oft nur implizit definiert ist: So wird das Auslösen von Aktionen auf Managementobjekten, für das in der Internet-Managementarchitektur kein explizites Sprachmittel vorgesehen ist, durch das Setzen entsprechender MIB–Variablen (sogenannter *Pushbutton*–Variablen) simuliert. Das CORBA–Analogon hierzu besteht im Aufruf einer Methode auf dem jeweiligen Managementobjekt. Der Übersetzungsalgorithmus, der die Abbildung der SNMP–Datentypen in äquivalente IDL–Konstrukte vornimmt, müßte also eine Pushbutton–Variable auf eine Methode des Managementobjektes abbilden. Da Pushbutton–Variablen jedoch nicht syntaktisch erkennbar sind, muß diese Transformation manuell durch den Entwickler erfolgen. Eine MIB-Variable `stoFormat`, deren Setzen die Formatierung einer Festplatte veranlaßt, muß daher manuell auf eine Methode `storage_format` der Objektklasse `Storage` abgebildet werden, da sie ansonsten durch algorithmische Umsetzung zu einem einfachen Attribut würde.

2.3 Ergebnis der algorithmischen Transformation: Ein erstes Objektmodell

Der im vorigen Teilabschnitt beschriebene JIDM–Algorithmus bildet die Grundlage für die Gewinnung einer ersten Version des Objektmodells von UNIX-Endsystemen. Hierzu sind folgende Anmerkungen zu machen:
1. Der JIDM–Algorithmus dient in erster Linie dazu, Managementinformation für Managementgateways bereitzustellen; er soll einem CORBA–basierten Manager ermöglichen, SNMP–konforme Managementagenten durch ein dazwischen liegendes CORBA/SNMP–Managementgateway derart zu steuern, daß der Manager die SNMP–Agenten als CORBA–Agenten sieht. Dieses Szenario impliziert unter anderem, daß sämtliche zur Verwaltung der SNMP–Information notwendige Daten wie die Indizes der Tabellenzeilen in Attribute der CORBA–Objektklassen umgewandelt werden müssen. Während dies für Managementgateways zweifellos notwendig ist, ist ein solches Vorgehen für das objektorientierte Design des Agenten nicht wünschenswert, da beispielsweise die Indizes von SNMP–Tabellenzeilen bereits implizit in den Instanzenidentifikatoren der entsprechenden CORBA–Objektklassen enthalten sind.

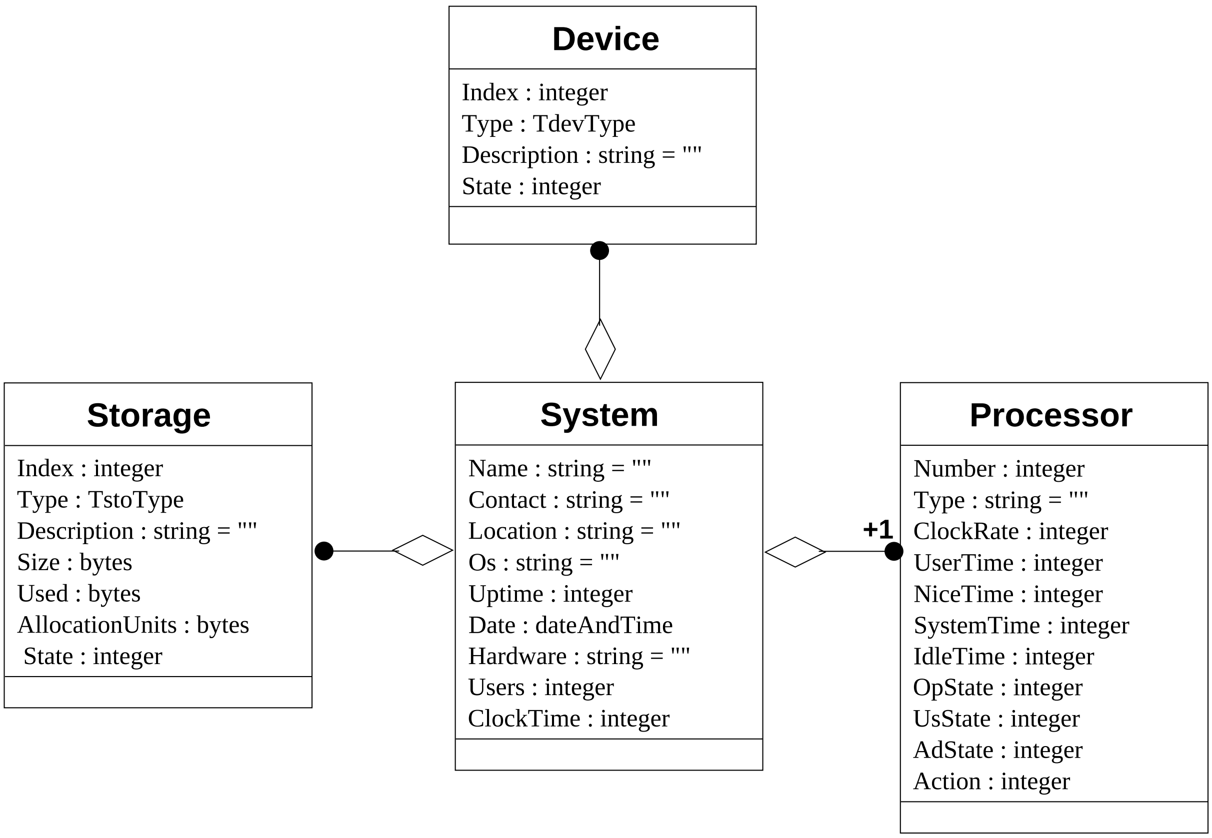

Abbildung 3: Algorithmisch erzeugtes Objektmodell in OMT-Notation (Teilansicht)

2. Die Zielsprache des JIDM–Algorithmus ist, wie oben erwähnt, OMG IDL. Für das weitere Design und die Erweiterung des UNIX-Managementagenten ist es jedoch zwingend erforderlich, die gewonnenen Objektbeschreibungen in eine Notation zu überführen, die es gestattet, die Beschreibungen mit kommerziellen CASE-Tools nachzubearbeiten. Das uns zur Verfügung stehende Werkzeug (beschrieben in Abschnitt 3.1) verwendet die *Object Modeling Technique (OMT)* nach Rumbaugh et al [11]. Die Umsetzung der IDL–Syntax in OMT mußte daher von Hand erfolgen. Dies war jedoch trotz des beachtlichen Umfangs der zugrundeliegenden MIB innerhalb kürzester Zeit machbar, da die Struktur und die Datentypen der Objektklassen bzw. deren Attribute bereits durch den JIDM–Algorithmus festgelegt waren.

Ein weiteres Problem bestand darin, daß das Internet–Informationsmodell keine Angaben über die Kardinalität der Beziehungen zwischen einzelnen Objektklassen vorsieht, da es zwar objekt*basiert* ist, keinesfalls jedoch objekt*orientiert*. Demzufolge fehlen Angaben bezüglich vorhandener Enthaltenseinsbeziehungen vollständig, wurden jedoch bereits bei der Überführung in die OMT–Notation von Hand eingefügt. Abbildung 3 stellt einen Ausschnitt aus dem so gewonnenen Objektmodell dar.

3 Gewinnung eines geeigneten Objektmodells

Zusätzlich zu den im vorigen Abschnitt angesprochenen Konvertierungen gibt es natürlich noch eine Reihe offener und für das Design des CORBA–Agenten substantielle Kritikpunkte, die nachfolgend aufgezählt sind. Es sei an dieser Stelle daran erinnert, daß das Ziel der Arbeit darin besteht, einen Agenten zu erhalten, der *möglichst vollständig* objektorientierten Prinzipien entspricht.

1. Zwischen den einzelnen Klassen fehlt (bis auf die Enthaltenseinsbeziehung zur Systemklasse) jegliche Hierarchie

Eine Enthaltenseinshierarchie ist nur ansatzweise (eben mit der Systemklasse) realisiert, eine Vererbungshierarchie fehlt gänzlich. Damit sind zwei wesentliche Möglichkeiten der Objektorientierung noch nicht ausgeschöpft. Polymorphismus wird damit zwangsläufig auch nicht unterstützt.

Dies sind unmittelbare Konsequenzen aus dem Internet–Informationsmodell, das keinerlei Beziehungen zwischen Objektklassen (weder Vererbung noch Enthaltensein) kennt.

2. Die noch vorhandenen Typvariablen widersprechen objektorientierten Grundsätzen

Beispiele für solche Typvariablen in obigen MIB-Ausschnitten sind `Type` in der `Storage`-Klasse oder `Type` in der `Device`-Klasse. Damit ist es zwar möglich, verschiedene Arten der Objektklasse `Storage` (durch entsprechendes Setzen der Typvariablen) zu bilden, die Objektstruktur wäre aber für die unterschiedlichen Arten von Speicherobjekten (Hauptspeicher, Festplatten, Magnetbänder, Disketten) dieselbe gewesen; die stark differierenden Eigenschaften der einzelnen Typen werden somit nicht berücksichtigt.

Hier wird das Fehlen einer Vererbungshierarchie im Internet–Informationsmodell deutlich.

3. Die Problematik der Pushbutton-Variablen bleibt bestehen

Das Problem, daß die Wertzuweisung an eine Variable unmittelbar eine Operation auslöst, ist auch in diesem ersten Objektmodell vorhanden.

Dies resultiert aus der Tatsache, daß SNMP keine *Action*–Protokolldateneinheit kennt.

4. Die Variablentypen beschränken sich nur auf ASN.1-Grundtypen

Auch Variablen mit stark eingeschränktem Wertebereich, wie z.B. `OpState`[3] oder `AdState`[4] in der Klasse `Processor`, besitzen im ersten Objektmodell einen einfachen integer-Datentyp.

Während dieser Punkt auf den ersten Blick als ein „kosmetisches" Problem erscheint, so sind Überlegungen bezüglich der Einschränkung der Wertebereiche von Attributen hinsichtlich späterer Konformitätstests wichtig.

Das objektorientierte Prinzip der Kapselung (*Encapsulation*) ist in diesem Stadium bereits berücksichtigt:

Wäre diese erste Version als Basis für die Implementierung herangezogen worden, so hätte der IDL-Compiler Rahmendateien erzeugt, in denen die einzelnen Attribute als private gekennzeichnet, also nur über spezielle get-/set-Operationen (deren Rahmen ebenfalls erzeugt würden) zugreifbar gewesen wären.

Es ist somit nicht möglich, auf die Attribute über andere als die bei der Implementierung vorgesehenen Wege zuzugreifen.

[3] `OpState` (Operational State) kann die Werte 1 (enabled) oder 2 (disabled) annehmen; ein Aufzählungstyp reicht hier aus.

[4] `AdState` (Administrative State) kann die Werte 1 (unlocked), 2 (locked) oder 3 (shutting down) annehmen; auch hierfür ist ein Aufzählungstyp besser geeignet.

3.1 Werkzeugunterstützung

Die im vorangehenden Abschnitt beschriebenen Mängel implizieren massive Eingriffe in die Struktur des ersten Objektmodells. Ebenso sollten zusätzliche neue Anforderungen von Seiten des Betreibers in die Konzeption des verbesserten Objektmodells einfließen. Die damit verbundenen Modifikationen sollten jedoch mit vertretbarem Aufwand für den Entwickler erfolgen; ebenso sollte die Einarbeitungszeit in akzeptablen Grenzen bleiben. Es fiel daher die Entscheidung, ein am Markt erhältliches CASE-Tool für das Re-Engineering des Managementagenten einzusetzen, das konform zur objektorientierten Analyse- und Designmethodik OMT ist. Ein weiteres Kriterium bestand darin, daß der Codegenerator des CASE-Tools in der Lage sein sollte, die Objektklassen in der OMG-Schnittstellenbeschreibungssprache *IDL (Interface Definition Language)* auszugeben, damit diese anschließend von der CORBA–Entwicklungsumgebung weiterverarbeitet werden können. Unsere Anforderungen konnten schließlich vom kommerziellen CASE-Tool *Software through Pictures* [14] erfüllt werden, mit dessen Unterstützung die Optimierungen am Objektmodell durchgeführt wurden. Hierbei ist hervorzuheben, daß für die unterschiedlichen Phasen der Softwareerstellung leistungsfähige Editoren bestehen, deren Navigationsmöglichkeiten den zyklischen Prozeß der Analyse und des Designs berücksichtigen ([10]). Die Erstellung graphischer Modelle und deren „Nachbearbeitung" in Tabellen wurde ebenso unterstützt wie die Dokumentation des Projektes. Features wie die Möglichkeit, Attributen bereits bei der Modellierung Eigenschaften wie „nur lesbar" oder Default-Werte zuzuweisen, haben sich beim Übergang zur Implementierung ebenfalls als vorteilhaft erwiesen. Insbesondere hilfreich war die übersichtliche Darstellung des vollständigen Objektmodells; die vergleichbare SNMP-MIB umfaßte demgegenüber 35 Seiten.

3.2 Optimierung des Objektmodells

Aus den am Anfang dieses Abschnitts gemachten Beobachtungen ließen sich folgende Regeln für die Optimierung des Objektmodells ableiten:

- In mehreren Klassen vorkommende identische Attribute und Operationen werden in einer (ggf. neu einzuführenden) Superklasse zusammengefaßt.
- Typenbezeichner werden entfernt. An die Stelle dieser Bezeichner treten neue Unterklassen.
- Pushbutton-Variablen werden zu Methoden der jeweiligen Klasse.
- Möglichst realitätsgetreue Modellierung von Objektbeziehungen; Einführung objektorientierter Hierarchien (Vererbung und Enthaltensein).
- Definition neuer Variablentypen für Attribute mit bestimmten Wertebereichen (z.B. Aufzählungstypen).

Die Konsequenzen der Anwendung dieser Regeln sind nachfolgend detailliert beschrieben.

3.2.1 Neue Superklassen für gemeinsame Attribute und Operationen

Auffällig an der ersten Version des Objektmodells ist, daß mehrere Klassen Attribute mit identischer Bedeutung (aber zum Teil abweichender Bezeichnung) hatten. Stattdessen bot es sich an, gemeinsame Attribute in eine Oberklasse aufzunehmen, betreffende Klassen von dieser Oberklasse abzuleiten und dafür in den Unterklassen die gemeinsamen Attribute zu streichen. Typische Beispiele für gemeinsame Attribute mehrerer Klassen sind Identifikatoren, Namensbezeichnungen und Statusvariablen. Diese traten u.a. in den Objekten `Printer`, `Storage` und `Processor` auf. Anstatt nun eine neue Oberklasse einzuführen, wurde der Klasse `Device` (umbenannt in `Generic_Device`) die Rolle der Oberklasse zugewiesen. Damit wurde die

Aufgabe der `Device`-Klasse geändert. Sie dient nun als Wurzel der Vererbungshierarchie mehrerer Systemkomponenten. Dies erleichtert auch spätere Erweiterungen des Objektmodells, bei denen eine neue Systemkomponente von `Generic_Device` abgeleitet werden kann, wodurch schon eine gewisse Grundfunktionalität bereitgestellt werden kann (und zwar die, die in den meisten Komponenten gleich ist).

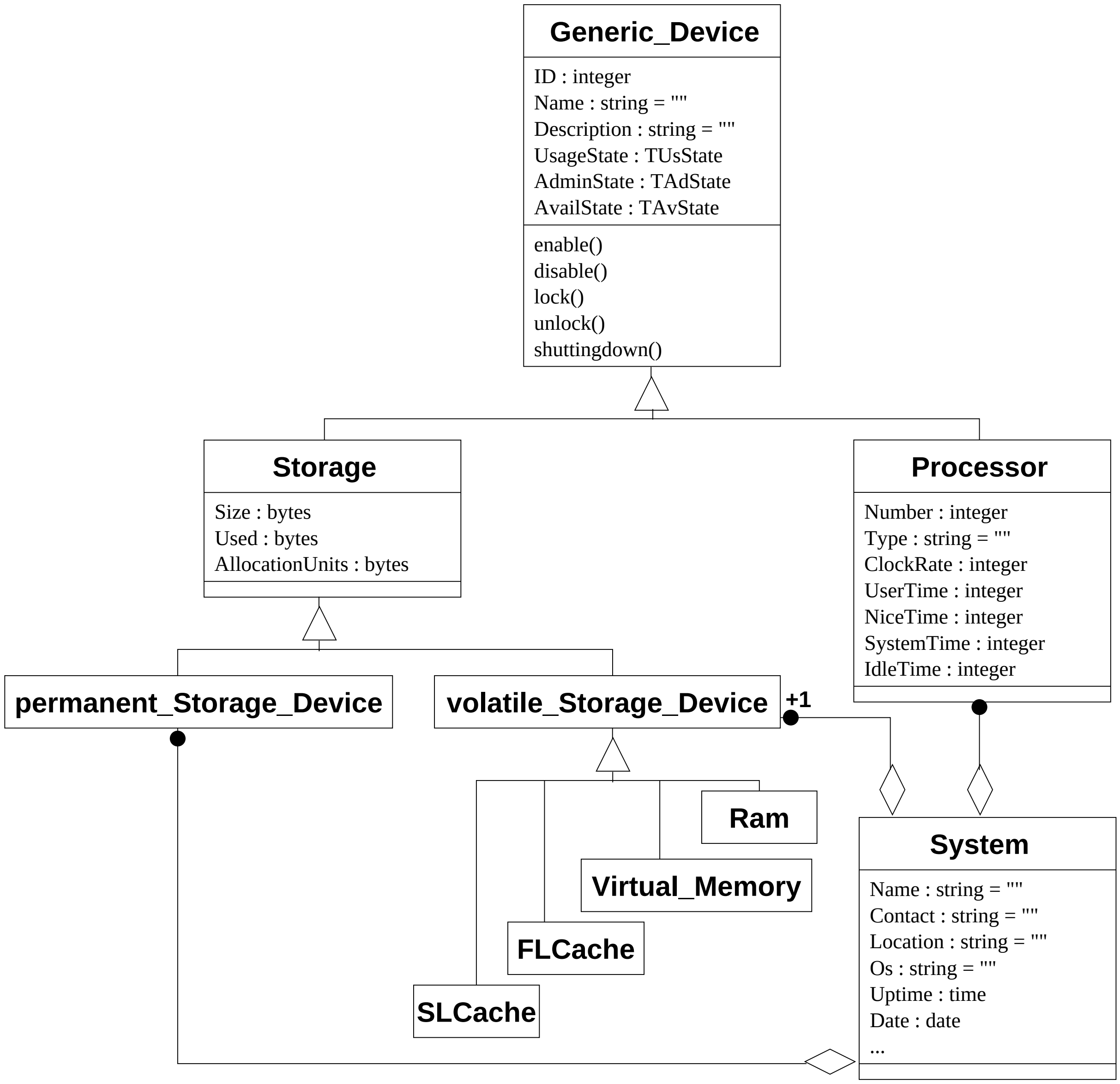

Abbildung 4: Optimiertes Objektmodell für das Workstation-Management (Ausschnitt)

3.2.2 Neue Unterklassen anstelle von Typvariablen

Dieser Optimierungsansatz ist die Antwort auf den zweiten Kritikpunkt an der ersten Version des Objektmodells. Durch vorhandene Typvariablen war es zwar möglich, Objekte verschiedener Typen einer Klasse zu instantiieren. Dies steht jedoch in klarem Widerspruch zum objektorientierten Klassenkonzept, wonach alle Instanzen einer Objektklasse dieselben Eigenschaften aufweisen sollten. Es war innerhalb des vorliegenden Objektmodells also nicht möglich, verschiedenen Typen einer Objektklasse unterschiedliche Eigenschaften (Attribute, Operationen etc.) zuzuweisen; jedes Objekt der Klasse (gleich welchen Typs) hätte den gleichen Aufbau

gehabt. So konnte man zwar Objekte verschiedenen Typs der Klasse `Device` erzeugen (durch entsprechendes Belegen der Variablen `Type`); für jede dieser Komponenten hätten dann jedoch nur drei Variablen (`Index`, `Description` und `State`) zur Verfügung gestanden, die, separat betrachtet, nur über beschränkte Aussagekraft verfügen. Dies war ein weiterer Grund, weshalb die virtuelle d.h. nicht instantiierbare Klasse `Generic_Device` zur Wurzel des Vererbungsbaumes gemacht wurde (siehe auch 3.2.1). Soll nun ein Objekt für eine Systemkomponente erzeugt werden, so wird nicht die `Generic_Device`-Klasse instantiiert, die nun als Containerklasse für allgemeingültige Attribute dient, sondern eine entsprechende Subklasse.
Ähnlich verhielt es sich mit der `Storage`-Klasse. Hier konnten zwar die Typen `Ram`, `Virtual-Memory`, `FLCache` und `SLCache`[5] erzeugt werden; es war jedoch nicht möglich, den einzelnen Typen auch unterschiedliche Eigenschaften zuzuordnen. Deshalb wurden die neuen Klassen `permanentStorageDevice` und `volatileStorageDevice` eingeführt. Dabei fielen unter die erste Subklasse Komponenten wie Festplatten, Magnetbänder oder Wechselfestplatten. Die zweite Subklasse sollte die Komponenten umfassen, die ursprünglich über die `Storage`-Klasse instantiiert wurden. Als Folge davon wurden die Subklassen `Ram`, `VirtualMemory`, `FLCache` und `SLCache` eingeführt.

3.2.3 Operationen anstelle von Pushbutton-Variablen

Das Problem der sogenannten Pushbutton-Variablen, welches in der SNMP-MIB bestand, wurde auch in die erste Version des Objektmodells (siehe Abbildung 3) übernommen: Eine Operation auf einem Objekt (und damit auf der entsprechenden Systemkomponente) mußte dadurch ausgelöst werden, daß einem entsprechenden Objektattribut ein Wert zugewiesen wurde. Hinsichtlich der Erkennbarkeit der Semantik einer Operation ist dies generell fragwürdig, da auf den ersten Blick nicht zwischen einem normalen Wertattribut und einem Pushbutton-Attribut unterschieden werden kann. Beim objektorientierten Ansatz ist dieses Problem jedoch sehr leicht zu lösen: An die Stelle der Pushbutton-Variablen traten Operationen. Für jeden möglichen Wert einer Pushbutton-Variablen wurde dabei eine eigene Operation eingeführt. Ein Beispiel hierfür ist das Attribut `cpuAction` der `Processor`-Klasse. Für die fünf Werte, die dieser Variable ursprünglich zugewiesen werden konnten, wurden die entsprechenden Operationen enable(), disable(), lock(), unlock() und shuttingdown() eingeführt. Die bisherige Wertzuweisung cpuAction:=1, ausgeführt durch das Senden einer SNMP `set`-Protokolldateneinheit an den Agenten, entspricht nun einem Aufruf der Operation enable(). Da diese fünf Operationen in vielen Komponenten eines Systems ebenfalls auftreten, wurden auch sie in die Klasse `Generic_Device` aufgenommen (siehe Abbildung 4).

3.2.4 Möglichst realitätsgetreue Modellierung der Objektbeziehungen

Hinsichtlich der Beziehungen zwischen Klassen weist die erste Version der Objektmodells dieselben Defizite auf, wie die SNMP-MIB: Es ist keinerlei Vererbungshierarchie vorhanden; Containment-Hierarchien sind nur ansatzweise vorhanden. Ein Grundgedanke des objektorientierten Paradigmas ist es jedoch, die reale Welt, d.h. die einzelnen Objekte sowie ihre Beziehungen zueinander, möglichst gut abzubilden. Die Einführung einer Vererbungshierarchie ist ein Modell für den Sachverhalt, daß eine Systemkomponente eine Spezialisierung einer anderen ist. So ist z.B. ein Mikroprozessor eine Spezialisierung eines Gerätes (`Device`).
Zur Wurzel des Enthaltenseins- oder Containment-Baumes wurde die Klasse `System` bestimmt. Dies entspricht auch der Realität: Ein (End-)System ist diejenige Hauptkomponente, die andere Teilkomponenten enthält. Dabei sollten Enthaltenseinsbeziehungen der Objektklasse `System` mit möglichst spezialisierten Klassen bestehen, also Klassen, die sich im Vererbungsbaum „unten“ befinden. Damit können die einzelnen Beziehungen individuell gestaltet werden: So benötigt ein System mindestens einen Prozessor (hier also eine 1:n-Beziehung mit $n \geq 1$), jedoch kann es beliebig viele `permanentStorageDevices` besitzen (in diesem Fall eine 1:n-

[5]**F**irst **L**evel **Cache** beziehungsweise **S**econd **L**evel **Cache**

Beziehung mit n≥0). Ein Drucker wiederum ist kein unmittelbarer Bestandteil eines Systems (im Sinne einer Workstation), sondern ein Peripheriegerät. Hier besteht also keine Aggregationsbeziehung, sondern eine einfache 1:n-Beziehung mit n≥0. Wäre die Beziehung zwischen System und „höher" liegenden Klassen modelliert worden, so wäre eine individuelle Gestaltung nicht möglich gewesen, da jede Subklasse dieselbe Beziehung zu `System` wie die entsprechende Oberklasse gehabt hätte. Abbildung 4 gibt einen Überblick über die Beziehungsstruktur eines Teils des optimierten Objektmodells.
Eine weitere Besonderheit findet sich in der Beziehung zwischen den Klassen `Filesystem` und `Account` (vormals die `User`-Klasse). Die Klasse `Account` enthielt in der SNMP-MIB Attribute, die benutzerspezifische Quoten für Betriebsmittel wie zum Beispiel Plattenplatz darstellen. Dies war semantisch eigentlich nicht korrekt, da eine Quota keine Eigenschaft eines Benutzers ist, sondern eine Eigenschaft der Beziehung zwischen einem Benutzer und einem Dateisystem. Aus diesem Grunde wurden die entsprechenden Attribute ausgelagert und unter der Assoziationsklasse `Quota` zusammengefaßt. Abbildung 5 veranschaulicht dies.

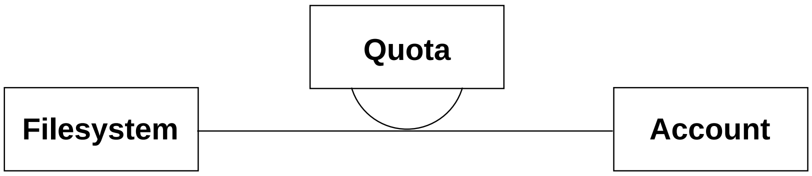

Abbildung 5: Anwendung von OMT-Assoziationsklassen

3.2.5 Neue Variablentypen für Variablen mit eingeschränktem Wertebereich

Die algorithmische Übersetzung der SNMP-MIB hatte nichts daran geändert, daß die meisten Attributtypen im Objektmodell einfache ASN.1-Grundtypen waren. Dies war in vielen Fällen problematisch: So hat die Variable `OpState` in der Objektklasse `Processor` den Datentyp integer, obwohl sie eigentlich nur die Werte 1 (für enabled) und 2 (für disabled) annehmen darf. Es wäre nun im Rahmen der Optimierung naheliegend gewesen, den Datentyp dieser Variablen auf boolean zu setzen. Es erwies sich jedoch als besser, einen (gleichwertigen) Aufzählungstyp zu definieren, der sofort erkennen läßt, in welchem Nutzungszustand sich ein Gerät befindet (enabled oder disabled).
Ein weiteres Beispiel für die Einführung eines neuen Datentyps ist die Variable `AdminState` (vorher `AdState`). Hier wurde (aus denselben Gründen wie oben) ein Aufzählungstyp definiert, der die Administrationszustände *unknown*, *unlocked*, *shutting down* und *locked* zuläßt.

4 Implementierung in CORBA

Da als Implementierungsarchitektur CORBA vorgesehen war, wurden zunächst die dem Objektmodell entsprechenden IDL-Schnittstellenbeschreibungen vom CASE-Tool generiert. Als Beispiel folgt in Listing 1 ein Auszug aus der Schnittstellenbeschreibung der Objektklasse `System`.
Die Ausgabe verdeutlicht, daß alle mit dem CASE-Tool angegebenen Einstellungen (readonly, Datentypen u.ä.) in die IDL-Ausgabe übernommen werden. An obiger Ausgabe ist auch gut zu erkennen, wie Beziehungen aus dem Objektmodell in IDL-Schnittstellenbeschreibungen abgebildet werden. Dabei stellte sich heraus, daß Beziehungen nur mangelhaft in IDL wiedergegeben werden.

```
{...}
// stp class definition 108
interface System
{
// stp class members
attribute string Contact;                    // einfache les-
attribute date Date;                         // und schreibbare
attribute string Hardware;                   // Attribute
attribute string Location;
attribute string Name;
readonly attribute string Os;                // nur lesbare
readonly attribute time Uptime;              // Attribute
readonly attribute long maxProcessNumber;
readonly attribute long maxProcessSize;
attribute sequence<Printer> assnPrinter;     // 1:n Assoziation
attribute Process assnProcess;               // einfache Assoz.
attribute sequence<Processor> aggrProcessor; // 1:n Aggregation
...
};
```

Listing 1: IDL-Schnittstellenbeschreibung der Objektklasse `System` (Auszug)

4.1 Semantische Nachbesserungen bei der Generierung von IDL-Schnittstellen

Die Generierung gibt die Beziehungen zwischen Objektklassen zwar korrekt wieder; folgendes Problem ergibt sich jedoch durch die Beschreibung von Beziehungen durch Attribute: Wird ein Objekt einer Klasse instantiiert, welche in mehreren anderen Klassen durch entsprechende Attribute als assoziiert gekennzeichnet ist, kann es leicht zu Inkonsistenzen kommen, wenn das Objekt z.B. in der sequence eines Objektes zwar enthalten ist, in einem anderen Objekt jedoch nicht.
Außerdem mußten neue Methodenschnittstellen eingefügt werden, die die Gültigkeit der Beziehungen periodisch überwachen. So überprüft beispielsweise die Methode der `System`-Objektklasse update_Processes(), welche Prozesse momentan aktiv sind. Für die Methoden, die ausschließlich der Verwaltung von Beziehungen zwischen Objektklassen dienen, mußten daher neue Methoden entworfen und implementiert werden. Dies resultiert aus der Tatsache, daß der OMG *Object Relationship Service* [9] zwar spezifiziert wurde, aber noch nicht Bestandteil der CORBA-Entwicklungssysteme ist. Eine zukünftige Version des CORBA-Agenten wird diesen Dienst nutzen.
Bezüglich des Erzeugens und Löschens von Objekten bzw. der Objektverwaltung allgemein ergibt sich ein weiteres Problem: Es besteht keine zentrale Komponente (eine sogenannte *factory*), über die Objekte einer Klasse erzeugt, gelöscht oder zum Beispiel aufgelistet werden könnten. Eine Lösung für dieses und das obige Problem war die Einführung von Metaklassen. Zu jeder Klasse existierte damit eine weitere Klasse, von der allerdings nur ein einziges Objekt instantiiert wird und Funktionen bereitstellt, die der Verwaltung von Objekten der Hauptklasse dienen. Da diese Überlegungen rein implementierungsbedingt sind, wurde darauf verzichtet, die Metaklassen in das Objektmodell aufzunehmen.
Ein Fall, in dem der Nutzen von Metaklassen auf eine andere Art zum Vorschein kommt, trat im Zusammenhang mit der Prozeß-Klasse auf: Generell gilt, daß die CORBA-Systemmanagement-Objekte beim Systemstart instantiiert werden sollten. Bei statischen Objekten bzw. Objekten, die nur über eine geringe Änderungsdynamik verfügen (wie z.B. Prozessoren, Festplatten) ist dies auch kein Problem. Anders verhält es sich aber mit dynamischen Objekten, wie zum Beispiel bei der Überwachung von Prozessen. Es ist nahezu unmöglich, diese Objekte mit dem realen Systemzustand konsistent zu halten. Dazu hätte bei jedem Start eines Prozesses ein entspre-

chendes Prozeßobjekt instantiiert und bei Prozeßterminierung wieder gelöscht werden müssen. Durch den Zugriff auf die Prozeßobjekte über eine sogenannte „before/after"-Metaklasse MetaProcess, konnte dieses Problem gelöst werden. Sobald nun ein Zugriff auf die Metaklasse durchgeführt wird, wird der Bestand an Prozeßobjekten aktualisiert; das anfragende Objekt (in diesem Fall das Managementsystem) erhält also beim Zugriff immer den aktuellen Systemzustand.

4.2 Syntaktische Nachbearbeitung der generierten IDL-Schnittstellen

Als CORBA-Entwicklungsumgebung wird das IBM *SOMobjects Developer Toolkit* [4] eingesetzt; es umfaßt einen CORBA 1.2-konformen Object Request Broker, einen IDL-Compiler, die CORBA-Laufzeitumgebung und mehrere zu den OMG-Standards konforme Dienste [13].
Die vom CASE-Tool erzeugten IDL-Schnittstellenbeschreibungen konnten nicht unmittelbar als Eingabe für den SOM IDL-Compiler eingesetzt werden, da Nachbearbeitungen an folgenden Stellen erforderlich waren:

- **Definition von Attributtypen**
  Datentypen, die nicht zu den IDL-Grundtypen gehören, müssen, sofern sie nicht in der Attributdeklaration festgelegt sind, gesondert definiert werden. Weiter ist zu beachten, daß der SOM IDL-Compiler zwar Sequenzen kennt, oft jedoch nicht den Datentyp, über dem die Sequenz definiert wurde. Dem Auftreten von `sequence⟨Processor⟩` mußte die Definition der Objektklasse `Processor` vorangehen.

- **Ableitung aller IDL-Interfaces von SOMObject**
  Die Konventionen des SOM IDL-Compilers verlangen, daß alle auftretenden Schnittstellen von der Objektklasse SOMObject abgeleitet werden.

- **Ergänzung aller IDL-Dateien um eine „Implementation Section"**
  Hierbei mußte dem IDL-Compiler mitgeteilt werden, in welche DLL[6] die spätere Ausgabe geschrieben werden soll. Außerdem mußten noget- oder noset-Anweisungen für Nur-Lese- bzw. Schreib-Lese-Attribute eingefügt werden, um zu verhindern, daß der Compiler automatisch interne get- oder set-Operationen erzeugt, die den entsprechenden Attributwert liefern bzw. setzen. Dies war notwendig, da der bereits existierende Code eingebunden werden sollte.

4.3 Übernahme bestehenden Agentencodes

Nachdem die IDL-Beschreibungen der Objektklassen mit ihren Attributen und Methoden erzeugt und an das CORBA-Entwicklungssystem angepaßt waren, mußte nun in einem weiteren Schritt die Umsetzung der Funktionalität erfolgen, die die Nutzung der Managementinformation erst ermöglicht.
Dies ist gleichbedeutend mit der in zahlreichen Bereichen der Informatik auftretenden Fragestellung, wie bestehende Altsysteme (*legacy systems*) schonend in die objektorientierte Welt migriert werden können. Zentraler Gedanke ist hierbei, unter Zuhilfenahme sogenannter *Wrapper* bestehenden Code geeignet in Objektklassen zu kapseln und somit für neue objektorientierte Systeme zugreifbar zu machen. Der betrachtete SNMP-Agent ist in der Programmiersprache C implementiert worden und genügt damit keinesfalls objektorientierten Prinzipien. Dies ist jedoch für die Migration ohne Bedeutung, da für objektorientierte Applikationen der Begriff des „Perception is reality" gilt. Es ist also nicht erforderlich, daß ein System objektorientiert implementiert worden ist; maßgeblich ist jedoch, daß seine Bestandteile wie Objekte aussehen. Sehr wichtige Voraussetzungen für eine Kapselung in hinreichend feiner Granularität sind, daß das zu migrierende System genügend modular implementiert worden ist und die Schnittstellen seiner Prozeduren offengelegt sind bzw. die Prozeduren im Quellcode vorliegen. Beides war in

[6] DLL: **D**ynamic **L**ink **L**ibrary

unserem Fall gegeben (siehe 2.1).
Die IDL–Schnittstellenbeschreibungen werden nun einem IDL–Compiler übergeben, der zum einen die Schnittstellen der neu erstellten Objektklassen (hier: die *Wrapper* für die bereits bestehenden Prozeduren) innerhalb des ORB–Laufzeitsystems bekanntmacht und zum anderen die Datenstrukturen und Schnittstellen des CORBA–Agenten in einer herkömmlichen Programmiersprache (im betrachteten Fall: C) generiert. Momentan sind Algorithmen zur Übersetzung (sogenannte *Language Mappings*) der Quellsprache IDL in die Zielsprachen C, C++ und Smalltalk durch OMG–Standards spezifiziert; Abbildungen für ADA, COBOL und Java sowie die Skriptsprache Perl sind geplant.
Die so entstandenen C-Programmrümpfe des CORBA–Agenten können nun um die entsprechenden Aufrufe von Prozeduren des bestehenden SNMP–Agenten ergänzt werden, deren Aufgabe lediglich darin besteht, die erhaltenen Parameter auf die Parameter der bestehenden Agentenprozeduren abzubilden und anschließend diese Prozeduren aufzurufen. Hierbei ist es von großem Vorteil, daß der Agent modular aufgebaut ist und die Schnittstellen des Agenten sowohl zu den Ressourcen als auch zum Managementprotokoll hin klar definiert sind.

5 Zusammenfassung und Ausblick

Der Beitrag hat anhand eines praxisnahen Beispiels aufgezeigt, welche Schritte erforderlich sind, um aus bestehendem modularen Code eines SNMP-Agenten kooperierende CORBA-Managementobjekte zu gewinnen. Das in diesem Beitrag vorgestellte Vorgehensmodell stellt einen universell verwendbaren, systematischen Ansatz zum Re-Engineering bestehender Managementagenten dar. Drei kritische Erfolgsfaktoren konnten hierbei identifiziert werden: Der modulare Aufbau des zu portierenden Agenten, die Abstützung auf standardisierte Verfahren zur Transformation der Managementinformation sowie eine gute Werkzeugunterstützung durch CASE-Tools, die State-of-the-art–Modellierungstechniken (wie z.B. OMT) instrumentieren.
Die Tatsache, daß momentan noch syntaktische Divergenzen zwischen dem von CASE-Tools erzeugten und von CORBA-Entwicklungssystemen akzeptierten IDL-Schnittstellenbeschreibungen bestehen, ist angesichts des geringen Alters von CORBA verständlich; dies sind vielmehr natürliche Reibungsverluste bei der Kombination von frei am Markt erhältlichen Produkten, die sich zum Teil noch in einem sehr frühen Stadium befinden. Unter diese Kategorie fallen auch die angesprochenen Probleme bei der Überwachung der Gültigkeit von Beziehungen; sobald geeignete, bereits spezifizierte CORBAservices Bestandteil von kommerziell erhältlichen Entwicklungssystemen sind, wird sich eine zukünftige Version unserer Implementierung darauf abstützen können.
Ebenso konnte die Tragfähigkeit von CORBA für die Belange des Systemmanagements nachgewiesen werden, selbst wenn heutige CORBA-Entwicklungsumgebungen noch nicht die für den Produktionsbetrieb erforderliche Leistungsfähigkeit und Skalierbarkeit besitzen: Unzulänglichkeiten heutiger CORBA-Toolkits beruhen beispielsweise auf der Implementierung des *Interface Repository* in Form einfacher Dateien, deren Verzeichnisse von den Maschinen, auf denen der Object Request Broker läuft, wechselseitig über das *Network File System* exportiert bzw. gemountet werden müssen. Hierdurch entstehen massive Einbrüche der Performance.
Zweifellos sind mit dem derzeitigen Stand des Projektes noch nicht alle Möglichkeiten ausgeschöpft, die moderne OOA/OOD-Methoden bieten, da bisher das (statische) Objektmodell von Managementagenten im Mittelpunkt der Betrachtungen stand. Weitere Schritte bestehen daher in der Untersuchung und Modellierung der dynamischen Aspekte von Managementagenten sowie der Potentiale möglicher Delegierbarkeit von Managementaufgaben zur Laufzeit durch Managementsysteme an Managementagenten.

Danksagung
Der Autor dankt dem Münchner Netzmanagement Team für intensive Diskussionen zu früheren Versionen dieses Beitrags. Das MNM-Team, das von Prof. Hegering geleitet wird, ist eine Gruppe von Wissenschaftlern beider Münchner Universitäten und des Leibniz-Rechenzentrums der Bayerischen Akademie der Wissenschaften.

Literatur

[1] Case, J., McCloghrie, K., Rose, M., Waldbusser, S.: Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1902. IAB. (Januar 1996)

[2] The Common Object Request Broker: Architecture and Specification. OMG Specification Revision 2.0. Object Management Group. (Juli 1995)

[3] Gutschmidt, M., Neumair, B.: Integration von Netz- und Systemmanagement: Ziele und erste Erfahrungen. In: *Proceedings der 3. Fachtagung Arbeitsplatzrechensysteme (APS'95), Hannover*. Mai 1995

[4] SOMobjects: A Practical Introduction to SOM and DSOM. IBM Corporation, International Technical Support Organization. Research Triangle Park, NC 27709-2195, Juli 1994. Order Number: GG24-4357-00

[5] IBM Systems Monitor: Anatomy of a Smart Agent. IBM Corporation, International Technical Support Organization. Research Triangle Park, NC 27709-2195, Dezember 1994. Order Number: GG24-4398-00

[6] Mowbray, T. J., Zahavi, R.: The Essential CORBA - Systems Integration using Distributed Objects. John Wiley & Sons, Inc. 1995

[7] SunSoft's NEO Product Family. Product Overview. SunSoft Inc. (März 1996). http://www.sun.com/sunsoft/neo/external/whitepapers/FamilyNEO.ps

[8] HP OpenView Operations Center Concepts Guide. User Manual. Hewlett Packard. (1993)

[9] CORBAservices: Common Object Services Specification, Volume 1. OMG Specification. Object Management Group. (März 1996)

[10] Poston, R. M.: Automated from Object Models. Communications of the ACM, (September 1994)

[11] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice-Hall International, Inc. 1991

[12] Rutt, T.: Comparison of the OSI Management, OMG and Internet Management Models. A report of the Joint X/Open-NM Forum Inter-Domain Management Task Force. AT&T Bell Laboratories. (März 1994)

[13] SOMobjects Developer Toolkit Programmer's Guide Volume 2: Object Services. IBM Corporation. März 1996. First Edition

[14] Software through Pictures Technical White Paper. Interactive Development Environments. San Francisco, CA 94105, 1995

[15] Tivoli TME 10. IBM Announcement Letters (US) - Document '296-216. (Juni 1996). http://www1.ibmlink.ibm.com/PS/ALET/296216.PS

[16] Inter-Domain Management Specifications: Specification Translation (Final Sanity Check Draft). Preliminary Specification Pxxx. X/Open Ltd. (September 1996)